ISBN 978-88-904388-0-6

Angelo Gargantini (Ed)

Eclipse-IT 2009

4th Italian Workshop on Eclipse Technologies

Bergamo, September 28 -29, 2009
Università degli Studi di Bergamo

Eclipse-IT 2009

Angelo Gargantini (Ed.)

Eclipse-IT 2009

4th Italian Workshop on Eclipse Technologies

Proceedings

Bergamo, September 28-29, 2009
Università degli Studi di Bergamo

This volume is edited by
Angelo Gargantini
Dipartimento di Ingegneria dell'Informazione e
Metodi Matematici
Università degli Studi di Bergamo
viale Marconi, 5 - 24044 Dalmine BG - Italia
http://cs.unibg.it/gargantini/
email: angelo.gargantini@unibg.it

Eclipse Italian Community
http://eclipse.dis.unina.it/
Conference web site:
http://eit09.unibg.it/

Printed in Bergamo, Italy
September 2009
ISBN: 978-88-904388-0-6

Cover designed by Emanuele Piana

Preface

This volume contains the papers presented at EclipseIT 09, the *4th Italian Workshop on Eclipse Technologies* held on September 28th and 29th in Bergamo. The previous three EclipseIT workshops took place in Rome (2006), Naples (2007), and Bari (2008).

The goal of EclipseIT is to bring together industry and academia, students and professors, researchers and practitioners interested in the Eclipse technologies, their use and their possible future developments. Eclipse is a very good example of a technology in which the open source community and the business community, universities and companies can work together to bring better products to their students and to their costumers.

Indeed, Eclipse was initially designed as an integrated development environment (IDE) for Java or at most for object-oriented application development. Today Eclipse is an open source community whose projects are focused on building an open development platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across its lifecycle. Around Eclipse there is a live community which continuously works in experimenting, extending, and supporting the Eclipse platform.

This year workshop, which gathers the whole (from South to North) Italian Eclipse community, reflects the viability and richness of the community. We had more than 70 registered participants from every part of Italy. There were 10 submissions of research papers and 8 of them were accepted for publication in the research track. We had a student track with 6 presentations and demos. This year workshop has also featured an industrial track with 9 presentations from industrial partners and final users of Eclipse. For the first year, we accepted also contributions about the Jazz technology.

This year program also includes 2 invited talks. The first one is delivered by Ralph Mueller (Director of Eclipse Foundation, Ecosystem - Europe) who presents his view on *Open source for Business*. The second one is given by Ferdinando Gorga (IBM Rational Italy) and Scott Rich (Jazz Foundation PMC Lead) and it is a tutorial on *Jazz: the modern platform for software engineering tools*.

I would like to thank the authors who submitted their papers, the members of the Program Committee and the additional reviewers for their very precise reviews and the entire Eclipse Italian Community chaired by Paolo Maresca who supported this workshop. Special thanks go to the sponsors of this event, IBM Italy with Carla Milani, the Faculty of Engineering and the Department of Information Technology and Mathematical Methods of the University of Bergamo, the Eclipse Foundation, and Opera21. I thank the student track chair, Stefano Paraboschi, who carefully scrutinized all the student papers, and the industrial track chair Domenico Squillace, who was able to advertise this event among industrial users and developers. Thanks also to Patrizia Scandurra and Elvinia Riccobene who helped me in organizing this year workshop.

Bergamo, September 2009 Angelo Gargantini

Conference Organization

Program Chairs

Workshop Chair	Angelo Gargantini, Università di Bergamo
Publicity Chair	Paolo Maresca, Università Federico II di Napoli
Student Track Chair	Stefano Paraboschi, Università di Bergamo
Industrial Track Chair	Domenico Squillace, IBM Italia

Programme Committee

Marco Aimar	Alessandro Assab
Luciano Baresi	Marco Brambilla
Fabio Calefato	Andrea Calvagna
Andrea De Lucia	Arcelli Francesca
Giacomo Franco	Rosario Gangemi
Filippo Lanubile	Enrico Oliva
Alfonso Pierantonio	Elvinia Riccobene
Patrizia Scandurra	Giuseppe Scanniello
Vittorio Scarano	Carmine Seraponte
Domenico Squillace	Rodolfo Totaro

Local Organization

Patrizia Scandurra
Paolo Arcaini

External Reviewers

Davide Di Ruscio
Carmine Gravino

Sponsors

IBM Italia

Università degli Studi di Bergamo

Eclipse Foundation

Opera21

Dipartimento di Ingegneria
dell'Informazione e Metodi Matematici

Table of Contents

Open source for Business

Ralph Müller

Eclipse Foundation
Director, Ecosystem - Europe

Open Source- for many the expression is a synonym for 'no cost' or 'easy to obtain'. Only known to the insider is the fact that Open Source methods and licenses have become the basis of many industry collaboration efforts. Here Open Source is applied to jointly develop, maintain and distribute software relevant for the industry. Due to its well-defined Intellectual Property Management and its Development Process is the Eclipse Project today well positioned to support industry initiatives in the collaborative value creation process. The commerce-friendly Eclipse Public License (EPL) makes it easy to capture value in commercial exploitation, providing the basis for a successful and sustainable eco system. The talk will present examples of industry collaborations at Eclipse. We will look at the Eclipse Mobile Working Group (Pulsar) as well as the current status of the automotive E/E tools and Silicon Tools initiative in the mebedded idustry.

Biography

Ralph Müller started working for the Eclipse Foundation in September 2005. A computer scientist by trade, he sees the main focus of his work in the exploration and building of the European Eclipse Eco System. In addition to this, he is one of the main contact points in Central Europe for companies and individuals that want to use Eclipse, provide services around Eclipse or contribute to Eclipse. Based on his past experience, he is also exploring collaboration with industries such as Automotive or Aerospace with the goal to promote the Eclipse idea into these areas. Having been a software developer and a manager of software development teams in various companies, Ralph joined Object Technology International (OTI) in 1994, where he helped to found the European OTI organisation with its most prominent member, the OTI lab in Zurich - one of the cradles of Eclipse technology. Ralph then joined IBM, where he served as a lead architect for one of IBM's strategic initiatives for the Automotive industries.

Jazz: the modern platform for software engineering tools

Ferdinando Gorga[1] and Scott Rich[2]

[1] IBM Rational Software, Roma, Italy
[2] IBM in Research Triangle Park, North Carolina, USA

In this presentation we will speak about Jazz platform and its usage in software development. Jazz is the integration platform to enable IBM Rational tools to communicate. Main aspect of Jazz based tools is to provide a frictionless work environment that helps teams collaborate, innovate, and create great software. To that end, we are focusing on driving fundamental improvements in team collaboration, automation, and reporting across the software lifecycle.

***Collaborate*:** drive organizational consensus on priorities and improve workforce productivity

***Automate*:** Lower costs and improve quality by automating workflow based on real-time information

***Report*:** Continuously improve by measuring progress against desired business outcomes

Jazz platform enables development team to work in modern fashion. Jazz changes the points of views normally adopted working in a team:

- Lifecycle integration and not only Desktop integration,
- Team first and not function first
- Process automation and not manual process adoption only

These fundamentals will be explained showing some scenarios using tools: quality management activities, requirement and collaboration activities, development governance activities. We will see some demos about Rational Team Concert, Rational Quality Manager and Rational Requirement Composer.

Biography Ferdinando Gorga

Ferdinando Gorga is graduated in computer science in the Università di Salerno, with specialization in programming languages and software engineering. After several years working as designer and analyst in a lot of projects in areas Government, Finance, Telecomm, he joined Rational Software, working as technical representative. Since then, and after the acquisition of Rational by IBM, his role is to present and show the IBM Rational solutions in Italian companies involved in software development. Simultaneously also he is a University Ambassador, caring the relationship with Universities and improving the knowledge and the usage of the IBM Rational approach to systems and software development.

Biography Scott Rich

Scott Rich is a Distinguished Engineer and Master Inventor at IBM in Research Triangle Park, North Carolina, USA. He is the lead of the Project Management Committee for the Jazz Foundation Project, and a founding member of the Jazz project. Scott is a member of the core team of the IBM Tools Development Council, and was previously development lead for Rational Application Developer. He has worked at IBM for 22 years, holding a number of technical positions in that time on VisualAge for Smalltalk and Java, WebSphere Studio, and now Rational's team tools.

WebRatio BPM: a Tool for Design and Deployment of Business Processes on the Web

Marco Brambilla[1], Stefano Butti[2], Piero Fraternali[1]

[1] Politecnico di Milano, Dipartimento di Elettronica e Informazione
P.za L. Da Vinci, 32. I-20133 Milano - Italy
{marco.brambilla, piero.fraternali}@polimi.it
[2] Web Models S.r.l., I-22100 Como - Italy
stefano.butti@webratio.com

Abstract. This paper presents WebRatio BPM, an Eclipse-based tool that supports the design and deployment of business processes. The tool applies Model Driven Engineering techniques to complex, multi-actor business processes, mixing tasks executed by humans and by machines. Business processes are described through the standard BPMN 1.2 notation, extended with information on task assignment and escalation policies, activity semantics, and typed dataflows, to enable a two-step generative approach: first the Process Model is automatically transformed into a Web Application Model in the WebML notation, which seamlessly express both human- and machine-executable tasks; secondly, the Application Model is fed to an automatic transformation capable of producing the code. The tool provides various features that increase the productivity and the quality of the resulting application: one-click generation of a running prototype of the process from the BPMN model; fine-grained refinement of the resulting application; support of continuous evolution of the application design after requirements changes (both at business process and at application levels).

1 Introduction

Business process languages, such as BPMN (Business Process Management Notation) [8], have become the *de facto* standard for enterprise-wide application specification, as they enable the implementation of complex, multi-party business processes, possibly spanning several users, roles, and heterogeneous distributed systems. Indeed, business process languages and execution environments ease the definition and enactment of the business constraints, by orchestrating the activities of the employees and the service executions.

This paper presents an integrated approach and a supporting toolsuite to the specification, design and implementation of complex, multi-party business processes, based on a Model-Driven Engineering (MDE) methodology and on code generation techniques capable of producing dynamic Web applications from platform independent models.

The proposed approach is a top down one: the (multi-actor, multi-site) business process is initially designed in an abstract manner, using the standard

BPMN notation for schematizing the process actors, tasks, and business constraints. The resulting BPMN model is an abstract representation of the business process and cannot be used directly for producing an executable application, because it lacks information on essential aspects of process enactment such as: task assignment to humans or to Web Services, data flows among tasks, service invocation and user interface logics. Therefore, the standard BPMN specification is manually annotated with the missing information, to obtain a *detailed process model* amenable to a two-step transformation:

- A first model-to-model transformation (*Process to Application*) translates the detailed process model into: 1) a platform-independent model of the Web user interface and of the Web Service orchestrations needed for enacting the process, expressed in a Domain Specific Language called WebML [2]; 2) a Process Metadata Model, representing the business constraints (e.g., BPMN precedence constraints, gateways, etc).
- A second model-to-text transformation (*Application to Code*) maps the Application Model and the Process Metadata Model into the running code of the application. The resulting application is runtime-free and runs on any standard Java Enterprise Edition platforms.

The original contributions of the paper are: (i) an MDE forward-engineering methodology that starts from a high-level process model and progressively transforms it into the running code of the application, comprising Web Service interaction logics and Web User Interface logics; (ii) a refinement of the BPMN notation for augmenting the semantics of the Process Model and enable effective model transformation; (iii) a two-step generative framework transforming the refined Process Model into an Application Model, and for generating the executable code from the Application Model; and (iv) an extended version of the WebRatio toolsuite [11], called WebRatio BPM, that fully implements all the steps of the presented methodology.

The paper is organized as follows: Section 2 discusses the background technologies and notations; Section 3 discusses the approach to application development; Section 4 and Section 5 illustrate the extended process model and the application model, respectively; Section 6 describes the implementation of the WebRatio BPM tool; Section 7 discusses the related work; and Section 8 draws the conclusions.

2 Background: BPMN, WebML, and WebRatio

This work builds upon existing methods and tools to cover the different design phases.

BPMN [8] supports the high level specification of service choreographies, and allows one to visually specify actors, tasks, and constraints involved in a business process. Precedence constraints are specified by arrows, representing the control flow of the application, and gateways, representing branching and merging points

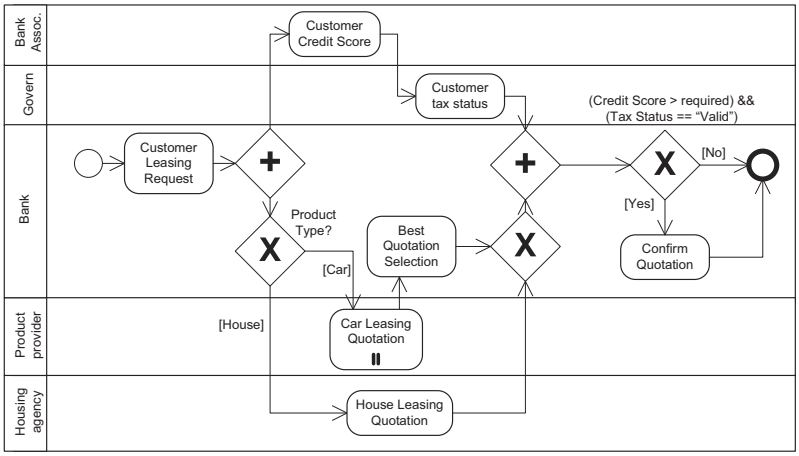


Fig. 1. Business process model of the leasing running example.

of execution paths. Parallel executions, alternative branches, conditional executions, events, and message exchanges can be specified. BPMN allows analysts to describe complex orchestrations of activities, performed by both humans and machines. Figure 1 shows an example of BPMN, describing a simplified leasing process for houses and cars.

WebML [2] is a Domain Specific Language for data-, service-, and process-centric Web applications [1]. It allows specifying the conceptual model of Web applications built on top of a data schema and composed of one or more hypertexts used to publish or manipulate data. The data model can be specified through standard E-R or UML Class diagrams. Upon the same data model, different hypertext models (*site views*) can be defined (e.g., for different types of users or devices). A site view is a graph of *pages*, consisting of connected *units*, representing data publishing components. Units are related to each other through *links*, representing navigational paths and carrying parameters. WebML allows specifying also update *operations* on the underlying data (e.g., the creation, modification and deletion of instances of entities or relationships) or operations performing arbitrary actions (e.g. sending an e-mail, invoking a remote service [6], and so on). Figure 2 shows a simple site view containing two pages, respectively showing the list of houses and a form for searching cars available for leasing. Page *Search Leasing Cars* contains an entry unit for inputting the car model to be searched, a scroller unit, extracting the set of cars meeting the search condition and displaying a sequence of result blocks, and a multidata unit displaying the cars pertaining to a block of search results. Besides Web applications, WebML can be used to specify Web services, Web service orchestrations, and the consumption of Web services by Web applications.

WebML is supported by the WebRatio CASE tool [11], which allows the visual specification of data models and site views and the automatic generation

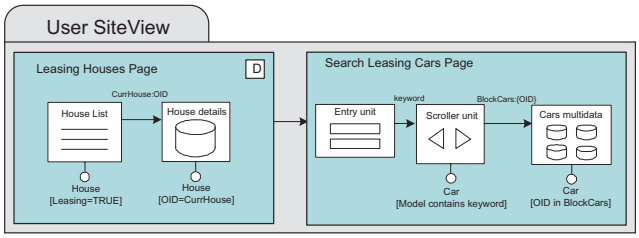


Fig. 2. WebML hypertext model example.

of J2EE code. The tool consists of a set of Eclipse plug-ins and takes advantage of all the features of this IDE framework. It also supports customized extensions to the models and code generators, model checking, testing support, project documentation, and requirements specifications.

The main features of WebRatio are the following: it provides an integrated MDE approach for the development of Web applications and Web services, empowered by model transformations able to produce the complete running code; it unifies all the design and development activities through a common interface based on Eclipse, which includes the visual editing of models, the definition of presentation aspects, and the extension of the IDE with new business components and code generation rules; it stores all the design artifacts into a common area (the Eclipse workspace) and manages them in a version control and collaborative work system; it can easily be extended with any existing Eclipse plugin (e.g., for Java component development, debugging, and so on).

3 Development Process

The development process supported by WebRatio BPM is structured in four main steps, represented in Figure 3.

Initially, business requirements are conceptualized in a high-level *Business Process Model*, which summarizes the needs of the stake-holders in a coarse BPMN schema. Figure 1 is an example of BPM that can be obtained as a requirement specification of the leasing application. Subsequently, the BPMN schema is refined by a BPM expert, who annotates it with parameters on the activities and data flows.

The resulting extended Process Model is subject to a first transformation, which produces the *Application Model* and *Process Metadata Model*. The Application Model (discussed in Section 5.2) specifies the details of the executable application according to the WebML notation, representing the hypertext interface for human-directed activities. The Process Metadata Model (discussed in Section 5.1) consists of a conceptual view (represented as a UML class diagram) of the activities of the process and of the associated constraints, useful for encapsulating the process control logic. This transformation extends and refines

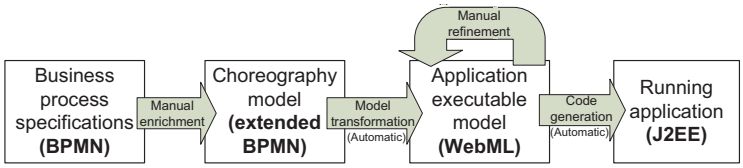


Fig. 3. Development process overview.

the technique for model-driven design of Web applications from business process specification initially proposed in [1]. Subsequently, the generated Application Model can be either executed as-is to get a first flavour of the application execution, or it can be refined manually by the designer, to add domain-dependent information on the execution of activities. Finally, the refined Application Model is the input of a second transformation, which produces the code of the application for a specific technological platform (in our case, J2EE); this step is completely automated thanks to the code generation facilities included in WebRatio.

4 Refined Process Model

The high-level BPMN process model designed in the requirement specification phase is not detailed enough to allow the generation of the application code. Its refinement is done using an extended BPMN notation, which enables a more precise model transformation into a WebML Application Model and then into the implementation code. In particular, the process model is enriched with information about the data produced, updated and consumed by activities, which is expressed by typed activity parameters and typed data flows among activities. Furthermore, activities are annotated to express their implicit semantics, and gateways (i.e., choice points) that require human decisions are distinguished.

Figure 4 shows the graphical notation of the extended BPMN activity. An activity is associated with a *Name* (1), which is a textual description of its semantics, and possibly an *Annotation* (2), which describes the activity behaviour using an informal textual description. An activity is parametric, and has a (possibly empty) set of input (3) and output (4) parameters. The actual values of

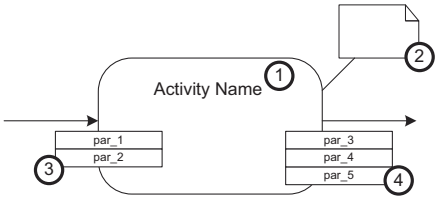


Fig. 4. Extended activity notation.

input parameters can be assigned from preceding activities; the output parameters are produced or modified by the activity. Analogous extensions are defined for gateways; these are further refined by specifying whether they are implemented as manual or as automatic branching/merging points. *Manual gateways* (tagged as Type "M") involve user interaction in the choice, while *Automatic gateways* (tagged as Type "A") automatically evaluate some condition and decide how to proceed with the process flow without human intervention. The output flow of gateways can be associated to a guard condition, which is an OCL Boolean expression over the values of the parameters of the gateway; the semantics is that the activity target of the flow link with the guard condition can be executed only if the condition evaluates to true.

5 Application Model

Starting from the Detailed Process Model presented above, an automatic transformation produces: (1) Process Metadata Model, describing the process constraints in a declarative way as a set of relations; (2) the Domain Model, specifying the application-specific entities; (3) and the Application Model, including both the site views for the user interaction and the service views for Web service orchestration.

Hence, the transformation consists of two sub-transformations:

- Detailed Process Model to Process Metadata: the BPMN precedence constraints and gateways are transformed into instances of a relational representation compliant to the Process Metamodel shown in Figure 5, for enabling runtime control of the process advancement;
- Detailed Process Model to Application Model: the BPMN process model is mapped into a first-cut Application Model, which can be automatically transformed into a prototype of the process enactment application or subsequently refined by the designer to incorporate further domain specific aspects.

Thanks to the former transformation, the BPMN constraints, stored in the Process Metadata Model, are exploited by the components of the Application Model for executing the service invocation chains and enacting the precedences among human-executed tasks.

5.1 Process Metadata Model Generation

Figure 5 shows, as a UML class diagram, the schema of the metadata needed for executing an BPMN process at runtime.

A *Process* represents a whole BPMN diagram, and includes a list of *Activities*, each described by a set of input and output *ParameterTypes*. A *Case* is the instantiation of a process, and is related to the executed *Activity Instances*, with the respective actual *Parameter Instances*. The evolution of the status history

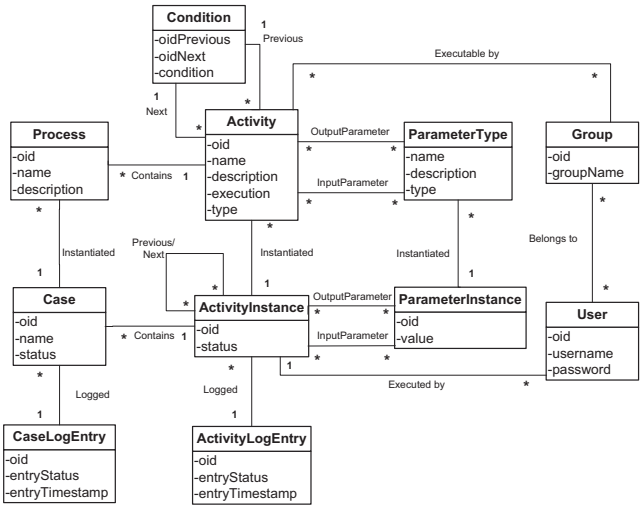


Fig. 5. Process Metadata describing the BPMN constraints.

is registered through *CaseLogEntry* and *ActivityLogEntry*. *Users* are the actors that perform a task and are clustered into *Groups*, representing their roles.

The transformation from the extended BPMN to the Process Metadata is a relational encoding of the BPMN concepts: each process model is transformed to a *Process* instance; each activity is transformed into an *Activity* instance; each flow arrow is transformed into a *nextActivity*/*previousActivity* relationship instance; each guard condition is transformed into a *Condition* instance.

5.2 Application Model Generation

The transformation from Refined Process Models to WebML coarse models of services and hypertext interfaces considers the type (human or automatic) of the gateways and the information on the data flows. The application models produced by the transformation still need manual refinement, to add domain-specific elements that cannot be expressed even in the enriched BPMN notation. However, by exploiting information about the activity type, a first-cut application model can be generated, which needs reduced effort for manual refinement.

The computation of the next enabled activities given the current state of the workflow is encapsulated within a specific WebML component, called *Next* unit, which factors out the process control logic from the site view or service orchestration diagram: the *Next* unit exploits the information stored in the Process Metadata to determine the current process status and the enabled state transitions. It needs the following input parameters: *caseID* (the currently executed process instance ID), *activityInstanceID* (the current activity instance ID), and the *conditionParameters* (the values required by the conditions to be evaluated). Given the *activityInstanceID* of the last concluded activity, the *Next*

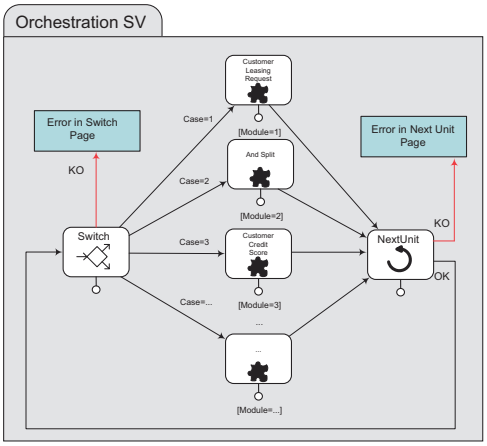


Fig. 6. WebML Orchestration Siteview.

unit queries the Process Metadata objects to find all the process constraints that determine the next activity instances that are ready for activation. Based on the conditions that hold, the unit determines which of its output links to navigate, which triggers the start of the proper subsequent activities.

The *Process to Application Model Transformation* from BPMN to WebML consists of two main rules: the *Process transformation rule*, addressing the structure of the process in-the-large; and the *Activity transformation rule*, managing the aspects of individual activities: parameter passing, starting and closing, and behavior. For modularity and reusability, the piece of WebML specification generated for each activity is enclosed into a WebML *module*, a container construct analogous to UML packages.

Figure **??** shows an overview of the outcome of the *Process transformation rule*: the hypertext page for manually selecting the process to be started and for accessing the objects resulting from process termination. This WebML fragment models the process wrapping logic, generated from the *Start Process* and *End Process* BPMN events.

The generated WebML model further comprises: (1) the orchestration site view, that contains the logic for the process execution; (2) a site view or service view for each BPMN pool; (3) a set of hypertext pages for each human-driven BPMN activity; (4) one service invocation (triggering the suitable actions for application data updates) for each automatic activity.

Figure 6 shows the model of the *orchestration site view*. The enactment of the process is performed through a loop of *WebML module* invocations, each representing the implementation of one of the activities.

The *Activity transformation* rule is based on the BPMN activity and gateway specifications, taking into account aspects like the actor enacting the activity (e.g., a human user or the system). For each BPMN activity and gateway, a

WebML module implementing the required behavior is generated. Each generated module has a standard structure: an input collector gathers the parameters coming from previous activities; the activity business logic part comprises a form with fields corresponding to the output of the activity and a Create unit that stores the information produced by the activity persistently, for subsequent use. For gateways, the transformation rule behaves according to the BPMN semantics and to the kind of executor assigned to the gateway (human or automatic): if the gateway is tagged as human-driven, a hypertext is generated for allowing the user to choose how to proceed; if the gateway is tagged as automatic, the choice condition is embedded in the application logic. The transformation of BPMN gateways is conducted as follows:

- *AND-splits* allow a single thread to split into two or more parallel threads, which proceed autonomously. The WebML model for AND-split automatic execution generates a set of separate threads that launch the respective subsequent activity modules in parallel, while manual execution allows the user to select and activate all the possible branches.
- *XOR-splits* represent a decision point among several mutually exclusive branches. Automatic XOR-splits comprise a condition that is automatically evaluated for activating one branch, while manual XOR-splits allow the user to choose one and only one branch.
- *OR-splits* represent a decision for executing one or more branches. Automatic OR-splits comprise a condition that is automatically evaluated for activating one or more branches, while the manual version allows the user to choose the branches to activate.
- *AND-joins* specify that an activity can start if and only if all the incoming branches are completed. This behavior is usually implemented as automatic.
- *XOR-joins* specify that the execution of a subsequent activity can start as soon as one activity among the incoming branches has been terminated. This behavior is usually implemented as automatic.
- *OR-joins* specify that the execution of the subsequent activity can start as soon as all the started incoming branches have been terminated. This behavior is usually implemented as automatic, possibly through custom conditions on the outcome of the incoming branches.

Figure 7 shows two simplified examples of generated modules: the *XOR (ProductType)* module (Figure 7.a) implements the automatic evaluation of the XOR gateway in the BPMN model of Figure 1: given the ProductID, it extracts its type and checks whether it is a car or a house. The next activity to be performed is set accordingly, and this information is passed to the Next unit in the orchestration site view. The *Customer Credit Score* module in Figure 7.b shows the generated hypertext page that allows the user to enter and store the credit score value for the customer, which is the output parameter of the *Customer Credit Score* activity of Figure 1.

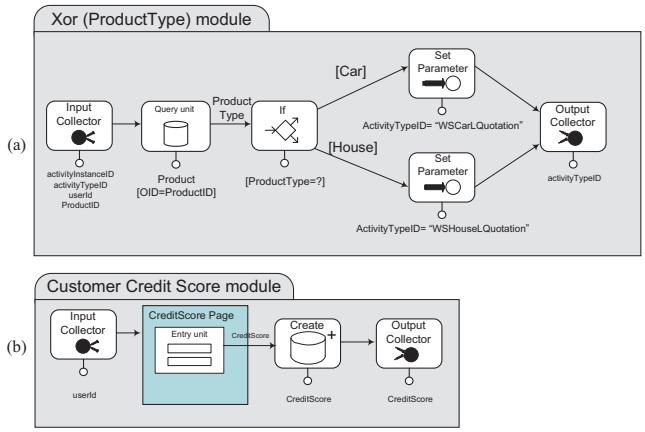


Fig. 7. WebML Modules for XOR gateway and Customer Credit Score.

6 Implementation of WebRatio BPM

The illustrated method has been implemented as a new major release of WebRatio, called WebRatio BPM. To achieve this result, all three major components of the tool suite have been extended: the model editing GUI, the code generator, and the runtime libraries. The *model editing GUI* has been extended by: 1) creating an Eclipse-based workflow editor supporting the definition of the refined BPMN Process Model; and 2) adding the Next unit as a new component available in the WebML Application Model editor. The *code generator* has been extended in two directions: 1) the BPMN to WebML transformation has been integrated within the toolsuite, thus allowing automatic generation of the WebML Application Models and of the Process Metadata. 2) the code generation from WebML has been augmented to produce the instances of the Process Metadata and to integrate the novel components (e.g., the Next unit) into the existing J2EE code generation rules.

Moreover, a one-click publishing function has been added to the BPMN editor, thus allowing the immediate generation of a rapid prototype of the BPMN process. The prototype is a J2EE dynamic, multi-actor application with a default look & feel, produced from the WebML Application Models automatically derived from the BPMN diagrams, according to the previously described techniques. The process prototype comprises a few exemplary users for each BPMN actor, and allows the analyst to impersonate each role in the process, start a process and make it progress by enacting activities and both manual and automatic gateways. Figure 8 shows a snapshot of the user interface of the WebRatio BPMN editor.

The WebRatio BPM tool is being tested in a real industrial scenario of a major European bank, that needs to reshape its entire software architecture according to a BPM paradigm with a viable and sustainable design approach.

The first set of developed applications addressed the leasing department. The running case presented in this paper is inspired by the leasing application that is under development. The real application involves more than 80 business processes, which orchestrate more that 500 different activities.

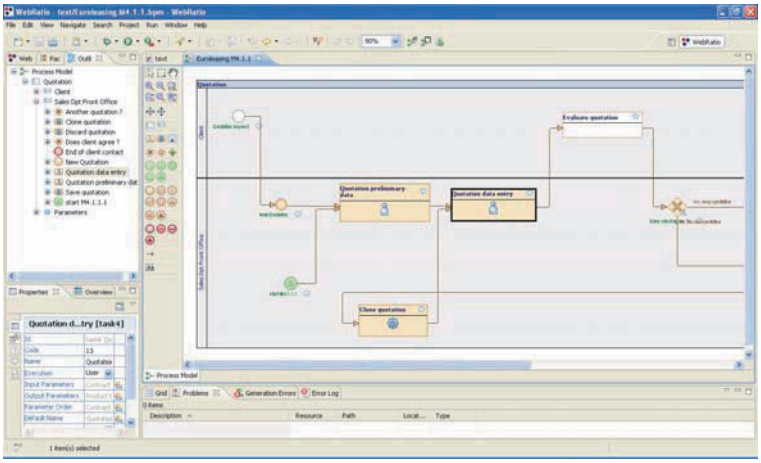


Fig. 8. WebRatio BPM user interface.

7 Related Work

A plethora of tools exist for business process modeling and execution, produced by major software vendors, open source projects, or small companies. In our review of existing tools, we identified more than fifty relevant tools in the field. A report from Gartner [4] describes the magic quadrant of the field and selects the most promising solutions. Among them, we can mention Lombardi Teamworks, Intalio, webMethods BPMS, Tibco iProcess, Savvion BusinessManager, Adobe Livecycle ES, Oracle BPM Suite, IBM WebSphere Dynamic Process Edition. Most of them rely on Eclipse as IDE environment and include a visual designer of business models and a generator of configurations for associated workflow engines. The main innovations of our approach with respect to the competitors are: (1) quick generation of a running prototype; (2) possibility of refinement of the prototype at the modeling level; (3) support of the final application generation through MDD.

In the scientific community, some other works have addressed the challenge of binding the business processing modeling techniques with MDD approaches addressing Web applications development. Koch et al. [5] approach the integration of process and navigation modeling in the context of UWE and OO-H. Also Araneus [7] has been extended with a workflow conceptual model, allowing

the interaction between the hypertext and an underlying workflow management system. In OOHDM [9], the content and navigation models are extended with activity entities and activity nodes respectively, represented by UML primitives. Torres and Pelechano [10] combine BPM and OOWS [3] to model process-centric applications; model-to-model transformations are used to generate the Navigational Model from the BPM definition and model-to-text transformations can produce an executable process definition in WS-BPEL.

8 Conclusion

This paper presented a methodology and a tool called WebRatio BPM for supporting top-down, model-driven design of business-process based Web applications. The tool is now available for testing purposes and will be commercially distributed starting from October 2009. Ongoing and future works include evaluation of productivity of the developers and of quality of the implemented applications, and coverage of further aspects of BPMN semantics (i.e., customized events and exceptions). The tool, albeit still in a pre-beta status, is being used in a large banking application and therefore we expect to collect useful user feedbacks, new requirements, and (quantitative) quality and productivity metrics for the proposed approach in the immediate future.

References

1. Marco Brambilla, Stefano Ceri, Piero Fraternali, and Ioana Manolescu. Process Modeling in Web Applications. *ACM TOSEM*, 15(4):360–409, 2006.
2. Stefano Ceri, Piero Fraternali, Aldo Bongio, Marco Brambilla, Sara Comai, and Maristella Matera. *Designing Data-Intensive Web Applications*. Morgan Kaufmann Publishers Inc., 2002.
3. Joan Fons, Vicente Pelechano, Manoli Albert, and Oscar Pastor. Development of web applications from web enhanced conceptual schemas. In *ER Workshop on Conceptual Modeling and the Web*, volume 2813 of *LNCS*, USA, 2003. Springer.
4. Gartner. *Magic Quadrant for Business Process Management Suites*. Feb. 2009.
5. Nora Koch, Andreas Kraus, Cristina Cachero, and Santiago Meliá. Integration of business processes in web application models. *J. Web Eng.*, 3(1):22–49, 2004.
6. Iana Manolescu, Marco Brambilla, Stefano Ceri, Sara Comai, and Piero Fraternali. Model-Driven Design and Deployment of Service-Enabled Web Applications. *ACM Trans. Inter. Tech.*, 5(3):439–479, 2005.
7. Paolo Merialdo, Paolo Atzeni, and Giansalvatore Mecca. Design and development of data-intensive web sites: The Araneus approach. *ACM Trans. Internet Techn.*, 3(1):49–92, 2003.
8. OMG, BPMI. BPMN 1.2: Final Specification. Technical report, 2009.
9. Hans Albrecht Schmid and Gustavo Rossi. Modeling and designing processes in e-commerce applications. *IEEE Internet Computing*, 8(1):19–27, 2004.
10. Victoria Torres and Vicente Pelechano. Building business process driven web applications. In *BPM*, volume 4102, pages 322–337. Springer LNCS, 2006.
11. Webratio. http://www.webratio.com.

Adding Social Awareness to Jazz for Reducing Socio-Cultural Distance between Distributed Development Teams

Fabio Calefato, Domenico Gendarmi, Filippo Lanubile
Dipartimento di Informatica, Università degli Studi di Bari, via E. Orabona 4
70125 Bari, Italy
{calefato, gendarmi, lanubile}@di.uniba.it

Abstract. A Collaborative Development Environment (CDE) provides a shared workspace with a standardized toolset that helps distributed development teams cope with geographical distance. However, CDEs lack any support to reduce socio-cultural distance, which poses practical barriers to the development of connections and shared culture in distributed settings. The recent rise of the Social Web created several opportunities to publish personal information and develop connections from a distance. We argue that disseminating additional social awareness information to developers, who have little or no chances to meet, can help to speed up the establishment of organizational values, attitudes, and trust-based inter-personal connections. In this paper, building on existing literature, we first propose our definitions of three distinct types of awareness. Then, by means of scenarios, we show how our extension of the Jazz, a CDE that already provides presence and workspace awareness, adds social awareness information about coworkers, i.e., interests, emotional state, in order to reduce socio-cultural distance, and improve team openness and well being in distributed settings.

Keywords: Group Awareness, Social Awareness, Social Web, Web 2.0, Mashup, Jazz, Eclipse, Collaborative Development Environment, CDE.

1 Introduction

In distributed settings, due to distance, software teams often rely on Collaborative Development Environment (CDEs), such as SourceForge[1], GoogleCode[2], Github[3], and Assembla[4], which are an integrated and flexible set of tools (e.g., code compiler and debugger, version control, issue tracking) that help distributed teams control their software development process. Yet, despite their ability to cope with geographical distance, CDEs provide little support to reduce socio-cultural distance. In fact, differences in culture, which can be intuitively epitomized as a fuzzy set of attitudes,

[1] http://sourceforge.net/
[2] http://code.google.com
[3] http://github.com/
[4] http://www.assembla.com/

beliefs, behavioral norms, basic assumptions and values that are shared by a group of people [21], pose practical barriers to the development of relationships and connections (i.e., common ground, mutual confidence, trust) within distributed teams, with a potential severe impact on project management effectiveness.

The idea of applying social software to help distributed teams deal with socio-cultural distance is rather recent. The rise of the Social Web, also known as Web 2.0 [5], created lots of sources for personal, user-generated content, and opportunities to develop connections from a distance. In [1] we presented our initial prototype, which embeds information collected from social websites (e.g., Twitter[5], Facebook[6], Last.fm[7]) into the IBM Jazz CDE. We used the Jazz CDE, because it already provides both presence and workspace awareness, and leveraged the FriendFeed aggregator service to embed personal information about distributed co-workers, collected from social networks. Here, building on relevant literature, we first propose our own defintions of the existing types of awareness. Then, after showing the features or our Jazz extension, we present some key usage scenarios to illustrate how providing distributed software teams with overall group awareness (i.e., presence, workspace, and social awareness) aggregated in one place can help to speed up the establishment of organizational values, attitudes, and trust-based personal connections between distant team members, with little or no chances to meet.

The remainder of this paper is organized as follows. In Section 2, we present an overview on CDEs and Jazz, in particular. In Section 3, building on relevant previous works, we propose our definitions of different awareness types. In Section 4 we first discuss the rise of Social Web applications in general, and then we present FriendFeed as a particular example of Web Mashup. Our Jazz extension is discussed in detail in Section 5, whereas in Section 6 we illustrate the usage scenarios. Finally, we conclude in Section 7.

2 Collaborative Development Environments

In software development, control is the process of adhering to goals, policies, standards, and quality levels, set either formally (e.g., formal meetings, plans, explicit guidelines) or informally (e.g., team culture, peer pressure). Because in distributed settings it is not possible to control units by walking, organizations had to fall back to using collaborative tools to control the software development process from a distance. Collaborative Development Environments (CDEs), such as SourceForge, Gforge, Google Code, are the most used and full-featured process-aware tools to support distributed teams.

CDEs were envisioned by Booch & Brown, who first acknowledged the need for ‘frictionless surface’ in development environments [4], motivated by the observation that much of the developers’ effort is wasted in switching back and forth between different applications to communicate and work together. According to this vision, collaborative features should be available as components that extend core applications

[5] http://twitter.com/
[6] http://www.facebook.com/
[7] http://www.lastfm.it/

(e.g., the IDE), thus increasing the users' comfort and productivity. Therefore, CDEs support developers by incorporating the standard toolset needed (e.g., compiler, debugger, version control system, bug tracker) within a single project workspace, reducing the effort of running multiple different applications to communicate and work together.

Earliest CDE were developed within open source software (OSS) projects because OSS projects, from the beginning, have been composed of dispersed individuals. Today a number of CDEs are also available as commercial products.

2.1 IBM Jazz CDE

Jazz [8] is one of the most noticeable commercial CDE because it can be customized to support any process. Besides, Jazz is an extensible platform, which leverages the Eclipse's notion of plugins to build CDE products. The present version has a wide-ranging scope, but in the former version of Jazz the goal was to integrate synchronous communication and reciprocal awareness of coding tasks into the Eclipse IDE, following Booch & Brown's vision.

Jazz is an extensible team collaboration platform based on a client-server architecture, which integrates many different technologies in a single environment. The Jazz server hosts a set of services (e.g., generate reports, resolve work items from the Web) and houses data in its repository (e.g., configurations, source code). Remote clients communicate with the Jazz server over the network, using SOAP/XML over HTTP (Fig. 1). The full-featured Jazz client is Rational Team Concert, an extension of the Eclipse IDE, packed with all the plugins necessary to the Jazz development platform. It provides presence awareness, thanks to the integration with Lotus Sametime, and workspace awareness, by generating an RSS feed of all project-related events occurring the workspace.

The essential components of Jazz are the Repository and Team Process, which represent the platform kernel and are developed by the Jazz Project Member (Fig. 2). Other members of the Jazz community develop additional components to add new capabilities to Jazz, such as source control and reporting. While the Team Process component is meant to make Jazz a customizable, process-aware platform, Repository allows to store tool-specific information in a central place where it can be made available to all other components in all client and server configurations. Thus, the Repository plays a key role in Jazz extensibility.

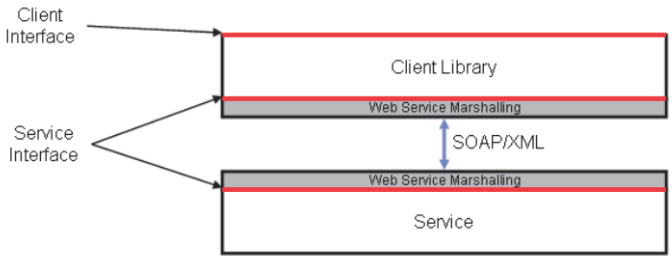


Fig. 1. Jazz Client-Server Communication [11]

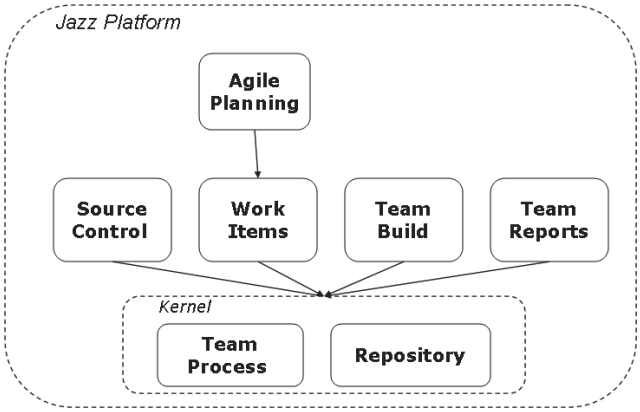


Fig. 2. Components of the Jazz Platform [11]

3 Types of Awareness: Definitions

Dourish and Bellotti were among the first to define the concept of awareness as an understanding of the activities of others, which provides a context for one's activity, so that individual contributions are relevant to the group's activity as a whole [7]. The concept of awareness is strictly connected to the activities of displaying and monitoring of information [16]. When performing a shared, collaborative activity, *displaying* refers to the notification of one's information (e.g., presence and actions, typically) that can be relevant to others involved. *Monitoring* is the complement of information displaying, as it refers to the peripheral observation of others in an unobtrusive way, in order to avoid interruptions.

Being a complex, information-intensive, and highly collaborative activity, software development can greatly benefit from awareness, especially in distributed settings, where teammates have to collaborate from a distance. Ko et al. [13] conducted a study to understand the information needs in software developments teams, showing that the most frequently sought information included awareness about tasks, artifacts, and co-workers. We have identified three major types of awareness: presence, workspace, and social.

Presence awareness is the awareness of what distant colleagues are doing, their availability for interaction, and how they prefer to be reached, helping coworkers to minimize interruptions and disturbances when engaging in collaborative processes [9]. Presence awareness has almost become synonymous with IM and VoIP because such tools represent the preferred, lightweight means to broadcast information or questions, as well as ascertain and negotiate availability to accommodate opportunistic interaction between co-workers.

Workspace awareness means knowing project teams and their internal structure, as well as team members and artifacts. Tools such as Palantìr [18], Hipikat [6], and Mylyn [12] provide developers with workspace awareness information that helps

developers to identify other teammates, artifacts, and tasks that are related to the artifact/task at hand. Workspace awareness is particularly relevant to project managers and team leads as it helps to track the state of a project.

Social awareness is the awareness about interests, opinions, and emotional state of members of a group, which can be extremely beneficial to increase the sense of "teamness" in distributed software settings [16]. Social awareness has been acknowledged only recently, and unlike presence and workspace awareness, it cannot be directly considered contextual to a software development project. Nonetheless, since it helps to develop an organizational culture and to consolidate connections and trust-based relationships between distant collaborators, social awareness contributes to project success by improving team's well being and social health [18]. There are very few software development-oriented tools that support social awareness. One of the most noticeable is Github[8], a software repository that combines standard features of social networking sites (e.g., following or messaging developers, watch projects' activity timeline through feeds) with Git, a distributed source-control system. Codebook [3], instead, is a Microsoft prototype that aims at developing a social networking services over code, in which people can also be friends with the artifacts they share.

Because presence, workspace, and social awareness provide an answer to specific requests of information, if aggregated in one place they can help teams maintain an overall *group awareness*. A recent research study by Omoronya has shown that tool support for distributed software development teams are still inadequate in enhancing distributed awareness because most tools are designed to support a specific kind of awareness in isolation [16]. To date no tool has provided distributed development teams with support to group awareness. IBM Jazz, discussed in the next section, is one of the most recent and full-featured Collaborative Development Environments, which provides presence and workspace awareness in one place, but lacks any support to social awareness.

4 Social Web

Recently, the rise of the Social Web [5], created new incentives and motivations for publishing personal information on the Web. Nowadays, plenty of user-generated content is public available. Applications like Wikipedia[9], Flickr[10], Delicious[11], LibraryThing[12], for example, provide each day new wiki articles, photos, bookmarks and book reviews, as well as new metadata, which are directly added by users.

However, these forms of collaborative contributions are restricted to one single application and current Social Web applications are isolated from one another, like 'walled gardens'. The main reason for this lack of interoperation is that for the most

[8] http://github.com/
[9] http://www.wikipedia.org/
[10] http://www.flickr.com/
[11] http://delicious.com/
[12] http://www.librarything.com/

part in the Social Web, applications' owners are quite reluctant to provide programmatic access to user generated content, which is hosted within their web sites.

In such a context, Web mashups have emerged providing a dynamic approach to compose content and functionalities originating from disparate web sources [23]. Among the different classes of mashup available on the Web, we focused on News Mashups [15] also known as Syndication Feed Mashups since they use syndication technologies like RSS and Atom to aggregate news related to various topics and create personalized feed views. In particular, we envisioned the opportunity to aggregate feeds about personal information originating from social networking sites in order to foster group awareness within distributed teams.

4.1 FriendFeed, a Social Information Aggregator

FriendFeed[13] is a real-time feed aggregator that consolidates the updates from a number of social networking websites (e.g., Delicious, YouTube[14], Last.fm, LinkedIn[15], Facebook), as well as any other custom website providing an RSS/Atom feed. Thus, FriendFeed users can use this stream of information to create customized feeds to share (and comment) with friends. The main reason of FriendFeed success is that it provides the facility to track users' activities (such as posting on blogs, Twitter and Flickr, or listening to music on Pandora[16]) across a broad range of different social networks, whereas other services exclusively facilitate tracking of their own members' activities on their particular social service.

What looked attractive to us about exploiting the FriendFeed service into Jazz is that: (1) a free API is available for leveraging the service output into third-party applications; (2) private groups can be defined so that updates from members are bound within group and not visible to users on the outside; (3) users can decide what is relevant or appropriate to stream and share, reducing information privacy and overloading concerns.

Members of the same Jazz project-area can create a FriendFeed private groups where they can choose which feeds they want to share and who can see the shared feeds. They can also start a conversation around shared items, or just show that they like a feed someone else has shared. By aggregating in the same place different feeds from social networking services used on a daily basis by the project members, we can thus foster the discovering and discussion of personal information regarding people within the project-area, speeding up the development of connections and shared context between team members.

[13] http://friendfeed.com/
[14] http://www.youtube.com/
[15] http://www.linkedin.com/
[16] http://www.pandora.com/

5 The Jazz Client Extension

The Jazz client extension was coded using a Java wrapper of the FriendFeed API. Unfortunately, the API itself does not allow doing as much as desired. For instance, groups cannot be created or managed using the API, which basically allows getting group description and members, reading the feed, and posting messages. In the remainder of this section, we briefly illustrate how the extension works in a simple scenario.

Using the FriendFeed web interface, the project-lead of a Jazz project-area creates a private group for the project. In order to avoid the friction of switching to the web browser and then back to the Jazz client, a browser window can be opened in one of the workspace views to complete group creation (see Fig. 3a).

Other developers needs to be invited to join the group, using either their FriendFeed usernames, in case they are already registered, or just emails (see Fig. 3b). Because the group is private, developers will have to enter their FriendFeed credentials to subscribe the aggregated feed and post contribution. Using the Eclipse Modeling Framework, we extended the project-area model in order to store FriendFeed credentials about both users and groups into the Jazz server repository. This way, even when changing the machine where the Jazz client is run (e.g., in case of switching from the office desktop to the laptop for a business travel), the group feed is available as long as the extension is installed on the client.

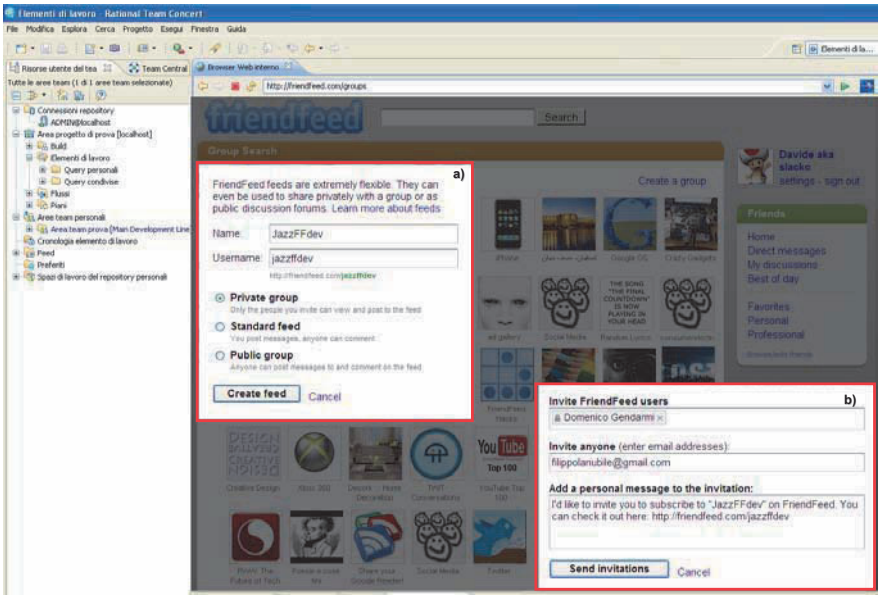


Fig. 3. FriendFeed group creation *a)* Friend Invitation to the group *b)*

Developers can be invited to the group as regular members or admins: Only in the latter case they are allowed to add a service to the aggregated feed. As a group admin, each developer is allowed to specify what personal information collected from

external social web sites can be streamed to the group, without violating his/her privacy. For instance one developer can choose to stream what he likes on Last.fm and Pandora, whereas another one can share her bookmarks saved in Delicious and the book reviews she posted on LibraryThing.

Once the group is created, the project-lead can register it with the project area in Jazz by entering the URL of the aggregated feed. Upon registering the group, the associated feed appears in the Team Artifact view of Jazz, under the Feed folder (see Fig. 4a), along with the default feeds that provide workspace awareness by informing developers about development-related events (e.g., commits, build failures) or any other change occurring in the workspace (e.g., changes to the milestones release date). Hence, the aggregated feed of personal information is visualized using the same internal feed reader provided by Jazz (see Fig. 4b). Because source services can be added to the aggregated stream by each developer who subscribed the FriendFeed group as admin, the larger the development team, the higher the risk of information overload is. Hence, we extended the feed reader to give each developer the opportunity to filter the updates from undesired sources. For instance, one can choose to filter out the updates from Facebook, YouTube and Flickr, while displaying all the others available in the aggregated feed (see Fig. 4c).

Finally, an extra view has been made available in the workspace to let developers post message to the FriendFeed group directly from the Jazz client (see Fig. 5a). All subscribers to the group are then able to view new messages as feeds, through both the FriendFeed web-based interface and the feed reader included within the Jazz client (Fig. 5b).

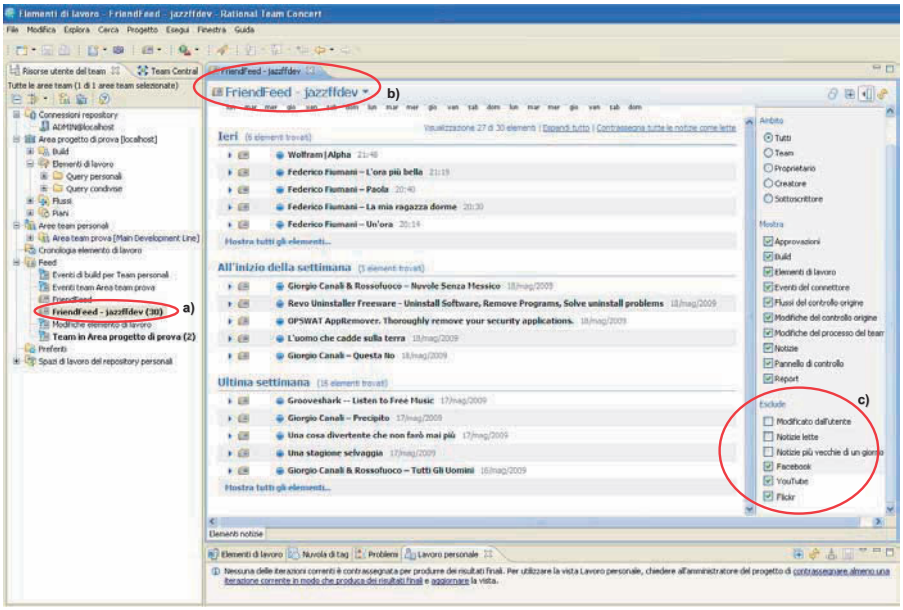


Fig. 4. Name of the feed for the created group a) Feed view on Jazz b) Filtering options c)

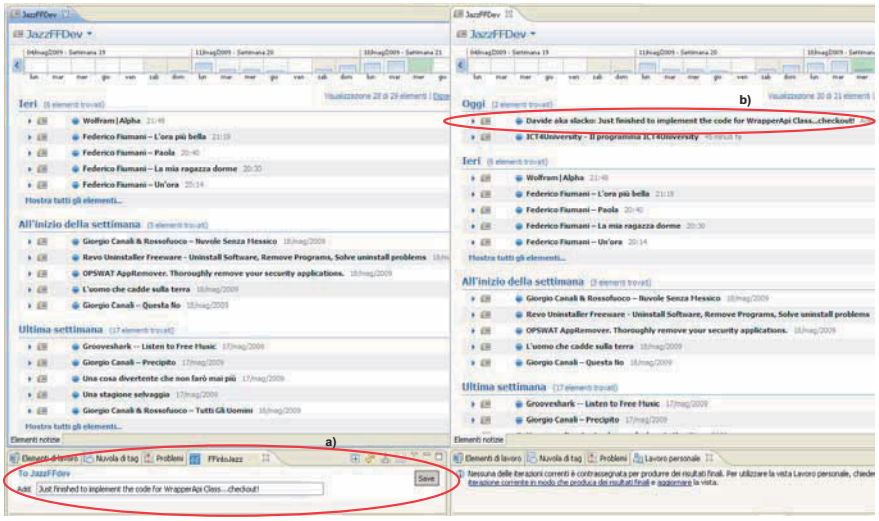


Fig. 5. Posting a message to the group *a)* View of the updated feed *b)*

6 Usage Scenarios

Here we describe some key scenarios to clarify the potential benefits to teams adopting our Jazz client extension.

Facilitate interpersonal connections. Ian has just joined the Irish team of the ALPHA project, which is distributed over multiple sites. Ian's first task is to develop a plugin for Eclipse. Looking at task assignments in the Jazz workspace, he is not able to determine what teammates, if any, have previous experience with the development of plugins.

Adding our FriendFeed extension to Jazz can increase the transparency of group structure and competences, and facilitate the establishment of interpersonal connections. Looking at the project-related FriendFeed stream, Ian decides to filter out any content other than the bookmarks saved in Delicious and the books reviewed on LibraryThing. He finds that Brian has shared several references to articles and has reviewed a couple of books about developing plugins in Eclipse. Thus, Ian, on the one hand, can go through the references he found; on the other hand, he can get in touch with Brian, using one of the communication media available to project members, or he can take advantage of informal, water-cooler conversation, in case they are collocated.

Improve team openness. The BETA project is distributed over two sites, in Ireland and in India. In such settings, one of the solutions that has proved effective to reduce cultural distance and improve team openness is "team buddies", which contemplates that each developer from the Indian site is "buddyed up" with one from the Irish site, providing one-on-one coaching [10]. Thus, Rajiv is buddyed up with Ian. By getting to know each other better, both Rajiv and Ian will broaden their mind, making themselves more open to cultural difference, as well as more indulgent to

language issues and prone to think that the others are not going to act, feel, or think the same way.

The use of our FriendFeed extension can help to increase the effectiveness of team buddies. First, our extension gives both Rajiv and Ian insights about interests, opinions, etc. of each other: for instance, books that one has been reading to get started with a new technology they are going to use in the project, or the pictures from a conference that the other has attended. Besides, our extension can be helpful to learn about the daily rhythm and habits of work. For example, Rajiv may learn that when Ian is contributing something to the FriendFeed stream, is a good time to contact him for a help request with lower chances to interrupt him. Finally, our extension can also help to decrease the number of cultural awareness workshops, which are often used as an effective, but expensive, means to reduce cultural distance in distributed projects [2].

Build a socially open workspace. The GAMMA project is a distributed software development project that spans over multiple sites, including one in India and another one in the US. Indian and North American are quite different cultures. Unlike the latter, Indian is a collectivist, strictly hierarchical culture in which for younger developers it is considered unfair to say no to or disagree with senior developers, team leads, and upper management [17,22]. Our Jazz extension can help to lose strict hierarchical relationships and grant a more equal participation to unhindered discussions.

In collocated projects, equality of participation and unhindered communication is encouraged by adopting a physically open environment, with no cubicles or separated offices for managers and team leads [14]. Our Jazz extension can help to break “sir” relationships, by fostering the development of connections established on a more personal basis, and consequently build a socially open workplace where, despite seniority, it is easier for younger developers to deal with senior team leads and participate in discussion with lower peer pressure.

7 Conclusions & Future Work

Jazz is a Collaborative Development Environment that provides developers with both presence and workspace awareness. In this paper we have presented an extension of the Jazz platform, which leverages the feed aggregation service of FriendFeed to embed social information collected from social networks into the Jazz client and provide developers with group awareness as well. We argue that disseminating additional social awareness information to developers working in dispersed teams can help to speed up the establishment of organizational values, attitudes, and trust-based inter-personal connections, thus facilitating the overall distributed software development process.

While current work is focusing on the enrichment of the Jazz client, as future work we plan to extend Jazz also on the server side by augmenting the Jazz user profile with new professional information. Through the mashup of information collected from both business- and development-oriented social websites (e.g., LinkedIn and

Ohloh), we intend to give to Jazz users the opportunity to customize their profile by including information about professional connections and expertise.

Extending Jazz server-side model allows leveraging the information collected in the whole profile rather than within a specific project area, thus enabling information sharing within different teams inside Jazz. Moreover, we plan to develop the server-side mashup as a web service that exposes personal information as RDF Linked Data in order to enable Jazz users exporting professional profiles and reusing them in third-party relevant applications.

Acknowledgments

Our thanks to Davide Fucci for implementing the initial prototype of our Jazz client extension.

References

1. Abbattista, F., Calefato, F., Gendarmi D., and Lanubile, F. Incorporating Social Software into Agile Distributed Development Environments. Proc. 1st ASE Workshop on Social Sofware Engineering and Applications (SOSEA 2008), L'Aquila, Italy, 15 September 2008.
2. Aston, J., Laroche, L., Meszaros, G. Cowboys and Indians: Impacts of Cultural Diversity on Agile Teams. Agile Conference (AGILE '08), Toronto, 4-8 Aug. 2008, pp. 423-428.
3. Begel, A., DeLine, R. Codebook: Social networking over code. Proc. Int'l Conf. Software Engineering (ICSE '09) Vancouver, Canada, 16-24 May 2009, pp. 263-266.
4. Booch, G. and Brown, A.W., Collaborative Development Environments, Advances in Computers 59, 2003.
5. Chi, E.H., The Social Web: Research and Opportunities, IEEE Computer, vol.41, no.9, pp.88-91, 2008.
6. Cubranic, D., Murphy, G.C., Singer, J., and Booth, K.S. Hipikat: a project memory for software development. IEEE Transactions on Software Engineering, 31(6):446-465, 2005.
7. Dourish, P. and Bellotti, V. 1992. Awareness and coordination in shared workspaces. In Proc. ACM Conf. Computer-Supported Cooperative Work (CSCW '92), Toronto, Ontario, Canada, Nov. 1-4, 1992, DOI= http://doi.acm.org/10.1145/143457.143468.
8. Frost, R., (2007). Jazz and the Eclipse Way of Collaboration. IEEE Software, 24(6), pp. 114-117.
9. Herbsleb, J.D., Atkins, D.L., Boyer, D.G., Handel, M., and Finholt, T.A. Introducing Instant Messaging and Chat into the Workplace. Proc. Int'l Conference on Computer-Human Interaction (CHI '02), Minneapolis, MN, USA, 2002.
10. Holmstrom, H., Conchuir, E.O., Agerfalk, P.J, and Fitzgerald, B. Global Software Development Challenges: A Case Study on Temporal, Geographical and Socio-Cultural Distance. Int'l Conf. Global Software Eng. (ICGSE '06), Florianopolis, Brazil, 3-11 Oct. 2006, pp. 3-11.
11. Jazz Platform Technical Overview, https://jazz.net/learn/PrintableLearnItem.jsp?href=content/docs/platform-overview/index.html
12. Kersten, M. Focusing knowledge work with task context. PhD Thesis, University of British Columbia, 2007.

13. Ko, A.J., DeLine, R., and Venolia, G. Information Needs in Collocated Software Development Teams. Proc. 29th international conference on Software Engineering, Minneapolis, 2007.
14. Law, A., Ho, A., A study case: evolution of co-location and planning strategy, Proc. Agile Development Conference ’04, Salt Lake City, Ut, USA, 22-26 June 2004, pp. 56- 62.
15. Merrill, D. 2006. Mashups: The new breed of Web app. An introduction to mashups. IBM developerWorks. http://www.ibm.com/developerworks/xml/library/x-mashups.html
16. Omoronyia, I. Sharing awareness during distributed collaborative software development. PhD Thesis, University of Strathclyde, November 2008.
17. Rayhan, S.H., Haque, N. Incremental Adoption of Scrum for Successful Delivery of an IT Project in a Remote Setup. Proc. Agile Conference (AGILE ’08), Toronto, 4-8 Aug. 2008, pp. 351-355.
18. Robinson, H., Sharp, H. Organizational culture and XP: three case studies. Proc. Agile Conference (Agile ’05), 24-29 July 2005, pp. 49- 58.
19. Sarma, A., Noroozi, Z., and Hoek, A. Palantír: Raising Awareness among Configuration Management Workspaces. Proc. 25th Int’l Conf. on Software Eng. Portland, 2003.
20. Sillito, J., Murphy G.C., and De Volder, K. Asking and Answering Questions during a Programming Change Task. IEEE Trans. on Software Engineering, 34(4):434-451, 2008.
21. Spencer-Oatey, H. Culturally Speaking: Managing Rapport through Talk across Cultures. New York: Cassel, 2000.
22. Summers, M. Insights into an Agile Adventure with Offshore Partners. Proc. Agile Conference (AGILE ’08), Toronto, 4-8 Aug. 2008, pp. 333-338.
23. Yu, J., Benatallah, B., Casati, F., and Daniel, F. 2008. Understanding Mashup Development. IEEE Internet Computing 12, 5, 44-52.

Weaving Eclipse Applications

Fabio Calefato, Filippo Lanubile, Mario Scalas,

[1] Dipartimento di Informatica, Università di Bari, Italy
{calefato, lanubile, scalas}@uniba.it

Abstract. The Eclipse platform fully supports the ideas behind software components: in addition it also adds dynamic behavior allowing components to be added, replaced or removed at runtime without shutting the application down. While layered software architectures may be implemented by assembling components, the way these components are wired together differs. In this paper we present our solution of Dependecy Injection, which allows to build highly decoupled Eclipse applications in order to implement real separation of concerns by systemically applying Aspect Oriented Programming and the Model-View-Presenter pattern, a variant of the classic Model-View-Controller.

Keywords: Eclipse, Aspect Oriented Programming, Dependency Injection.

1 Introduction

The Dependency Inversion Principle [19] (DIP) states that (both high and low level) software parts should not depend on each other's concrete implementation but, instead, be based on a common set of shared abstractions: one application of the DIP is the Dependency Injection, also called Inversion of Control (IoC) [11]. From an architectural perspective, DI allows to explicit the dependencies between software components and provides a way to break the normal coupling between a system under test and its dependencies during automated testing [25].

This is possible because the software is composed by aggregating simpler, loosely coupled objects that are more easily unit-testable [32]. Additionally, by separating the clients by their dependencies, we also make their code simpler because there is no need for them to search for their collaborators.

The Eclipse Platform [3],[8] is a collection of frameworks for building integrated development environments that has expanded to cover also the development of Rich Client applications [21]. Its building blocks are the Open Services Gateway Initiative (OSGi) [26] specifications, which define a dynamic module system for Java so as to offer a plugin-based component model, and the Standard Widget Toolkit (SWT), a graphic library which provides native application look and feel. However, the Eclipse platform does not have Dependency Injection built-in.

Dependency Injection has proved to be a valuable architectural asset [11],[30]. In particular, according to our own experience [4],[5], during the development of the eConference over ECF [6], a text-based conferencing tool based on Eclipse technologies developed internally, we integrated this pattern as a common asset to be

used for developing every plugin. Rather than creating yet another Dependency Injection framework, we decided to reuse an already existing solution, while only providing the necessary glue-code. In this paper we present how we have used Aspect Oriented Programming to implement Dependency Injection and support the Eclipse dynamic component model.

The remainder of this paper is structured as follows. Section 2 will present an outline of the Dependency Injection and its use cases; section 3 will describe the issues involved in implementing it within Eclipse; section 4 will present the broader context in which we are applying it. Finally, section 5 will present conclusions and future work.

2 Dependency Injection

Dependency Injection comes from the research field of Architecture Description Languages (ADLs), which attempts to assemble or wire components together via configuration mechanisms.

A component is a unit of software that can be instantiated and is insulated from its environment by explicitly indicating (via interfaces) which services are provided and required [23]. The idea of software component comes from the field of electronics engineering: building software should be like wiring electronics components. As long as interfaces are compatible, we should be able to replace old components with new ones, an idea as old as 1968 [22].

The rest of this section provides a brief introduction about the Dependency Injection in general and the Eclipse component model.

2.1 Background

Component Based Software Engineering (CBSE) has two basic concepts, Component Types and Component Instances, which can be respectively mapped to Classes and Instances in Object Oriented Programming (OOP).[1]

More specifically, in Java a class may be seen as a component declaration, thanks to the definition of the implemented interfaces, which can be used as a description of the services it provides. Nonetheless, a class definition fails to declare its dependencies and some kind of convention is required to describe which interfaces are required. This is where containers and configuration mechanisms kick in (see Figure 1).

Clients relinquish to directly instantiate objects and, instead, request them to the container. The latter will use its own configuration, describing the object dependency graph, to retrieve such an instance and return it to the client for usage.

[1] Within the rest of this paper we will use the terms "objects" and "components" as synonyms unless we explicitly provide a different meaning for the different cases.

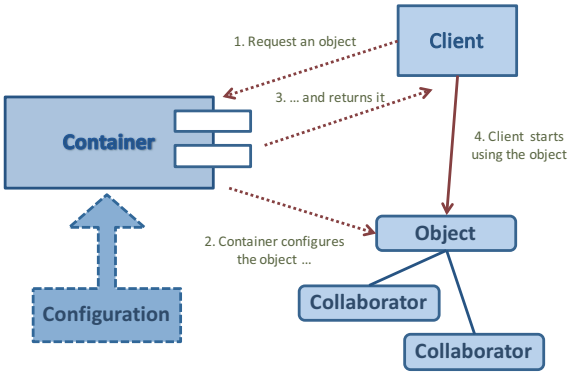


Fig. 1. Dependency Injection

In this scenario, the container itself becomes a key component of a software architecture: it must be boot-strapped before the application starts running and its lifecycle is parallel to the application's. When included in full-fledged web application frameworks, like Spring [30], the container is transparent to the application code: the application must be still aware of the container services but must not care about bootstrapping it since the web framework is handling this task by integrating itself within the application server infrastructure (i.e., a J2EE Application Server). We will call these *managed containers*.

In other uses cases, the container must be explicitly started by some initialization code before it can be used by the client: in this case the client must have direct access to container instance in order to perform requests for objects. We will call these *unmanaged containers*. Integrating a container within the Eclipse Platform is such a case.

Historically, three ways that allow clients to explicit their required dependencies are used:

- Type 1 or Interface-based injection, where clients must implement specific interfaces in order to tell the container which collaborators they need.
- Type 2 or Setter Injection, where clients declare their dependencies by the means of setter methods, which accept specifics collaborator types.
- Type 3 or Constructor Injection, where clients' constructor parameters are their dependencies.

Type 1 is nowadays an inheritance from the past. Setter injection supports the Java Beans convention about class' properties: the setter methods will be used by the container to inject the dependencies. While this is a simple solution, it also opens the class contract by allowing the dependency to be changed at a later time. Constructor injection is stricter about the class contract: dependencies are provided at object instantiation-time and can never be changed as long as an object is alive.

In order to write container configurations, a Domain Specific Language [10] (DSL) is required. A DSL (such as CSS, regular expressions and SQL) is a language

targeted for a particular and limited purpose, not a fully fledged programming language.

A DSL can be *internal*, that is, implemented by using an host language, an approach popularized by the Ruby language, often providing a fluent API.

*External DSL*s, instead, use their own syntax and require a parser to be used. In the case of Dependency Injection, XML has been the most used language, although its syntax badly suits the purpose because of its verbosity-over-expressiveness ratio.

The appearance of built-in annotations within the Java platform from the release 5.0 has enabled an additional way for declaring dependencies, thus pushing several container projects, like Google Guice [15], Pico Container [28] and even Spring, to opt for internal DSLs. The client code will then use library-provided annotations to mark methods or even fields that have to be used to inject required objects whereas the container will use class introspection to scan for annotations and set the required object references.

Internal DSLs have multiple advantages over external DSLs. With an internal DSL: (1) developers have just a single source file to track; (2) the fluent interface is written in the same programming language of the application (e.g., Java), which typically benefit from strong refactoring tools available in many modern IDEs; (3) there is early syntax check. By converse, with internal DSL an abuse of annotations may produce a less readable source code.

Hence, we decided for an internal DSL-based solutions and opted in particular for the Google Guice framework because of the existence of an extension, Peaberry [27], which supports the OSGi component model.

2.2 The OSGi Component Model

OSGi is a set of specifications that define a dynamic module system for Java. In OSGi, components may hide their implementations from other components by the means of *Services*, objects shared across several components (see Figure 2).

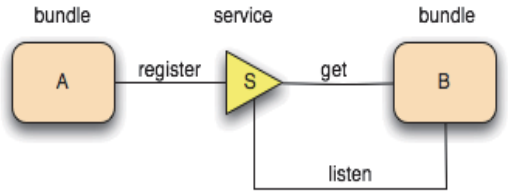


Fig. 2. OSGi publish-subscribe mechanism (from http://www.osgi.org)

Services use a publish-subscribe pattern: components start listening for specific services registered by other bundles. The *Service Registry* framework takes care of tracking down the service instances while specific API is to be used by subscribers, to get actual service instances, and publishers, to make service implementations available to the rest of the system.

Services are deployed within bundles (a synonym of plugin) and the latter can be installed, removed, or updated without shutting down the whole system. Hence, because services can become available or unavailable over time, a service tracking API is needed.
The vision that OSGi designers intended to endorse is that of a *collaborative environment* where applications emerge by dynamically assembling different components with no *a-priori* knowledge of each other (see Figure 3). One of the biggest advantage of OSGi consists in the ability to update, change or introduce new functionalities in a running software system without shutting it down, which is a why OSGi is interesting for application server vendors.

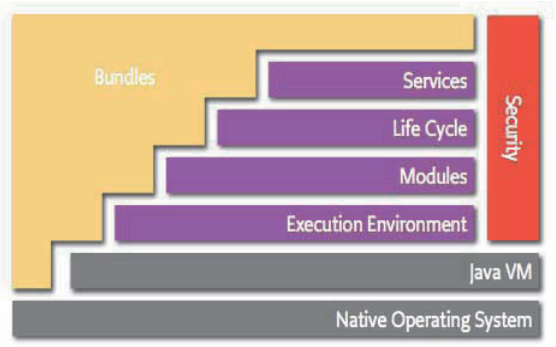


Fig. 3. The OSGi architecture layers (from http://www.osgi.org).

Application bundles use the framework services, such as the publish-subscribe mechanism, the dynamic lifecycle management, and standard Java security. The whole system is based on the concept of modularity: bundles are just plain JAR files with additional OSGi metadata, which define public and private parts. By versioning bundles and, therefore, services, it is possible to have within the same Virtual Machine different versions of the same classes.

Having to deal with dynamic services poses an important question when thinking about a Dependency Injection of OSGi services. In this case, infact, a static injection is not suitable since object structure graph is going to change over time. A simple solution is the introduction of service proxies, as implemented by the Guice/Peaberry. Service proxies act like placeholders for real services: when the actual service component is available, then the proxy passes the call on, otherwise it throws a Service Unavailable exception.

While there are several implementations of the OSGi platform specifications, the current reference implementation is the Eclipse Equinox runtime, the core on top of which the whole Eclipse eco-system is built. In addition to OSGi services, the Eclipse platform historically supports another mechanism for extending software functionalities through platform extensions (plugins). Extensions allow components to be declared and made available to the system, without the need to be loaded until they are actually used (lazy loading).

3 Weaving Dependency Injection

The Eclipse Platform does not support Dependency Injection out-of-the-box: integrating it becomes a framework integration problem, in this particular case, of the Guice and the OSGi frameworks. This section first describes the usage of AOP and then outlines the problems of integration and related solutions.

3.1 AOP and Eclipse

Aspect Oriented Programming [16] (AOP) is a programming paradigm addressing the separation of concerns into reusable modules called *aspects*. AOP complements classical OOP rather than replacing it: while classes modularize primary application concerns (like domain entities, business services, or user interface views), aspects encapsulate secondary, or system, concerns, such as transactions, tracing, security policy enforcement, or performance monitoring.

Merging classes and aspects together is a process called *weaving* and it is usually performed at bytecode level. The weaving process may be executed at compile time (compile time weaving, CTW), by the means of an ad-hoc compiler, or at load time (load time weaving, LTW), by a *weaving agent* that intercepts class loading operations performed by the Java Virtual Machine. At the base of AOP there is the *Join Points Model*, an abstraction for the OOP language constructs, which exposes where aspects can be hooked in the code (e.g., method calls or constructor invocations). An aspect, then, is a construct composed by two parts: a rule-based section, specifying which joint points to capture, and a body part, containing which code to apply when the rules match.

AspectJ [17] is an AOP solution for Java that has tooling support within the Eclipse IDE [2]. Supporting AOP within a dynamic environment as Eclipse poses issues with the aspects weaving: (1) plugins hosting aspects that were woven on classes belonging to other plugins may be become unloaded (i.e., because updated) so the original unwoven classes should be restored before any other re-weaving is possible; (2) new bundles hosting new aspects may be installed within the system and needed to be woven on already loaded classes. All of these cases can only be supported through a careful implementation of LTW, which is the purpose of the Equinox Aspects project [9], which provides new metadata for supporting the two aforementioned scenarios, a set of bundles exporting the weaving service as an OSGi-compliant weaving agent, and a bytecode caching service to improve runtime performance.

The most recent implementation also supports language metadata (through to Java annotations) enabling a declarative way for expressing concerns ([18]).

When implementing Dependency Injection as a system concern, the primary domain concern is the application code requesting the provisioning of collaborators. A possible implementation of the former is detailed in the next section.

3.2 AOP as gluecode

The idea of using Dependency Injection as a system-wide cross-cutting concern and as a reusable abstract base aspect is not new: frameworks like Spring already use it [31]. In particular, programmers mark fields to be injected with ad-hoc annotations like @Autowired so that a special Spring facility, called a weaving agent, will scan components and provide the required dependencies at objects' instantiation time. AOP is then used in order to match the annotations and wire the required code to perform the operation. Nevertheless, implementing the same idea in a dynamic component architecture like OSGi (and Eclipse) requires, instead, special care dealing with the services' dynamic behavior and the different classloading architecture. In fact, an aspect performing Dependency Injection needs to: 1) have access to the BundleContext objects (different for every plugin) in order to access the OSGi services; 2) be provided with a configured container instance (i.e, a Guice container instance); 3) support plugins loading/unloading and, consequenty, aspects corresponding weaving/unweaving (for example, by using Equinox Aspects). In this context, such an aspect will contain all the code necessary to wire objects together with their container, with concrete aspects only differing for the scope of its application (i.e., the packages to weave).

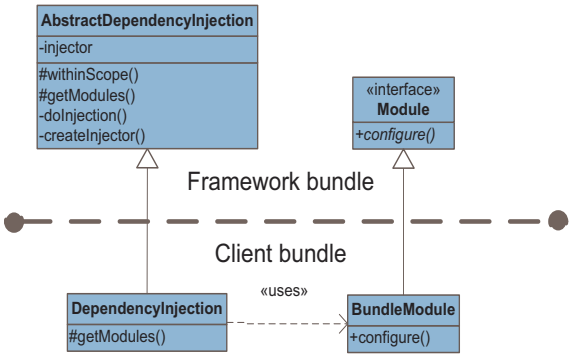


Fig. 4. Modularization of Dependency Injection.

The ability to reuse a common implementation for different contexts is really useful when we have to deal with plugins. Because of the Eclipse platform specifics, in fact, we need to have different concrete Dependency Injection aspects, one for each plugin.

In this model a plugin may publish one or more service objects implementing a contract, that is, a standard Java interface. These services are tracked by the OSGi Service Registry and made available to the rest of the system. Client plugins may request implementations of such contracts and use them as seamless Java Objects with no overhead (apart from the retrieval operations). By intercepting framework events, clients may track their needed dependencies but, in this model, application code intermixes business code with system code. Often, it may be simpler to wrap the objects behind a Proxy and have the latter deal with the OSGi behavior, throwing

exceptions if clients try to use unavailable objects. This is also the solution adopted by Peaberry.

Additionally, to track service objects, the OSGi API is accessible only through the *BundleContext* object which is passed to the plugin Activator's *start()/stop()* methods: this is the standard mechanism provided by the framework to enable client bundles to be notified about events. The bundle context is obviously different for each plugin, so we have to implement a different Dependency Injection aspect for each plugin in order to capture the right bundle context.

In Eclipse-based application, developers are required to provide implementations of standard framework interfaces or classes in order to take advantage of the Eclipse facilities. Frameworks are designed for adaptation and extension, not for integration [20] and Eclipse is no exception since there is little room for configuring the objects that are being created by the platform.

One solution would be to employ the Singleton pattern for locating the container instance and have the newly instantiated object to inject itself, as shown in Listing 1.

```
public class MyCommandHandler extends AbstractHandler {

    @Inject private SomeService someService;

    public MyActionCommand()  {

        // Use Singleton to retrieve the container

        // and call its services ...

        Container.getInstance().configure( this );

    }

    public Object execute ( ExecuteEvent event ) {

          someService.doSomething();

          return null;

    }

}
```

Listing 1. Usage of the Singleton pattern to perfom injection of platform created objects

At runtime, when the default constructor is invoked by the Eclipse framework, the container is also invoked and the dependency injected. Employing Singletons to gain access to the container instance is simple to implement but also defeats the decoupling we are searching in our software system because we are tightly wiring the specific container instance with the client code. Though there is no real other way out with standard OOP but it is still possible to achieve the same effect without any

"hardwiring" of the dependency between the client code (our command handler) and the specific container instance.

The basic idea behind this is to employ the (concrete) Dependency Injection aspect to effectively act as glue-code between the application code instantiated by Eclipse and the container while keeping both separated.

The first action of the Dependency Injection aspect is to intercept the call of the *start()* method to capture the *BundleContext* object and the *stop()* method in order to release service objects when they are no more needed (because OSGi uses reference counting to know when a service object can be released). After this, the Dependency Injection aspect will intercept the creation of instances of classes annotated with the *@Injectable* annotation and configure them. The resulting effect at runtime is the same as in previous solution (i.e., the constructor will get modified at runtime by the weaving agent), but the code concerns remains separated and testable in isolation. Thus, we are able to inject even objects that are written by developers, but instantiated by the Eclipse Framework (e.g., views or command handlers). Doing so, we are using AOP as an integration layer for different frameworks (Eclipse and Guice) in order to bind the application components together [29] (i.e., views with their business service objects), which is also one of the basic steps we need in order to proceed towards further developments, as outlined in the next section.

4 Implementing Model-View-Presenter

Separating presentation from domain means ensuring that no part of the domain code refers to any part in the presentation code [14]. This means that, when writing a WIMP (Windows, Icons, Mouse and Pointer) GUI application, it should also be possible to write a command line interface with the same functionalities without touching the domain code.

Systematically applying the Model-View-Controller [13] (MVC) architectural pattern is a way to enforce separation of concerns since it organizes GUI applications along three primary concerns:

- Model, encapsulating the domain logic behind a set of abstractions (classes and interfaces);
- View, showing the Model's content and notifying the input events to the Controller;
- Controller, which reacts to Model and View events according to some behavior.

Model View Presenter [12] (MVP) is an MVC-variant which further separates Model and View so that they no longer knows about each other; instead, the Controller (called Presenter) is the only listening to both layers' events, driving them according to some application logic, which can be tested.

Separation of presentation and domain logic means not only a way to increase the reuse software parts, but also to design better testable software. In fact, while tools exist to capture mouse clicks for web user interfaces, the resulting macros are tricky to maintain. Separating the domain code improves testability: the greater testability is, the better design becomes.

Additionally, MVP can be applied it in a test-driven process by using the Presenter-first technique [1]. Because this approach avoids dealing with the UI directly, the views must be simple as they only present results or perform data-binding. Testing the presenter means unit-testing it. Dependency Injection finds its application also during testing to assemble the right MVP triplets.

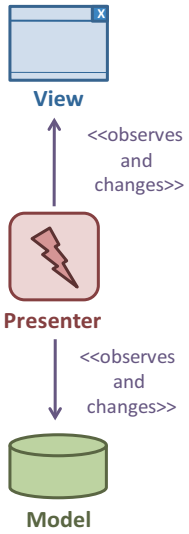


Fig. 5. Modularization of Dependency Injection.

5 Conclusions and future work

At this time we have implemented the Dependency Injection bundle in a project of ours, eConference [4], [5], [6]. eConference is an Eclipse RCP-based distributed meeting system. The primary functionality provided by the tool is a text-based group chat, augmented with agenda, meeting minutes editing, and typing awareness capabilities. Around this basic functionality, other features have been built to help organizers to control the discussion during distributed meetings. The tool has been successfully used to offer the students the opportunity to experience development of software in geographically, distributed multi-cultural teams [7]. The current generation of eConference, eConference-over-ECF, is built on top of the Eclipse Communication Framework and has won the 2006 Eclipse Innovation Award.

As future work, we expect to proceed through the following steps:

1. Extract the framework bundles (like Dependency Injection) from eConference in an order to define a reusable tool for other applications.
2. Perform an architectural check-up of eConference.

3. Design and implement the MVP test and runtime bundles by using eConference-over-ECF as a proof of concept (e.g., the whiteboard and file transfer bundles)
4. Extend Guice and Peaberry in order to support Eclipse concepts and make the process of writing tests for this environment a streamlined process.
5. Get feedback from academic as well as industry projects.

Acknowledgement

This work has been supported by the 2008 IBM Faculty Award.

References

1. Alles, M., Crosby, D., Harleton, B., Pattison, G., Erickson, C., Marsiglia, M., Stienstra, C., “Presenter First: Organizing Complex GUI Applications for Test Driven Development”, Proceeding of the Agile Conference, 23-28 July 2006
2. AspectJ Development Tools, http://www.eclipse.org/ajdt
3. Birsan, D., “On Plug-ins and Extensible Architectures”, Queue, ACM, vol. 3, n. 2, March 2005, pp. 40-46.
4. Calefato, F., Lanubile, F., Scalas, M., "Porting a Distributed Meeting System to the Eclipse Communication Framework", Proceedings of the 2007 OOPSLA workshop on eclipse technology eXchange. p. 46-49, New York, 2007
5. Calefato, F., Lanubile, F., Scalas, M., "Evolving a Text-Based Conferencing System: An Experience Report", Collaborative Computing: Networking, Applications and Worksharing. p. 427-431, Los Alamitos, 2007
6. Calefato, F., Scalas, M., "Adopting the Eclipse Communication Framework: The Case of eConference", Proceedings of the 3rd Italian Workshop on Eclipse Technologies (Eclipse-IT 2008). Bari, Italy, 2008
7. Damian, D., Lanubile, F., Mallardo, T., "An empyrical Study of the Impact of Asynchronous Discussions on Remote Synchronous Requirements Meetings", Lecture Notes in Computer Science, Vol. 3922, 2006
8. Eclipse Platform, http://www.eclipse.org
9. Equinox Aspects, http://www.eclipse.org/equinox/incubator/aspects/
10. Fowler, M., "Domain Specific Language" (Book web draft), http://martinfowler.com/dslwip/
11. Fowler, M., “Inversion of Control Containers and the Dependency Injection pattern”, http://martinfowler.com/articles/injection.html
12. Fowler, M., “Model View Presenter”, http://martinfowler.com/eaaDev/ModelViewPresenter.html
13. Fowler, M., “Patterns of Enterprise Application Architecture”, Addison Wesley Professional, 1st edition, 2002
14. Fowler, M., "Separating User Interface Code", IEEE Software, March/April 2001
15. Google Guice, http://guice.googlecode.com
16. Kiczales,G., Lamping, J. et Al., “Aspect Oriented Programming”, Proceedings of the European Conference on Object-Oriented Programming, vol.1241, pp.220-242, 1997
17. Laddad, R., “AspectJ in action”, Manning, 2004

18. Laddad, R., "AOP and metadata: A perfect match", http://www.ibm.com/developerworks/java/library/j-aopwork3/
19. Martin, R. C., "Dependency Inversion Principle", http://www.objectmentor.com/resources/articles/dip.pdf
20. Mattson, M., Bosch, J., Fayad, M. E., “Framework Integration: Problems, Causes, Solutions”, Communications of the ACM, October 1999, Vol. 42, No. 10.
21. McAffer, J., Lemieux, J-M., “Eclipse Rich Client Platform: Designing, Coding, and Packaging Java™ Applications”, Addison Wesley Professional, 2005.
22. McIlroy, M. D., "Mass Produced Software Components", "Software Engineering, Report on a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968", Scientific Affairs Division, NATO, Brussels, pg. 138-155, 1969 (a transcript can be found at http://www.cs.dartmouth.edu/~doug/components.txt)
23. McVeigh, A., “The Rich Engineering Heritage Behind Dependency Injection”, http://www.javalobby.org/articles/di-heritage/
24. Melnik, G. , Maurer, F., Chiasson, M. Executable Acceptance Tests for Communicating Business - Requirements: Customer Perspective. In Proc. of the Agile Conference (AGILE’06), IEEE Computer Society, pp. 35-46, July 2006.
25. Meszaros, G., “xUnit Test Patterns”, Addison Wesley, 2007
26. OSGi Consortium, Open Service Gateway initiative (OSGi), http://www.osgi.org
27. Peaberry, http://peaberry.googlecode.org
28. PicoContainer, http://www.picocontainer.org
29. Schmidt, D.C., Gokhale, A., Natarajan, B., “Leveraging Application Frameworks”, Queue, July/August 2004.
30.Spring Framework, http://www.springframework.org
31. Walls, C., Breidenbach, R., "Spring in Action", Manning Publications, 2008
32.Weiskotten, J., “Dependency Injection and Testable Objects”, Dr. Dobbs Journal, http://www.ddj.com/development-tools/185300375

An Eclipse Plug-in for Design Pattern Recovery

Andrea De Lucia, Vincenzo Deufemia, Carmine Gravino,
Michele Risi, Genoveffa Tortora

Dipartimento di Matematica e Informatica
Università di Salerno,
84084 Fisciano (SA), Italy
{adelucia,deufemia,gravino,mrisi,tortora}@unisa.it

Abstract. Design patterns are not only beneficial to the forward engineering process but they also help typical reverse engineering activities such as design recovery and program understanding. Indeed, conspicuous insight on the software structure and its internal characteristics are provided by design patterns recovered from source code. In this paper, we present an Eclipse plug-in implementing a reverse engineering tool able to detect pattern instances in Java programs. The plug-in exploits a two phase approach for the recovery of structural design patterns. In the first phase a set of candidate is identified by a visual language parser which considers the design structure only, whereas the second phase validates the candidates through a source code analysis. Moreover, the presented tool integrates reporting facilities and a graphical representation using UML class diagram on the recovered pattern instances.

Keywords: Reverse engineering, Design pattern recovery, Eclipse plug-in, Source code analysis.

1. Introduction

A design pattern can be seen as a set of classes, related through aggregation and delegation, which represents a partial solution to a common non-trivial design problem [10]. Design patterns are widely used to separate an interface from the different possible implementations, to wrap legacy systems, to encapsulate command requests, to use different platforms, and so on [10]. They represent a useful technique in forward engineering since they allow reusing successful practices, to improve communication between designers and to share knowledge between software engineers. However, design patterns also represent useful architectural information that can support a rapid understanding of software design and source code [1, 4, 18, 21]. In reverse engineering of OO software systems they allow to capture relevant information which help the comprehension of the adopted solution. In particular, if a software engineer can understand that a particular design pattern has been used as part of an OO design, he/she can understand the capabilities and limitations of that part of the system.

The extraction of design pattern instances from source code can provide reverse engineers with considerable insight on the software structure and its internal characteristics. Unfortunately, since pattern descriptions are abstract and informal,

and not explicitly documented in software source code, their recovery has to be manually performed in most cases. This is an extremely time consuming task. Therefore, researchers have proposed techniques to automatically recover design patterns in a program, which perform either a static analysis by considering the structural requirements of a pattern, or they combine a static and dynamic analysis, by considering both structural and behavioral requirements. Obviously, for design patterns which comprise significant behavioral aspects the application of static analysis only leads to the recovery of many false positives, i.e., patterns that satisfies the structural requirements of a pattern, while violating its behavioral requirements.

In [6] we have presented an approach to recover structural design patterns from OO source code, which combines a diagram-level analysis, by using a parser for visual languages, with a source code-level analysis. The recovery process is organized in two phases. In the first phase, design pattern instances are identified based on the design structure only by using a recovery technique based on visual language parsing [5]. The design pattern recovery problem is reduced to the problem of recognizing subsentences in a class diagram, where each subsentence corresponds to a design pattern specified by a grammar. In the second phase the identified candidate patterns are validated by performing a source code analysis, which eliminates false positives and consequently increases the precision of the recovery approach. To validate the proposed design pattern recovery approach, we developed a tool, named *DPRE* (*Design Pattern Recovery Environment*), which supports the whole recovery process.

In this paper we present the Eclipse plug-in implementing our recovery approach. The motivations underlying the implementation of the plug-in are that Eclipse is the most used open source development framework and it is strongly based on the concept of extending its functionalities through the implementation of plug-ins. Moreover, the Eclipse platform encourages the exploitation of the functionalities of other plug-ins and modules of the Eclipse framework improving and speeding up the development process.

The paper is organized as follows. Section 2 focuses on the design pattern recovery process implemented by DPRE. In Section 3 details concerning the proposed plug-in for Eclipse to recover design pattern instances are shown. In Section 4 related work on design pattern recovery tools is described. Conclusion and future work are given in Section 5.

2. The Design Pattern Recovery Process

A design pattern is composed of a small number of classes that, through delegation and inheritance, provides a robust and modifiable solution [10]. Design patterns are classified as structural, which concentrate on object composition and their relations in the runtime object structures, creational, which address object instantiation issues, and behavioral, which focus on the internal dynamics and object interaction in the system.

The design pattern recovery process we propose focuses on structural design patterns only, since we only take into account static information. Indeed, structural design patterns are usually described through a class diagram detailed with method

invocations. This has allowed us to define a recovery technique able to work directly on the class diagram representing the input source code.

Fig. 1 shows the proposed design pattern recovery process, where rectangles represent data, while rounded rectangles represent phases of the process. During the Preliminary Analysis information proper to recover design patterns is extracted from input OO source code and stored in a repository. In particular, class diagram information such as the name and type of classes, inheritance and association relationships are stored to be used for the Structural Analysis (in Fig. 1 this information are referred as structural information). Moreover, information on method declarations and invocations useful to perform the Low-level Analysis are also stored.

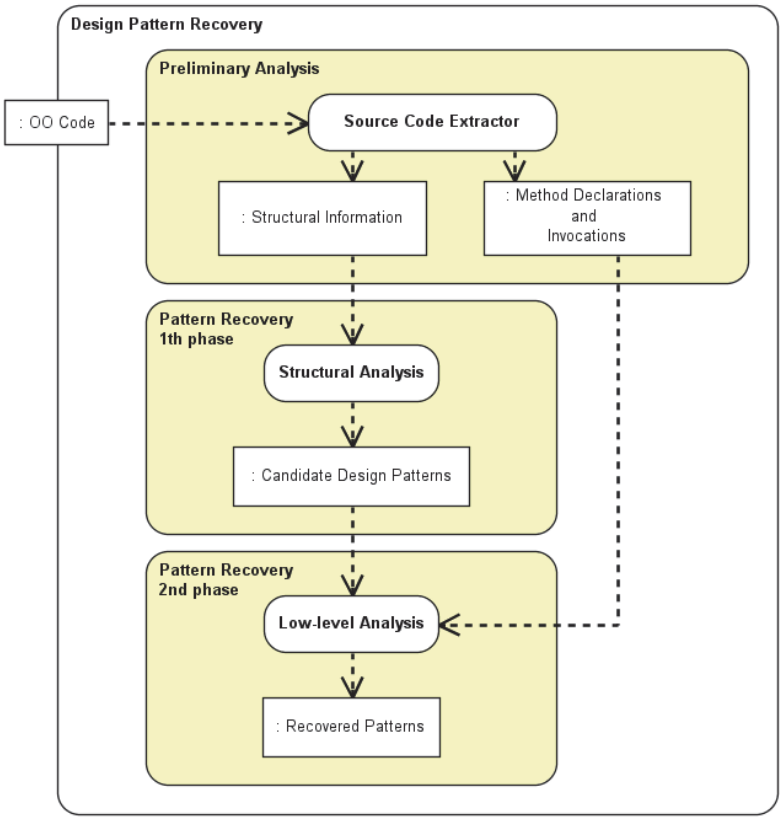


Fig. 1. The design pattern recovery process.

Instances of design patterns are identified by analyzing the class diagram structure during Pattern Recovery. This recovery process is organized in two phases. In the first phase, the candidate design patterns are identified at a coarse-grained level by analyzing the class diagram information obtained during the Preliminary Analysis. In particular, a set of candidate instances of design patterns are identified by analyzing the class diagram constructed from the source code.

The recovery technique applied in the first phase (Structural Analysis) is based on visual language parsing. In particular, the candidate design patterns are identified at a coarse-grained level by analyzing the class diagram information through a visual language parser [6]. In particular, once the class diagram has been abstracted from the source code, the problem of design pattern recovery is reduced to the problem of recognizing subsentences in a class diagram, where each subsentence corresponds to a design pattern instance. Indeed, the idea is to specify the class structure of the design patterns to be recovered in terms of a grammar specification from which it is possible to obtain the corresponding parser automatically [5].

In the second phase (Low-level Analysis), a fine-grained source code analyzer checks if the identified candidate patterns are correct patterns or false positives. This is accomplished by verifying at source code level the declarations and the invocations of the methods of the classes involved in the candidate design patterns. Observe that the design pattern recovery process has been implemented to recover the structural design patterns Adapter, Bridge, Composite, Decorator, Façade and Proxy. The definition of the recognition algorithms for the considered structural design patterns, checks, and more details on the static analysis phase can be found in [6].

3. DPRE Eclipse Plug-in

As introduced by Gamma and Beck in [11], Eclipse can be considered as a technology, a development platform, or a group of tools. Therefore Eclipse is a collection of both plug-ins and access points of plug-ins. To take advantage of the latest development techniques and to enhance DPRE's extensibility [6], we chose the Eclipse framework as our development platform and decided to implement DPRE as an Eclipse plug-in. In the following we describe the architecture of the plug-in (shown in Fig. 2) and how it implements the proposed design pattern recovery process.

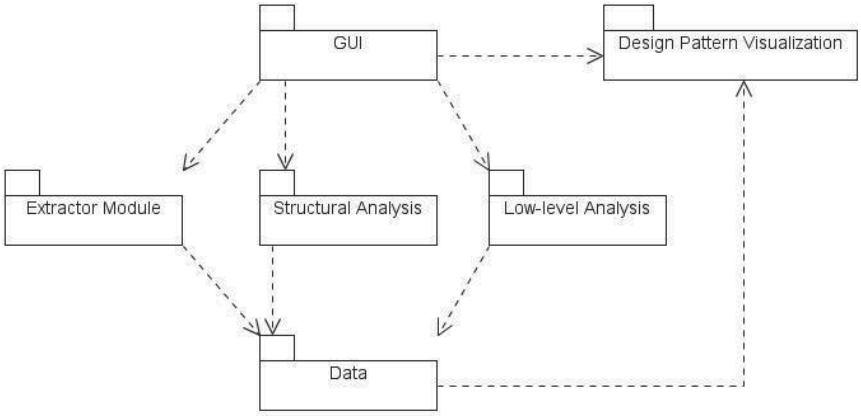


Fig. 2. The architecture of the DPRE Eclipse plug-in.

The plug-in exploits the code analysis tool *Source Navigator* [22] in order to implement the Extractor module in Fig. 2. Source Navigator is able to recover almost

all the necessary information and organize it in a structure suitable for our purposes and supports several programming languages, such as C++, Java, and Python, and provides APIs allowing programmers to construct a specific parser. However, Source Navigator is not able to recover some relevant information for design pattern identification. As an example, in case of Java the tool does not extract the inheritance relationships between classes where the super class is an interface. To overcome these deficiencies of the tool, we have developed an ad-hoc scanning module to recover the missing information from the source code. The information on the source code is structured as a set of tables storing information on classes, relationships between classes, method declarations, and method invocations, and so on. The adoption of this data representation makes the recovery process independent of the programming language adopted for the input OO code. Indeed, the information is organized extracting the syntactic features common to OO programming languages.

The Structural Analysis of the plug-in has been implemented as an LR-like parser, obtained from the design pattern grammar specification [6]. The input to the parser is the set of tables obtained from the Extractor module, while the output is a set of textual descriptions representing the candidate design pattern instances. In particular, the output descriptions include the qualified name of the classes involved in the pattern instances together with the low-level checks to be performed on them.

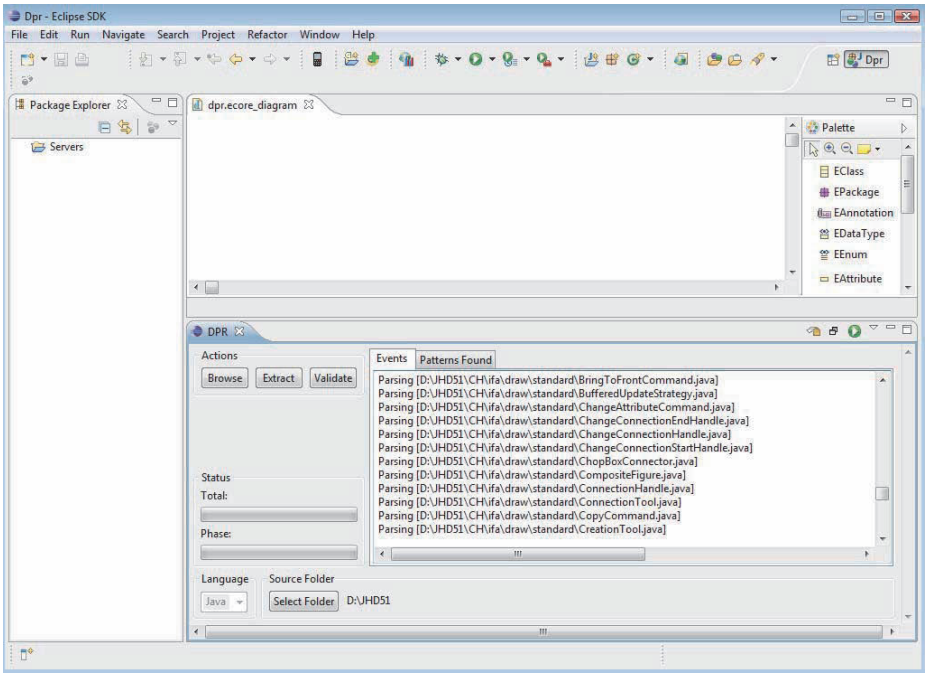


Fig. 3. The graphical interface of the DPRE Eclipse plug-in.

The Low-level Analysis takes in input the outputs produced by the Extractor module and the Structural Analysis and returns the set of textual descriptions representing the recovered design pattern instances. It analyzes the candidate

descriptions in order to invoke the methods accomplishing the low-level checks on the classes involved in the pattern instances. The pattern descriptions that succeed the checks of the methods are given in output.

Fig. 3 also shows the use of the plug-in during the design pattern recovery process of JHotDraw 5.1 (8300 LOC and 155 classes), one of the analyzed case studies. The graphical interface allows users to select (by clicking on the *Select Folder* button) the directory containing the source code. At this point the plug-in provides a window for choosing the path of the folder where the source code files of the system to be analyzed are stored. Note that these files have to be stored on the local file system of the computer where the plug-in is installed. Successively, the recovery process is started by accomplishing the Preliminary Analysis (by clicking on the *Browse* button). Then, the user carries out the Structural Analysis and the Low-level Analysis of the recovery process by clicking on the *Extract* and *Validate* buttons, respectively.

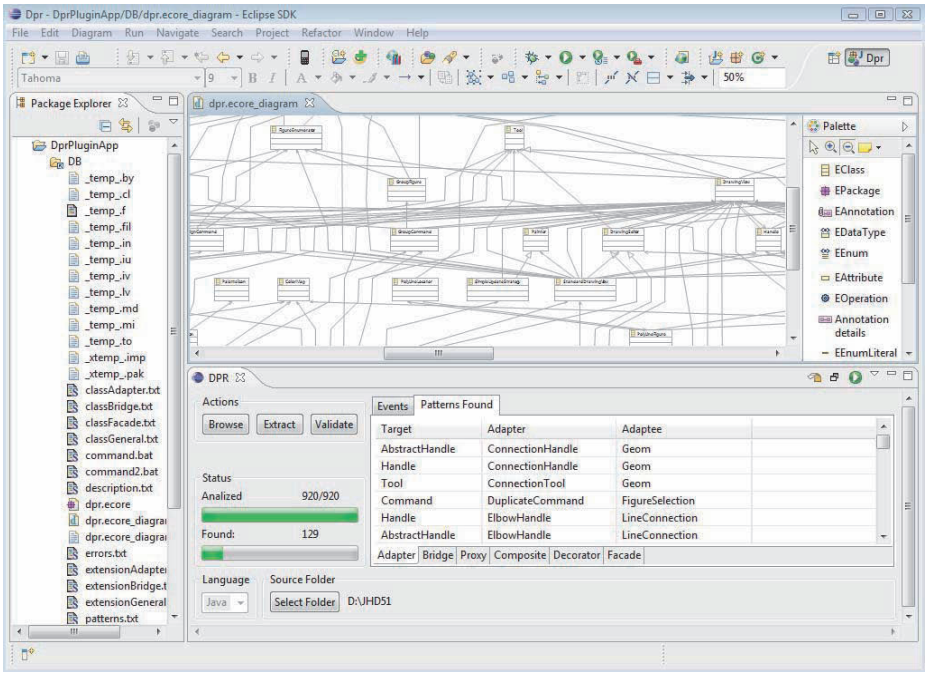


Fig. 4. The recovered design patterns.

Fig. 4 shows a screen-shot of the plug-in after the design pattern recovery process. The *dpr.ecore_diagram* view visualizes the UML class diagram using the information carried out during the Preliminary Analysis phase. These information are properly stored in a data structure and saved into text files. In particular, the Package Explorer is updated and shows the list of these files created and used during the recovery process. The plug-in uses these files to restore the views and the UML class diagram every time it is lunched avoiding to re-execute the recovery process for each analyzed OO source code.

Observe that the user can visualize the recovery results by clicking on the *Patterns*

found button. In particular, the plug-in shows a table containing the statistics on the pattern instance recovered, and for each recovered pattern instance the user can visualize the classes involved in the pattern and the role they play. As an example, for the considered case study (i.e., JHotDraw 5.1), the candidates extracted during the Structural Analysis were 920, whereas the final number of recovered design pattern was 129 (after the Low-level Analysis). In particular, Table 1 depicts the pattern instances identified by the plug-in taking into account the Structural Analysis, i.e., the first phase of the recovery process, and the complete recovery that includes the Low-level Analysis, i.e., the second phase of the recovery process.

Table 1. Recovered instances of design patterns (JHotDraw 5.1).

	1st phase	2nd phase
Adapter	506	35
Bridge	404	86
Composite	5	0
Decorator	0	0
Façade	5	8*
Proxy	0	0

* For Façade pattern the low-level checks allow to detect for each candidate instance many façade actual instances.

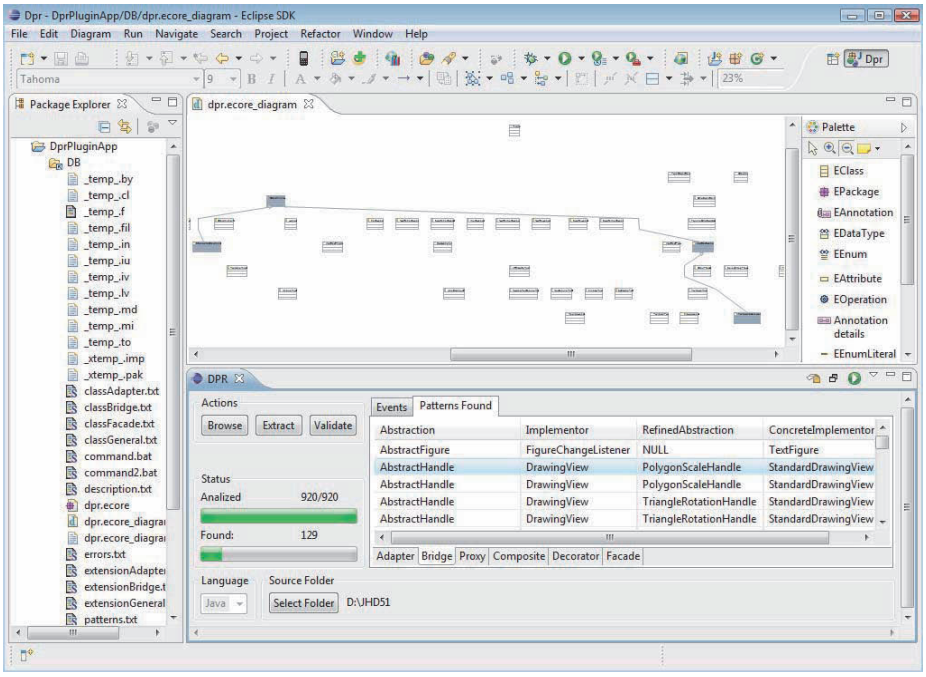


Fig. 5. The selection and the visualization of recovered design patterns.

Selecting a particular instance, the involved classes are highlighted in the UML class diagram shown in the *dpr.ecore_diagram* view. In particular this view is suitably zoomed in order to show all the classes composing the recovered design pattern. As an example, Fig. 5 shows the selection of a pattern in the list of retrieved Bridge patterns and how the classes *AbstractHandle*, *DiagramView*, *PolygonScaleHandle* and *StandardDrawingView* in the UML class diagram view are highlighted. Moreover all the relationships involved in the design pattern are also shown.

Finally, to control the recovery process the plug-in provides a preference tab-sheet, as shown in Fig. 6. In particular, the user can choose the structural design pattern to be considered in the recovery process and the low-level checks to be performed on them during the Low-level Analysis.

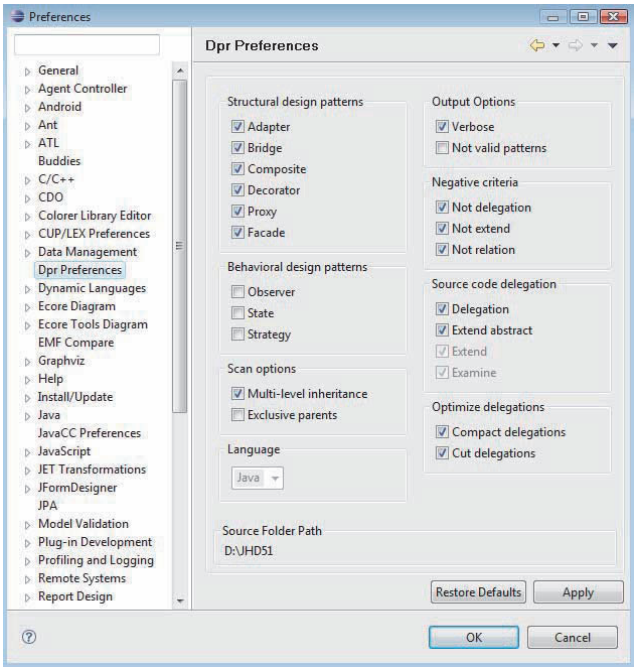


Fig. 6. The preferences tab-sheet of the DPRE Eclipse plug-in.

4. Related Work

Some evaluation and classification of design pattern recovery approaches and tools supporting them have been provided in the literature taking into account the employed pattern identification strategy, the representation used for coding design patterns, the kind of support they provide for recognition (i.e., manual, semi-automatic or automatic pattern recovery), the type of design patterns they are able to recover, the software analyzed to assess the effectiveness of the proposed pattern recovery

strategies (see e.g., [6, 12]). In the following we provide information on some the tools provided to identify design pattern instances.

The Pat system is able to recover instances of structural design patterns using information on the pattern class structure [17]. In particular, design patterns are represented as Prolog rules whereas source code is expressed in terms of Prolog facts. Thus, instances of design patterns are retrieved by applying a Prolog query. Some information such as the difference between concrete and abstract classes is not extracted. The approach is automatic, but false positives have to be removed manually.

CrocoPat is a tool able to automatically recover design pattern instances from object-oriented programs, by exploiting relational expressions to specify properties of a system [2, 3]. In particular, a relation manipulation language is used to manipulate n-ary relations and the effectiveness of the system is based on binary decision diagrams which represent the relations compactly. CrocoPat works in three steps. In the first step the data relevant for the analysis is extracted from source code while in the second step the pattern of interest have to be defined by using a pattern specification language which exploits the relations stored in a relation file. The third step concerns the analysis, where CrocoPat translates the relations, contained in the relation file, into binary decision diagrams [2, 3].

SPOOL is a tool that retrieves design patterns from C++ code based on structural descriptions of design patterns [16, 17]. The relevant C++ source code elements are represented in UML/CDIF format, whereas the patterns are represented as abstract design components and stored in a central repository. The environment of SPOOL has a three tier architecture and supports both forward and reverse engineering of design pattern instances. The tier at the bottom provides physical storage of the information on the reverse engineering model and on the design. The tier at the top is represented by the end-user tools providing functionality like the ones for capturing source code and for visualization, while the object-oriented schema of the reverse engineering model represents the tier in the middle. SPOOL supports the manual, semi-automatic, and automatic recovery and the design patterns retrieved are Template Method, Factory, and Bridge.

The approach proposed in [14, 15] combines static and dynamic analyses to recover design pattern instances. In particular, two Prolog-based languages, namely SanD-Prolog and SanD, are used to specify predicates. The input source code is represented as a set of predicates that encode the corresponding AST (Abstract Syntax Tree). During static analysis queries are carried out on the source code representation based on the static specification of the pattern. The dynamic analysis monitors the execution of the program and classifies the pattern instance candidates obtained by the static analysis according to their conformance to the expected pattern behavior. In particular, the employed predicates specify the relevant states and state transitions of a pattern's behavior to detect in a concrete program run.

The tool PINOT implements a recovery approach based on a static analysis which exploits inter-class relationships [19]. The program behavior is efficiently recognized using a lightweight static program analysis. These analysis are performed on symbol tables and ASTs constructed by a compiler.

The approach proposed in [8, 9] is based on the use of matrices and weights to recover instances of design pattern and is implemented in the DP-Miner toolkit. In

particular, DP-Miner builds a matrix from source code where all classes in the system correspond to the rows and columns, and the relationships between each pair of classes to be the value of the corresponding cell in the matrix. The relationships between classes are encoded in each cell value by using different prime numbers and combination of relationships by product of prime numbers. Since the information on design pattern are encoded as matrix and weights, the discovery of design patterns is reduced to matching such matrices and weights in arithmetic computations.

The approach proposed in [23] to automatically detect modified design patterns is based on the use of graphs. The software and the design patterns to be retrieved are represented as graphs and matrices are used to represent important aspects of their static structure. Then, a graph similarity algorithm is employed to detect instances of candidate design patterns. The idea is to apply the similarity algorithm to clusters of hierarchies in order to restrict the analysis to smaller subsystems rather than to the whole system. The proposed recovery methodology has been implemented as a Java program, which is able to identify instances of Adapter/Command, Composite, Decorator, Factory method, Observer, Prototype, Singleton, State/Strategy, Template method, and Visitor patterns.

The recovery technique and corresponding tool (named DEMIMA) proposed in [13] are based on a multilayered approach able to identify idioms pertaining to the relationships between classes and design motifs characterizing the organization of the classes[1]. The first layer is devoted to the construction of models from source code, by using a language whose metamodel is inspired by UML and includes all the elements of a Java system like class, interface, method, and so on. In the second layer programming idioms are identified, which specify specific characteristics of classes or relationships between them. In the third layer the language used to describe the source code is also used to represent design motifs, and microarchitectures similar to the specified design motifs are recovered from the model representing the source code by using explanation-based constraint programming and constraint relaxation.

5. Conclusion and Future Work

Software system maintenance requires a deep comprehension of the existing system in order to modify and integrate it with new or changing requirements. Design patterns represent useful architectural information that can support a rapid understanding of software design and source code. In reverse engineering of OO software systems they allow to capture relevant information which help the comprehension of the adopted solution [1, 4, 18, 20, 21].

We have presented the Eclipse plug-in implementing the two phase design pattern recovery approach we proposed in [6]. In the first phase the class diagram extracted from the source code is analyzed for identifying design structures that are candidate pattern instances. This is accomplished using a recovery technique based on visual language parsing [5]. In the second phase the code of the classes involved in the identified candidate patterns is examined for verifying their compliance to the code-

[1] It is worth noting that the authors used the term design motif to indicate design pattern and microarchitecture to refer to design pattern instance.

level constraints defined by the corresponding structural patterns. These checks allow us to eliminate many false positives and consequently to increase the precision of the approach.

Future work will be devoted to integrate the Eclipse plug-in with other functionality and modules of the Eclipse framework improving and speeding up the development process. In particular, we will improve the design pattern instances representation in the UML class diagram providing more information about the recovered pattern (such as the involved methods) and reorganizing the layout respect to the role played by the classes involved in the design pattern instance. Moreover, we are extending the Eclipse plug-in with another module for the detection of behavioral design patterns according to the approach proposed in [7].

References

1. G. Antoniol, G. Casazza, M. Di Penta, R. Fiutem, "Object-oriented design pattern recovery", *Journal of Systems and Software*, 59, 2001, pp.181-196.
2. D. Beyer and C. Lewerentz, "CrocoPat: Efficient pattern analysis in object-oriented programs", in *Proceedings of the International Workshop on Program Comprehension* (IWPC'03), Portland, Oregon, USA, 2003, pp. 294–295.
3. D. Beyer, A. Noack, and C. Lewerentz, "Efficient Relational Calculation for Software Analysis", *IEEE Transactions on Software Engineering*, 31(2), 2005, pp. 137–149.
4. K. Brown, "Design reverse-engineering and automated design pattern detection in smalltalk", Master Thesis, North Carolina State University, Raleigh NC, 1996.
5. G. Costagliola, A. De Lucia, V. Deufemia, C. Gravino, M. Risi, "Design Pattern Recovery by Visual Language Parsing", in *Proc. of European Conference on Software Maintenance and Reengineering* (CSMR'05), 2005, pp. 102-111.
6. A. De Lucia, V. Deufemia, C. Gravino, M. Risi, "Design Pattern Recovery through Visual Language Parsing and Source Code Analysis", *Journal of System & Software*, 18(7), 2009, 1177-1193.
7. A. De Lucia, V. Deufemia, C. Gravino, M. Risi, "Behavioral Pattern Identification through Visual Language Parsing and Code Instrumentation", in *Proceedings of European Conference on Software Maintenance and Reengineering* (CSMR'09), 2009, pp. 99-108
8. J. Dong, D. S. Lad and Y. Zhao, "DP-Miner: Design Pattern Discovery Using Matrix", in *Proceedings of IEEE International Conference on Engineering of Computer Based Systems* (ECBS'07), Tucson, Arizona, USA, 2007, pp. 371-380.
9. J. Dong and Y. Zhao, "Experiments on Design Pattern Discovery", in *Proceedings of International Workshop on Predictor Models in Software Engineering* (PROMISE'07), 2007.
10. E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Menlo Park, CA, 1995.
11. E. Gamma, K. Beck, *Contributing to Eclipse: Principles, Patterns, and Plugins*, Addison-Wesley, 2003.
12. G. C. Gannod, B. H. C. Cheng, "A framework for classifying and comparing software reverse engineering and design recovery techniques", in *Proceedings of Working Conference on Reverse Engineering* (WCRE'99), 1999, p. 77–88.
13. Y. Guéhéneuc, G. Antoniol, "DeMIMA: A Multilayered Approach for Design Pattern Identification", *IEEE Transactions on Software Engineering*, 34(5), (2008), pp. 667-684.

14. D. Heuzeroth, S. Mandel, and W. Lowe, “Generating Design Pattern Detectors from Pattern Specifications”, in *Proceedings of the International Conference on Automated Software Engineering* (ASE’03), 2003, pp. 245–248.
15. D. Heuzeroth, T. Holl, W. Lowe, “Combining Static and Dynamic Analyses to Detect Interaction Patterns”, in *Proceedings of the International Conference on Integrated Design and Process Technology* (IDPT’02), 2002.
16. R. K. Keller, R. Schauer, S. Robitaille, P. Pagé, “Pattern-Based Reverse-Engineering of Design Components”, in *Proceedings of International Conference on Software Engineering*, Los Angeles, 1999, pp. 226-235.
17. C. Kramer, L. Prechelt, “Design recovery by automated search for structural design patterns in object oriented software”, in *Proceedings of Working Conference on Reverse Engineering*, IEEE CS Press, 1996, pp. 208-215.
18. J. Niere, W. Shafer, J. P. Wadsack, L. Wendehals, J. Walsh, “Towards Pattern design recovery”, in *Proceedings of International Conference on Software Engineering*, Orlando Florida, USA, 2002, pp. 338-348.
19. R. Olsson, N. Shi, “Reverse Engineering of Design Patterns from Java Source Code”, in Proc. of IEEE/ACM International Conference on Automated Software Engineering (ASE'06), 2006, pp. 123-134.
20. I. Philippow, D. Streitferdt, M. Riebish, S. Naumann, “An Approach for Reverse Engineering of Design Patterns”, *Journal of Software and System Modeling*, 4(1), 2005, pp. 55-79.
21. F. Shull, W.L. Melo, V.R. Basili, “And inductive method for discovering design patterns from object-oriented software systems”, *Technical Report*, University of Maryland, Computer Science Department, College Park MD, 1996.
22. SourceNavigator, http://sourcenav.sourceforge.net/
23. N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, S. T. Halkidis, “Design Pattern Detection using Similarity Scoring”, *Transaction on Software Engineering*, 32(11), 2006, pp. 896-909.

An Eclipse Plug-in to Enhance the Navigation Structure of Web Sites

Damiano Distante[*], Michele Risi[+], Giuseppe Scanniello[†]

[*] Faculty of Economics, Tel.M.A. University, Rome, Italy
[+] Department of Mathematics and Computer Science, University of Salerno, Fisciano, Italy
[†] Department of Mathematics and Computer Science, University of Basilicata, Potenza, Italy
damiano.distante@unitelma.it, mrisi@unisa.it, giuseppe.scanniello@unibas.it;

Abstract. This paper presents a process and a tool developed as an Eclipse plug-in for automatically enhancing the navigation structure of Web sites. The process extends each page of a site with a Semantic Navigation Map, i.e., a set of links connecting the page to other pages of the site showing similar or related content. The process uses Latent Semantic Indexing to compute a dissimilarity measure between the pages of the site, a graph theoretic clustering algorithm to identify groups of pages with similar or semantically related content, and AJAX code to extend each page with the corresponding Semantic Navigation Map. Semantic navigation maps for a given Web site are recovered once and kept up to date as new pages are added to the site or content of existing pages is updated. Additionally to presenting the process, the underlying techniques and the tool supporting the process, the paper also presents the results obtained from a case study involving a real world Web site.

Keywords: Clone Detection, Latent Semantic Indexing, Software Maintenance, Web Site Evolution, Web Site Navigation Structure, Eclipse plug-in.

1 Introduction

The success of a Web site depends in part on easy and quick access to the information it provides, i.e. on its navigability. Navigability, indeed, is one of the critical factors determining the usability of a Web site, i.e. the capability of the Web site to support the effective, efficient and satisfactory accomplishment of user tasks [29], particularly information gathering tasks.

The importance of navigation in Web sites[1] is also demonstrated by the attention devoted to this aspect by basically all most known Web engineering methods [5][18] [25][26], which devote a specific design activity and a specific design model to define the navigation structure of a Web site. A navigation model is usually based on two modeling concepts: the concept of Node and the concept of Link. Nodes are defined as self-contained uniquely identifiable units of information from/to which the user can navigate in a Web site. Links are used to connect nodes and to enable navigation

[1] Here and elsewhere in the paper, the term "Web site" can be generalized into that of "Web application", as our focus is on their navigation structure.

between them. Links between nodes are arranged to form particular access structures such as guided tours or access by index.

However, due to time-to-market constraints, lack of proper skills, and/or poor acceptability support [3][16], such Web engineering approaches, very often, are not used in the industrial practice and little effort is devoted, in particular, to designing a Web site prior to implementing it. Other times Web engineering approaches are only used for developing and deploying the first version of the site and then neglected during the rest of its life time, when new contents and/or functionalities are introduced and original ones removed.

This paper presents an Eclipse plug-in to improve the navigability of a Web site. This plug-in implements the approach previously presented in [24]. In particular, this approach extends the navigation structure of a Web site with Semantic Navigation Maps, i.e. with sets of links enabling navigation from each page of the site to other pages showing similar or related content. The process uses Latent Semantic Indexing (LSI) [10], a well known information retrieval technique, to compute a dissimilarity measure between the pages of the site, a graph-theoretic clustering algorithm [14] to identify clusters (groups) of pages showing similar or related content, and AJAX code [6] to extend at run-time each page of the site with a semantic navigation map. In order to automate the process of semantic navigation map recovery and injection and to facilitate the adoption of the approach, we have developed a prototype of a supporting tool as an Eclipse plug-in. The process and the tool have been applied with success in a case study, also presented in this paper, that proved their validity and usefulness.

The rest of the paper is organized as follows. Section 2 describes the process to recover semantic relations among the contents of a Web site and to enhance its navigation structure with semantic navigation maps. Section 3 describes the Eclipse plug-in we developed to support the recovery and evolution process. Section 4 reports the results of a case study while Section 5 discusses a number of works related to Web sites evolution. Finally, Section 6 concludes the paper by providing some final remarks and describing avenues for future work.

2 The Process

The process to extend the navigation structure of a Web site with semantic navigation maps is schematically represented in Figure 1. In the following subsections we describe in detail each of the phases and subphases of the defined process.

2.1 Recovering Semantic Navigation Maps

The phase *RecoveringSemanticNavigationMaps* is composed of three subphases: *ComputingDissimilarity*, *GroupingSimilarPages* and *RemovingPages*. ComputingDissimilarity extracts the textual content of each page (i.e., the text that is presented to a user visiting the page) of the analyzed Web site and computes the dissimilarity between any pairs of pages using a measure based on Latent Semantic Indexing (LSI) [10]. A dissimilarity matrix is produced as output of this activity. This

matrix is used by the subphase of GroupingSimilarPages to identify groups of pages with similar or related content. Pages included in single clusters, i.e., clusters containing only one page, are discarded by the RemovingPages subphase and are not considered in the following iterations of the process. The subphases GroupingSimilarPages and RemovingPages are repeated until no single clusters remain.

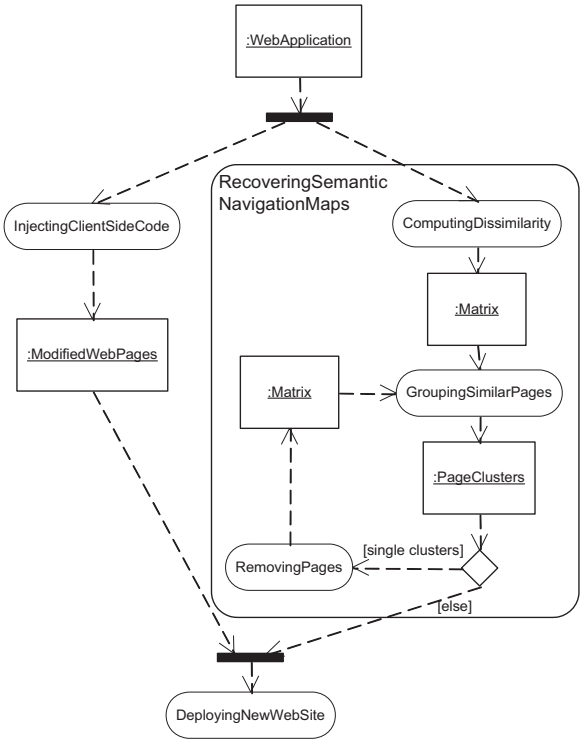


Figure 1. The overall process to evolve the navigation structure of a Web site by recovering and introducing semantic navigation maps

ComputingDissimilarity starts with extracting the textual content (i.e., the static text within the page body) of the client-side HTML pages of the given Web site; the extracted content undergoes a normalization phase in which non-textual tokens (i.e., operators, special symbols, numbers, etc.) are eliminated, terms composed of two or more words are split (e.g., "mail_address" is turned into "mail" and "address"), and terms with a length less than three characters are discarded. A stemming algorithm is also performed to reduce inflected (or sometimes derived) terms to their stem. Finally, all the terms contained in a stop word list are removed.

The concept space of a Web site is built on its normalized content adopting LSI. LSI is applied on a term-by-content matrix A, which is built on the normalized content of the considered Web site. In particular, this matrix is m x n, where m is the overall number of different terms appearing in the pages of the site and n is the number of considered pages. An entry $a_{i,j}$ of the matrix A represents a measure of the weight of

the *i-th* term in the *j-th* page. To derive the latent content semantics of a Web site we apply on this matrix a Singular Value Decomposition (SVD) [10]. Using this technique the matrix A (having rank r) can be decomposed in the product of three matrices, $T \cdot S \cdot D^T$, where S is an r x r diagonal matrix of singular values and T and D have orthogonal columns. SVD also provides a simple strategy for optimal approximate fit using smaller matrices and using only a subset of k concepts corresponding to the largest singular values in S.

Terms and pages could be graphically represented by vectors in the k space of the underlying concepts of a Web application. In our approach the rows of the reduced matrices of singular vectors are taken as coordinates of points representing the pages in a k dimensional space. To build the dissimilarity matrix of a web application we first compute the cosine between all the pairs of vectors representing the pages in the k dimensional space. Successively, the dissimilarity between the pairs of pages is computed normalizing the cosine similarity measure from 0 (when the semantics is the same) to 1 (when they have a different semantics).

GroupingSimilarPages uses a Graph-Theoretic clustering algorithm [14] to group pages according to the defined dissimilarity measure. Generally, a Graph-Theoretic clustering algorithm takes as input an undirected graph and then constructs a Minimal Spanning Tree (MST). Clusters are identified pruning the edges of the MST with a weight larger than a given threshold. Nodes within each tree of the obtained forest are included in a cluster. The Graph-Theoretic clustering algorithm is used on the strongly connected graph corresponding to the dissimilarity matrix computed in ComputingDissimilarity. In this graph, each node corresponds to a page and the weight associated to an edge represents the dissimilarity measure between the pair of pages connected by the edge. Clusters of similar pages are identified using as pruning threshold the arithmetic mean of the edge weights of the built MST. In case the clustering algorithm identifies single clusters, the subphase RemovingPages is executed. This subphase is in charge of removing from the dataset the pages that the used algorithm includes in the single clusters. The so obtained dataset is then provided again as input to GroupingSimilarPages. Note that the phases GroupingSimilarPages and RemovingPages are iterated until no single clusters are identified or no clusters at all remain. In the last extreme case, no semantic map is generated.

The phase of RecoveringSemanticNavigationMaps described in this section and its subphases should be performed as new pages are added to the site, existing pages removed or their content modified to keep semantic navigation maps consistent with the actual content of the site.

2.2 Injecting AJAX Client-Side Code

The phase *InjectingClientSideCode* extends each page of the Web site with the AJAX code able to (*i*) dynamically query the server that maintains the data on the recovered clusters and (*ii*) display the semantic navigation map associated to a given page. In particular, we use a Javascript function that interacts with a server component (a servlet) to retrieve the list of pages within the cluster of the considered page and a second function that uses the returned data to modify the HTML DOM of the page and display the map of the retrieved navigation links.

It is worth noting that the pages of the Web site are modified once for all. The injected AJAX code will dynamically query the server and recover the data on the semantic navigation map to show for the given page, each time the page is loaded in a browser.

2.3 Deploying the new Version of the Site

In the *DeployingNewWebSite* phase the enhanced version of the pages, the data on the recovered clusters of similar pages and the servlet component able to query these data are deployed on the server. In particular, the servlet needs to be deployed only once.

3 The Eclipse plug-in

We have developed a tool as an Eclipse plug-in that fully supports all the phases of the process depicted in Figure 1.

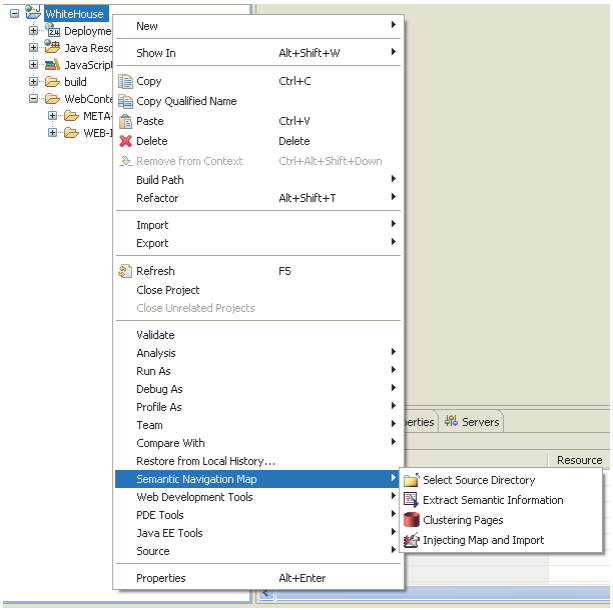


Figure 2. Semantic Navigation Map Menu

As a first operation, the plug-in requires to create an Eclipse project. To this end, the plug-in proposes a wizard which allows specifying the name of the project and its workspace. Once the project has been created, the analyst has to select the Web site whose navigation structure will be enhanced according to the proposed approach. This is possible by right-clicking the project within the *Package Explorer view* and

choosing *Select Source Directory* within the menu *Semantic Navigation Map* (see Figure 2).

At this point the plug-in will provide a window for choosing the path of the folder where the pages of the original Web site are stored. Note that these pages have to be stored on the local file system of the computer where the plug-in is installed, but future versions of the plug-in will also work on Web pages maintained on remote servers.

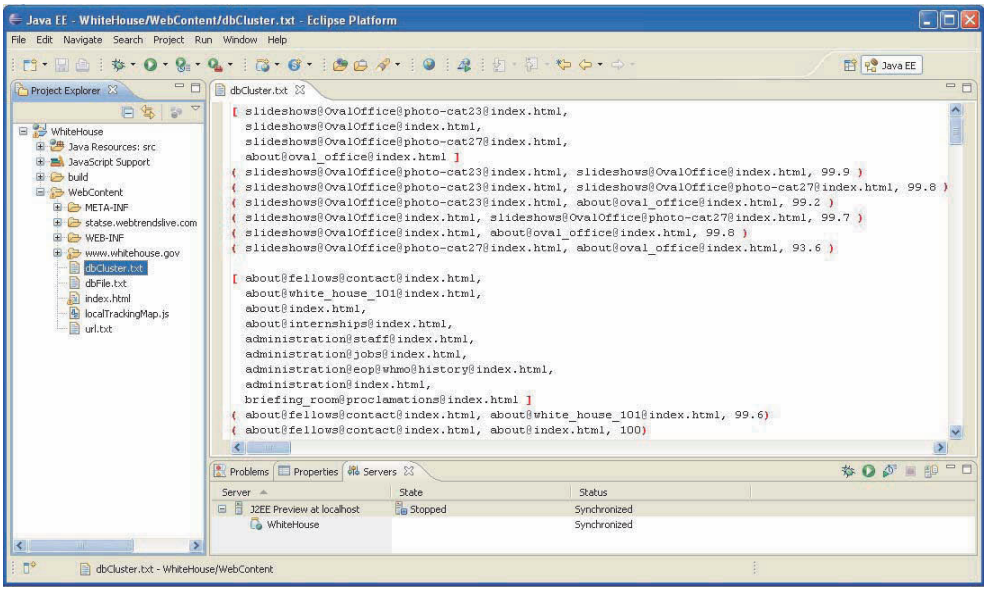


Figure 3. Improving Groups of Similar Pages

The clustering process is started by right-clicking the project and choosing *Clustering Pages* within the menu *Semantic Navigation Map* (see Figure 2). The plug–in will produce a file (i.e., dbcluster.txt) containing all the automatically identified clusters. Indeed, this file contains all the pairs of pages (actually, their paths and names) composing each cluster, and their similarity level (i.e., the cosine between the vectors of the pages in the content space). To improve the overall quality of the clustering process, the plug-in also provides for manually refining the automatically identified clusters. Indeed, if needed, the analyst can refine a cluster by modifying the set of included pages and their similarity levels. Figure 3 shows how the clustering results can be refined with the plug-in.

After the clustering process is complete, the analyst can use the plug-in to extend the pages of the Web site by automatically introducing the client side AJAX code that enables the visualization of the semantic navigation maps. Afterwards, the new pages and the Javascript library will be automatically added to the project workspace. Finally, the plug-in enables the deployment of the enhanced Web site on a suitable Web server. Note that the enhancement of the Web pages with the client side AJAX code could be also performed before performing the clustering process.

4 The Case Study

The approach and the plug-in have been assessed on several real-world Web sites. In the following subsections we present and discuss the results obtained from the National Gallery of London Web site (NGL)[2].

4.1 Experimental Context

To conduct the case study, we first dumped the pages of the selected Web site using the freeware dumper HTTrack Website Copier[3]. In the case of NGL, the dump was executed on June 9th, 2008. In particular, HTTrack Website Copier downloaded 6573 HTML pages using the index page as starting page and following the hyperlinks connecting the pages until the sixth level of depth was reached in the folder hierarchy of the Web site. The mirror of NGL has been successively analyzed to prune duplicated HTML pages created by the dumper and documents currently not considered by the current approach (e.g., PDF and Word files) and multimedia objects (e.g., JPG images and flash animations). Regarding the HTML pages, we limited the analysis to the section of the site named Collection, i.e. the section devoted to the museum entire permanent collection and long term loans. We also considered the pages presenting the works of art shown during the years in the gallery (i.e., the pages within the Exhibition section). The total number of HTML pages we have selected and considered in the presented case study is 2017.

4.2 Results

The clustering process grouped the 2017 pages of NGL into 243 different clusters. The largest cluster contained 32 pages and included pages from the collection *Scientific Instruments and Inventions from the Past* of the gallery. On the other hand, the mean number of pages within the clusters was 2.6. These and other descriptive statistics on the case study are summarized in Table 1.

Table 1. Descriptive statistics for the analyzed Web site.

	NGL
Number of analyzed pages	2017
Number of identified clusters	243
Number of iterations	5
Number of pages within single clusters	1387
Number of pages within clusters containing at least two pages	630
Number of pages within the largest cluster	32
Mean number of pages within the clusters	2.6
Number of characters within the analyzed pages	3.460K

[2] www.nationalgallery.org.uk
[3] Available at www.httrack.com

The plug-in has been also used to inject the AJAX code required to dynamically show into each of the analyzed pages the set of links connecting it with the other pages in the same cluster, i.e., the associated semantic navigation map. An example of the resulting pages is shown in Figure 4.

This page differs from its original version for the presence of the semantic navigation map shown on the right hand side. The semantic navigation map presents a set of links towards pages that have been found similar to it. Each link shows: (*i*) the title of the target page; (*ii*) the percentage of similarity with the current page obtained using LSI; (*iii*) a description of the target page obtained from data in its description meta tag.

4.3 Discussion

By analyzing the pages presenting information on the permanent collection and long term loans of the museum (i.e., the Collection section of the NGL Web site), we noted the presence of a navigation menu on the bottom right hand side. In particular, given an artist and his/her work of art, the menu enables users to directly access the pages presenting the other works of art by the same artist.

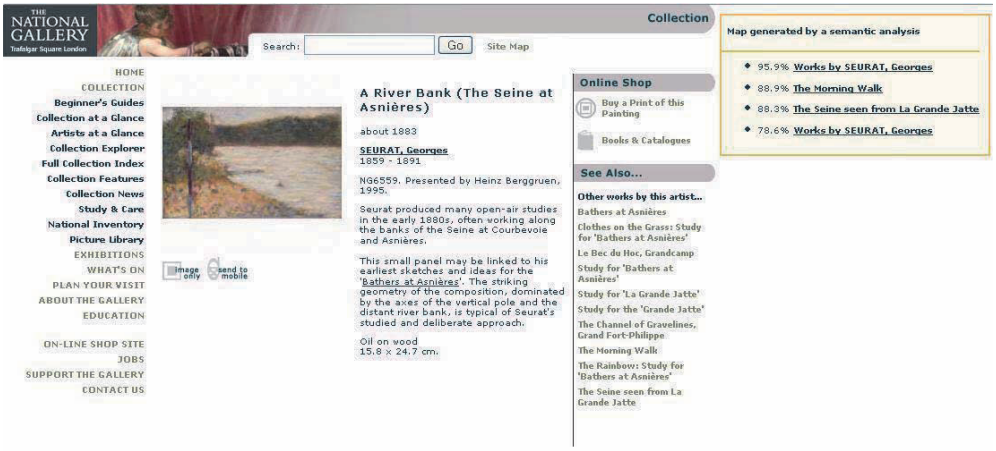


Figure 4. The enhanced version of a page

We observed that most of the links presented within the menu were also proposed by analogous links in the semantic navigation map that were automatically identified by the tool. In some other cases, we also noted that some pages of the Collection section presented a navigation menu with a number of links larger than the ones proposed by the semantic navigation maps. However, in most cases, the semantic navigation maps presented the same links as the corresponding navigational menu. This indicates that the links of the identified semantic navigation maps are generally correct.

It could be objected the fact that the recovered semantic navigation maps may result in duplicate navigation structures. This was true just in this particular Web site, were a navigation menu similar to our recovered semantic navigation maps was available, but in general this is not the case. Moreover, the purpose of the conducted case study was to assess the correctness of the produced navigation maps, a goal we can state the case study reached. Hence, we could expect that the plug-in will provide correct semantic navigation maps also for Web sites where such semantic navigation structures are not available.

5 Related Work

In the past, the problem of defining methods and tools to support software engineers in maintaining and evolving Web applications have extensively studied [1][4][7][12] [13][21][22]. For example, Ricca and Tonella [21] propose ReWeb, a tool for analyzing the structure and the evolution of static Web sites. They define a conceptual model for representing the structure of a Web site and several structural analyses relying on such model, including flow analysis, graph traversal algorithms and pattern matching. In [1], Antoniol *et al.* propose a methodology for reengineering a static Web site. The recovered design is based on the Relationship Management Data Model (RMDM) and the ER+ Model proposed within the Relationship Management Methodology (RMM) [17]. The reverse engineering phase consists of abstracting the navigational structure of the Web site and identifying the entities of the application domain and the relationship among them. The forward engineering phase is then performed by following the RMM methodology on the restructured version of the recovered ER+ diagram.

Bernardi *et al.* have recently proposed REUWA [2], an approach and a supporting tool to recover user-centered conceptual models from existing Web applications, according to the UWA methodology [30]. The approach recovers a model of the application contents, their semantic associations and access structures, and delivers it as an instance of the UWA MOF metamodel, which can be afterward used in a forward model-driven engineering process based on UWAT+.

Other authors have proposed approaches for the model-based evolution of Web applications. Garrido *et al.* in [15] introduce Web Model Refactoring as behavior preserving transformations for the navigation and presentation models of a Web application, aimed at improving its design and external quality. An approach to redesign business processes in Web applications is proposed in [27] by Tilley *et al.*, while Lowe and Kong in [20] propose NavOptim, an approach to redesign the navigation structure of a Web site.

Different authors have used clustering algorithms to identify similar Web pages. In [22] Ricca and Tonella enhance the approach based on the Levenshtein edit distance proposed by Di Lucca *et al.* in [11] (a pairs of pages is a clone if the Levenshtein edit distance [19] between the strings encoding the page structures is zero) using a hierarchical clustering algorithm to identify clusters of duplicated or similar pages to be generalized into a dynamic page. Differently from the approach proposed in [11], the distance of cloned pages belonging to the same cluster is not zero. Similarly, in

the [28] authors propose a semiautomatic approach based on an agglomerative hierarchical clustering algorithm to identify and align static HTML pages whose structure is the same and whose content is in different languages. The aligned multilingual pages are then merged into MLHTML pages. De Lucia *et al.* [9] also propose a semiautomatic approach based on the Levenshtein edit distance to compute the similarity of two pages at the structural, content, and scripting code levels. Clones are characterized by a similarity threshold that ranges from 0%, for different pages, up to 100%, for identical pages. An approach based on a general process that first compares pages at the structural level (i.e., the Levenshtein edit distance) and then groups them using a competitive clustering algorithm (i.e., Winner Takes All) is proposed by De Lucia *et al.* in [7].

Ricca *et al.* in [23] describe the results of an empirical study to group pages in Web site according to their content. Clustering is based on the similarity of the keywords within the page content. Keywords are weighted so that more specific keywords receive a higher score. The authors use the Natural Language Processing techniques to weight each keyword, according to its relevance and specificity. Similar pages are then grouped together by adopting a hierarchical clustering algorithm.

In [8] a comparison among clustering algorithms to identify similar pages at the content level is presented. In this work, three variants of the agglomerative clustering algorithm, i.e., a divisive clustering algorithm, k-means, and a competitive clustering algorithm, have been considered. The study reveals that the investigated clustering algorithms generally produce comparable results. To compare pages, an LSI based similarity measure is used.

6 Conclusion and Future Work

Our research is meant to automatically recover semantic relations between the pages of a Web site and build semantic navigation maps, accordingly. To this aim, we have defined a process that uses Latent Semantic Indexing (LSI) to compute a dissimilarity measure between the pages of a site, and a Graph-Theoretic clustering algorithm to group pages showing similar or related content, based on the computed dissimilarity measure. Finally, we use AJAX code to enhance the navigation structure of the Web site with a set of hyperlinks connecting each page to other pages within the same cluster. We defined this set of links as Semantic Navigation Map.

To automate the application of the approach, we have developed a supporting tool as an Eclipse plug-in. Both the approach and the tool have been validated in a case study.

Semantic navigation maps may be particularly useful when the navigation structure of the site is found to be not properly designed or when it has degraded during the Web site life-cycle. However, even properly designed Web sites may benefit from the use of the semantic navigation maps. In fact, they represent an additional navigation structure, complementary to those implementing the navigation model obtained during the design phase, and offering navigation paths based on content semantics and similarity.

In the future, we plan to apply the approach on other Web sites of different size and application domain. A further experimentation will be conducted to evaluate the completeness and correctness of the clusters of pages identified as similar at the semantic level by the tool prototype.

It will be also worth extending both the approach and the tool to make them suitable for dynamic Web sites, i.e., Web sites for which the content showed into pages and the accessible pages varies depending on some context variable (the user profile, location, etc.). In this direction, we are currently working on developing software components to be integrated in different and widely employed Web applications, e.g., CMSs, e-learning platforms, and e-commerce application frameworks.

Finally, with the intent of satisfying the user's expectation and requirements, we will also investigate the possibility of adapting the navigation maps considering the user profile.

References

1. G. Antoniol, G. Canfora, G. Casazza, and A. De Lucia. "Web Site Reengineering using RMM". In *Proc. of the 2nd International Workshop on Web Site Evolution*, Zurich, Switzerland, 2000, pp. 9-16.
2. M. Bernardi, G. A. Di Lucca, and D. Distante, "Reverse Engineering of Web Applications to Abstract User-Centered Conceptual Models". In *Proc. of the 10th International Symposium on Web Site Evolution*, IEEE CS Press, 2008, pp. 55-64.
3. C. Boldyreff and P. Tonella. "Web Site Evolution". Special Issue of the *Journal of Software Maintenance*, vol. 16, no 1-2, 2004, pp. 1-4.
4. C. Boldyreff and R. Kewish. "Reverse Engineering to Achieve Maintainable WWW Sites". In *Proc. of the 8th IEEE Working Conference on Reverse Engineering*, Suttgart, Germany, IEEE CS Press, 2001, pp. 249-257.
5. S. Ceri, P. Fraternali, and A. Bongio. "Web Modeling Language (WebML): a Modeling Language for Designing Web Sites". *Computer Networks* 33 (1-6), 2000, pp. 137-157.
6. D. Cran, E. Pascarello and J. Darren. "Ajax in Action" Manning Publications Co. October, 2005. ISBN: 1932394613
7. A. De Lucia, G. Scanniello, and G. Tortora "Identifying Similar Pages in Web Applications using a Competitive Clustering Algorithm". In *Journal on Software Maintenance and Evolution*, vol. 19, no. 5, September-October 2007, Wiley, pp. 281-296.
8. A. De Lucia, M. Risi, G. Scanniello, and G. Tortora "Clustering Algorithms and Latent Semantic Indexing to Identify Similar Pages in Web Applications". In *Proc. of the 9th IEEE International Symposium on Web Site Evolution*, Paris, France, October 5-6, 2007, IEEE CS Press, pp. 65-72.
9. A. De Lucia, R. Francese, G. Scanniello, and G. Tortora. "Identifying Cloned Navigational Patterns in Web Applications". In *Journal of Web Engineering*, vol. 5, no.2, Rinton Press, 2006, pp. 150-174.
10. S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. "Indexing by Latent Semantic Analysis". In *Journal of the American Society for Information Science*, no. 41, 1990, pp. 391-407.
11. G. A. Di Lucca, M. Di Penta, and A. R. Fasolino. "An Approach to Identify Duplicated Web Pages". In *Proc. of the 26th Annual International Computer Software and Application Conference*, Oxford, UK, IEEE CS Press, 2002, pp. 481-486.

12. G. A. Di Lucca, M. Di Penta, G. Antoniol, and G. Casazza. "An Approach for Reverse Engineering of Web-based applications". In *Proc. of the 8th IEEE Working Conference on Reverse Engineering*, Suttgart, Germany, IEEE CS Press, 2001, pp. 231-240.
13. D. Eichmann "Evolving an Engineered Web". In *Proc. of International Workshop Web Site Evolution*, Atlanta, GA, 1999, pp. 12-16.
14. P.J. Flynn, A. K. Jain, and M. N. Murty "Data Clustering: A Review". In *ACM Computing Surveys*, vol. 31, no. 3, 1999, pp. 264-323.
15. A. Garrido, G. Rossi, and D. Distante, "Model Refactoring in Web Applications". In *Proc. of the 9th International Symposium on Web Site Evolution*, IEEE CS Press, 2007, pp. 89-96.
16. F. Garzotto and V. Perrone. "On the Acceptability of Conceptual Design Models for Web Applications". In *Proc. of Conceptual Modeling for Novel Application Domains* – ER'03 Workshops. Chicago, US, October 2003, LNCS – 2814/ 2003, pp. 92-104.
17. T. Isakowitz, E. A. Stohr, and P. Balasubramanian, "RMM: a Methodology for Structured Hypermedia Design", *Communications of the ACM*, vol. 38, no. 8, 1995, pp. 34–44.
18. G. Kappel, B. Pröll, S. Reich, W. Retschitzegger (Eds.), "Web Engineering: The Discipline of Systematic Development of Web Applications", Wiley, 2006.
19. V. L. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals". *Cybernetics and Control Theory*, vol. 10, 1966, pp. 707-710.
20. D. Lowe and X. Kong, "NavOptim Coding: Supporting Website Navigation Optimisation using Effort Minimisation". In *Proc. of 2004 IEEE/WIC/ACM International Conference on Web Intelligence*, Beijing, China, 2004, IEEE CS Press, pp. 91-97.
21. F. Ricca and P. Tonella, "Understanding and Restructuring Web Sites with ReWeb", *IEEE Multimedia*, vol. 8, no. 2, 2001, pp. 40-51.
22. F. Ricca and P. Tonella, "Using Clustering to Support the Migration from Static to Dynamic Web Pages". In *Proc. of International Workshop on Program Comprehension*, Portland, Oregon, USA, 2003, pp. 207-216.
23. F. Ricca, P. Tonella, C. Girardi, and E. Pianta, "Improving Web site understanding with keyword-based clustering". In *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 20, n. 1, 2008, pp. 1-29.
24. G.Scanniello, D. Distante, M. Risi, "Using Semantic Clustering To Enhance the Navigation Structure of Web Sites". In *Proc. of the 10th International Symposium on Web Site Evolution*, IEEE CS Press, 2008, pp. 55-64.
25. D. Schwabe and G. Rossi, "An object-oriented approach to Web-based application design". *Theory and Practice of Object Systems* (TAPOS). Special Issue on the Internet, vol. 4, no. 4, October, 1998, pp. 207-225.
26. W. Schwinger and N. Koch, "Modeling Web Applications", In "Web Engineering", Chapter 3. G. Kappel, B. Pröll, S. Reich, W. Retschitzegger (Eds.). Wiley, 2006.
27. S. Tilley, D. Distante, and S. Huang. "Web Site Evolution via Transaction Reengineering". In *Proc. of the 6th IEEE International International Symposium on Web Site Evolution*, IEEE CS Press, 2004, pp. 31-40.
28. P. Tonella, F. Ricca, E. Pianta, and C. Girardi, "Restructuring Multilingual Web Sites". In *Proc. of the 18th International Conference on Software Maintenance*, Montreal, Canada, IEEE CS Press, 2002, pp. 290-299.
29. G. Tsakonas, C. Papatheodorou, "Exploring usefulness and usability in the evaluation of open access digital libraries". In *Proc. of International Journal of Information Processing and Management*, vol. 44 (3), May 2008, Pergamon Press Inc, pp. 1234-1250..
30. UWA Consortium, "Ubiquitous Web Applications". In *Proc. of the eBusiness and eWork Conference 2002*, Prague, Czech Republic, October 2002.

An Empirical Evaluation of the Eclipse Framework

Mariarosaria Lapolla[1], Michele Risi[2], Giuseppe Scanniello[1]

[1] Department of Mathematics and Computer Science, University of Basilicata, Potenza, Italy
[2] Department of Mathematics and Computer Science, University of Salerno, Fisciano, Italy
mariarosaria.lapolla@tiscali.it, mrisi@unisa.it, giuseppe.scanniello@unibas.it;

Abstract. This paper presents a preliminary investigation, based on the combined use of two techniques: a questionnaire-based survey and an empirical analysis, to assess the effectiveness and efficacy of the Eclipse framework to support novice developers in the development and maintenance of Java source code. The satisfaction of the developers has been investigated as well. The context of this study was constituted of Bachelor students in Computer Science at the University of Basilicata. The survey shows a good satisfaction degree of all the involved subjects, while the empirical analysis reveals that the effectiveness and the efficacy of the Eclipse framework can be considered appropriate.

Keywords: Empirical Study, Software Maintenance.

1 Introduction

The academic and industrial realities are proposing several software tools to support software engineers and developers during the development and maintenance. The adoption of these tools could not be made without a systematic and quantitative evaluation, in particular concerning the usability. In fact, if a tool is difficult to use, it will hardly be adopted, no matter how useful it may be. Although this is a critical point that is widely recognized and has been extensively discussed in the past, very few studies have been conducted for evaluating these tools.

The usability cannot be related to a precise aspect of a software system, but it strongly depends on the intended use of the system. According to the standard ISO 9241-11 [4], usability represents the degree of a software product to be used by specific users to pursue specific goals within a specific usage context effectively, efficiently, and with satisfaction. So far, few investigations have been carried out to assess these concerns in Integrated Development Environments (IDEs) both commercial and open source.

Usability methods have been developed for many years to evaluate the efficiency, interaction flexibility, interaction robustness, and quality of user interfaces [2][7][8]. Several usability evaluation techniques and guidelines are proposed in the literature aiming at planning and realize usability studies [6][11][13]. For example, Ricks and Arnoldy [11] assert that all the usability studies follow the same basic steps although

a wide variation of products can be analysed (e.g. software applications, printers, web sites, etc…). Generally, usability produces several benefits [7][13] ranging from the reduction of the training costs to the improvement of the user satisfaction. In particular, usability is often the major concern in the adoption of software tools in industry. Usability cannot be generally related to a precise aspect of a software system, but this concept varies depending on the intended use of the system. Hence, it is easy to understand that the evaluation of IDEs is generally very hard since they provide a huge number of functionalities and their user interface is very complex as they are intended for users with specific competences.

Usability methods can be broadly divided in two groups: expert evaluation and user studies. Expert evaluation includes cognitive walkthrough, model-based evaluation, and heuristic evaluation and can be used since the initial phases of the software development process. On the other hand, user studies are based on the direct involvement of users through observations, interviews, and questionnaires and include empirical methods and observational methods. These techniques are quite efficient for evaluating usability of user interface when concrete tasks are considered.

This paper presents a preliminary user study to assess the effectiveness and efficacy of the version 3.4.1 of the Eclipse framework to support novice programmers in the development and maintenance of Java source code. The satisfaction of the programmers has been also investigated here. The presented study is based on the combined use of two techniques: a questionnaire-based survey and an empirical analysis. The context of user study was constituted of Bachelor students in Computer Science at the University of Basilicata. The survey shows a good satisfaction degree of all the involved subjects, while the empirical analysis reveals that the Eclipse effectiveness and efficacy is appropriate.

The remainder of the paper is organized as follows: Section 2 shows the design of the presented user study. Section 3 discusses the observed results and Section 4 concludes the paper.

2 The design of the usability study

In this section we present the design of the empirical study presented here. In particular, we describe the data set and then the techniques we have adopted to assess the efficacy, efficiency of the tool and the users' satisfaction.

2.1 The data set

The study was conducted in a research laboratory at the University of Basilicata. Data for the study have been gathered considering a group of eleven volunteers. They were Bachelor students in Computer Science at the University of Basilicata. All the involved subjects had object-oriented programming experience and good knowledge of IDEs both commercial and open source. In particular, all the subjects passed the following programming exams: procedural and object oriented. On the other hand, the majority of them passed the advanced course of object oriented programming language. Table 1 summarizes the subjects' background. In particular, it presents for

each subject the known programming languages and IDEs. Note that all the subjects master Netbeans, while only one subject has a very good experience on the Eclipse framework. This was due to the fact that Netbeans was the IDE the subjects used within the procedural and object oriented programming courses. This can be considered interesting as it reproduce a possible scenario where a small software industry is intended to replace the used IDE (i.e., Netbeans) in favor a new one (i.e., Eclipse). A careful reader may object the fact that the use of Bachelor students as subjects may affect the soundness of this scenario. Nevertheless, the business of a small software house, especially in the south of Italy, is based on programmers with a Bachelor degree in computer science, thus considering the subjects of this study not far from junior programmers.

Table 1. Subjects' Background.

ID	Programming Languages	IDEs
1	Java, C#, C, C++, Fortran, Pascal, PHP	NetBeans, Visual Studio, #Develop, Eclipse
2	Java, C#, C	Netbeans, Visual Studio
3	C++, C#, Java, Fortran	Netbeans
4	Java, C++, C#	Netbeans, Visual Studio
5	Java, C#, Fortran, Pascal, C	Netbeans, Visual Studio, Visual C#
6	Java, C#	Netbeans
7	Java, C#, C++	Netbeans
8	Java, C#, C, C++	Netbeans
9	Java, C#, C++, Fortran	Netbeans
10	C#, C++, Fortran	Netbeans
11	C++, Fortran, C, Java, C#	Netbeans, Visual Studio, Visual Web

The study has been divided in four steps and performed in one-to-one session (i.e. a supervisor for each subject) using the think aloud technique. In the first step all the subjects have been introduced the Eclipse framework and its main functionalities. Successively, they have been asked to use the tool for 10 minutes without invoking any kind of tutor support. The subjects were asked to perform two tasks in the third step. The first task concerned the creation of a Java project and the implementation of some classes using the Eclipse features (e.g., the automatic generation of the getter and setter methods for each field). They were also asked to organize the implemented classes in packages. The second task regards the execution of maintenance operations on existing Java code. In particular, the subjects were asked to correct syntactic and semantic errors within a Java software system created by one of the author. This software was in charge of managing a gym and its customers. To avoid that the subjects exchange information on the errors within the considered software, we have developed a different task for each subject. The complexity to the tasks was however comparable as the number of errors was the same (i.e., 5) and the needed effort to correct them was nearly the same. In the fourth step, the subjects have filled in a post experiment survey questionnaire to achieve information on their satisfaction. Details on the questionnaire will be provided in the following.

During the experiment the supervisor did not provided any help to the subjects to avoid biasing the experiment. He only wrote the comments and problems of the subjects, when they spoke aloud. For each subject the needed time to accomplish the experiment was annotated as well.

It is worth mentioning that the tasks had to be carried out respecting some constraints. In particular, for the first task the subjects were asked to save the projects using their surname in the directory 'workspace'. They had to use the Eclipse features to create both the main method of their application and the methods get and set of each created field.

2.2 Design of the survey

Surveys can be used to collect information on the preferences and reactions of some selected subjects. This technique is particularly valuable for usability investigations since it allows designers to understand the user reaction and to identify possible problems. Typically, questionnaires are used to capture the user point of view in order to assess aspects of usability, validity and/or reliability of user interfaces [1]. Questions should be ordered in such a way that each question does not influence the response to subsequent questions and they should be presented in the same order to the subjects.

The questionnaire we used in this study contained 37 questions arranged in four categories: *subject experience*, *Eclipse satisfaction*, *first task*, and *second task*. In particular, it contained both open (required just filling in a comment or text) and closed questions. To collect information on the subjects' programming experience and on the known programming languages and IDEs we have defined the first part of the survey questionnaire. Special emphasis has been posed to Eclipse. The questions of this category (i.e., subject experience) are shown in Table 2.

Table 2. Subject experience.

ID	Questions
q1	Have you never used Eclipse framework? (yes- not)
q2	In case you answered yes to q1, how do you judge your knowledge? (scant – sufficient – good - very good)
q3	Which IDEs have you used in the past?
q4	How do you judge your knowledge about these IDEs? Indicates for each of them the knowledge level (i.e., scant, sufficient, good, or very good)
q5	Among the known IDEs what is the simpler to use?
q6	Which programming languages do know you?
q7	How do you judge your programming experience? (scant – sufficient – good - very good)
q8	Do you have professional programming experience? (yes-not)

To get information about the subjects' satisfaction on the Eclipse usage (i.e., Eclipse satisfaction category) the questions of Table 3 have been defined. On the other hand, the questions regarding the first and second tasks are reported in Table 4 and **Errore. L'origine riferimento non è stata trovata.**, respectively. All the questions reported in the tables from 3 to 5, except q22, expected closed answers according to a five point Likert scale [9] (from always to never). The question q22 is open and requires that the subjects provide their satisfaction on the usage of the analyzed version of the Eclipse framework.

Table 3. Eclipse Satisfaction.

ID	Questions
q9	Did you have any problem to use the Eclipse functionality?
q10	Did you have any problem using Eclipse?
q11	The time to learn a functionality of the Eclipse framework is appropriate.
q12	The user interface is pleasant and easy to understand.
q13	Did you have any problems to find the needed functionality
q14	Did you use the Eclipse help?
q15	The help provides a suitable support.
q16	The used Eclipse functionality is easy to master.
q17	Eclipse provides a suitable support to the programmer.
q18	The Eclipse tool bar is easy to understand.
q19	Did you need any external help during the experimentation?
q20	Eclipse promptly reacts to each required command/functionality.
q21	I have planned to use Eclipse in the future.
q22	Provide your personal observations about the used version of the Eclipse framework.

Table 4. First task.

ID	Questions
q23	Did you have any problems in importing external packages within the implemented program?
q24	The number of operation to accomplish the tasks was appropriated.
q25	The creation of the get and set methods was simple.
q26	The effort to create the signatures of the methods (e.g., main) is appropriate.
q27	The suggestions on the errors highlighted in red were easy to understand.
q28	Did you understand the he messages provided by Eclipse?
q29	Automatic code completion is useful.

Table 5. Second task

ID	Questions
q30	It was difficult to understand the software.
q31	Did you solve all the identified syntactic errors?
q32	Did you solve all the identified semantic errors?
q33	The suggestions provided by the Eclipse framework were useful to identify and correct the syntactic and semantic errors.
q34	Did you remove all the syntactic and semantic errors, thus enabling the system run?
q35	The classes were properly named.
q36	The project was properly structured.
q37	The behavior of the classes is clear.

2.3 Design of the empirical analysis

In this study we have considered the variables reported in Table 6. The first and the second column contain the name of the variables and their description, respectively. The type of the variable is reported in the third column. The variable MN1 denotes the effort, expressed in terms of minutes, to carry out the first task, while MN2 denotes the effort to carry out the second task.

Table 6. Selected variables.

Variable	Description	Scale
MN1	Number of minutes required to perform the first task.	Ratio
MN2	Number of minutes required to perform the second task.	Ratio
LOC1	LOCs coded by the subject within the first tasks.	Ratio
ER2	Number of not discovered errors within the second tasks	Ratio

In order to graphically represent the distribution of the size measures of the considered usability study, we have adopted the boxplots. They are widely employed in exploratory data analysis since they provide a quick visual representation to summarize the data using: the median, upper and lower quartiles, minimum and maximum values, and outliers. Note that we also used this kind of representation to summarize the answers of the survey questionnaire.

3 Results

In this section, we present the results of the post experiment survey questionnaire and draw some conclusions with respect to the empirical analysis.

3.1 Survey results

The answers of the subjects are summarized in Table 7. According to the defined categories (i.e., Eclipse satisfaction, first task, and second task) the answers are also visually summarized in Figure 1. The values on the *x* axes range from 1 (always) to 5 (never). The boxes show that the answers are generally concordant on each question. However, some consideration are due on some questions of the survey.

Table 7. Survey questionnaire results.

Questions	always	often	neutral	some time	Never
q9	1	3	6	1	0
q10	0	3	5	2	1
q11	0	1	4	4	2
q12	0	5	3	3	0
q13	0	5	3	3	0
q14	0	0	0	1	10
q15	0	0	0	1	0
q16	2	5	4	0	0
q17	0	1	4	4	2
q18	2	5	3	1	0
q19	0	1	5	3	2
q20	2	6	2	1	0
q21	0	2	4	1	4
q23	5	2	4	0	0
q24	5	4	2	0	0
q25	7	3	0	1	0
q26	9	0	0	1	1
q27	3	6	2	0	0
q28	4	2	4	1	0
q29	4	4	3	0	0
q30	4	4	3	0	0
q31	6	4	1	0	0
q32	5	2	3	0	1
q33	3	3	5	0	0
q34	7	4	0	0	0
q35	7	4	0	0	0
q36	6	4	1	0	0
q37	7	4	0	0	0

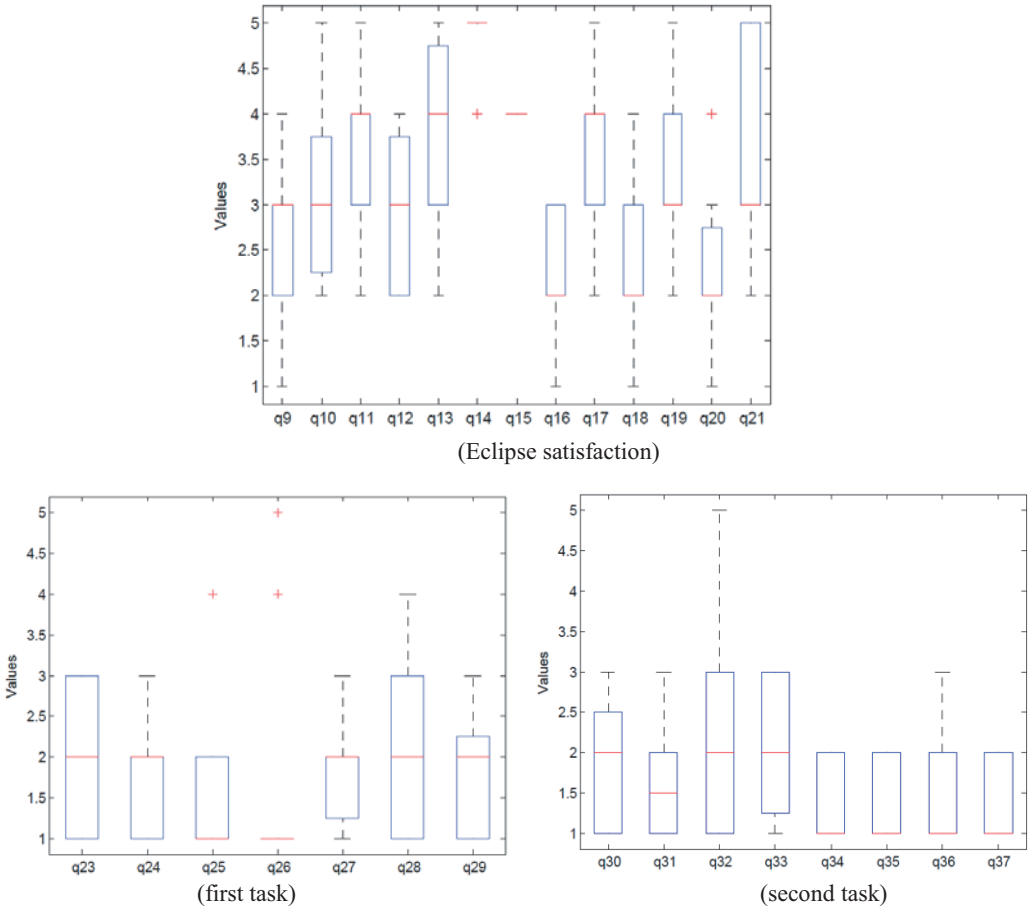


Figure 1. Boxplots of the questionnaire answers.

The majority of the subjects did not have any problem, while using Eclipse and its functionality. The subjects did not found the Eclipse user interface pleasant. However, they stated that the Eclipse user interface effectively supported them in finding and properly using unknown functionality, even without using the help.

Regarding the subjects' satisfaction in writing new source code (i.e., the first tasks) the results is generally concordant. In particular, the subjects expressed a very good judgment on the Eclipse framework. The only problem that they found concerned the import of an external package. This could be due to the fact that the steps to perform were different with respect to Netbeans. This is another issue that deserves a further future investigation (see Conclusion section).

A good satisfaction degree was manifested by all the subjects on the maintenance task. This task was also considered clear. In general, the answers of the subjects were concordant.

3.2 Empirical analysis results

The time that the subjects spent to accomplish both the defined tasks are reported in Table 8, while Table 9 shows descriptive statistics (i.e., variance, standard deviation, mode, min, max, and mean) of the considered variables. The times to perform the two tasks are also visually summarized by the boxplots in Figure 2. In particular, Figure 2(a) does not present outliers. Although the box is skewed, a good distribution of the time to accomplish the first task is shown. An outlier is shown in Figure 2(b). Also in this case the distribution of the time can be considered good.

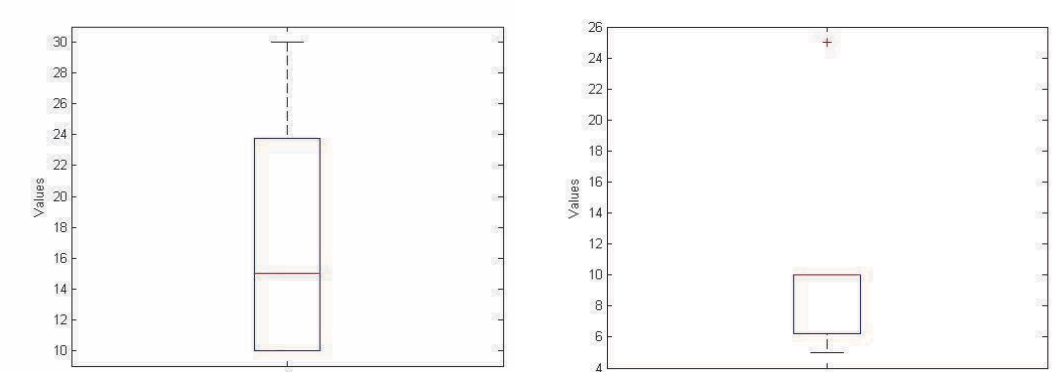
The best and the worst time to accomplish the first task were 10 and 30 minutes, respectively. On the other hand, 5 minutes was the best time to perform the second task, while the worst time was 25 minutes. The mean time to accomplish the first task was 17.27, while 10 is the mean time to accomplish the second task. As expected, the time to accomplish the first task presents more variability with respect to the time to accomplish the second tasks. This confirms the fact that the complexity of the second task was nearly the same for each subject involved in the user study.

Table 8. Row data.

Subject	MN1	MN2	LOC1	ER2
1	10	5	37	0
2	10	10	33	1
3	25	10	37	1
4	15	5	47	1
5	30	25	33	1
6	30	10	64	1
7	10	5	36	1
8	15	10	40	1
9	15	10	13	1
10	20	10	31	1
11	10	10	29	1

Table 9. Descriptive statistics.

Variable	Variance	Standard Deviation	Mode	Min	Max	Mean
MN1	61.82	7.86	10	10	30	17.27
MN2	30	5.47	10	5	25	10
LOC1	154.25	12.41	37	13	64	36.36
ER2	0.09	0.30	1	0	1	0.90



(first task)**(a)** (second task)**(b)**

Figure 2. Boxplots of MN1 and MN2.

The programs the subjects developed within the first task can be considered comparable in terms of complexity and size (see Table 9). In fact, the majority of the subjects coded simple programs to execute arithmetic operations and to show on the video the results.

The supervisor noted that the majority of the subjects made nearly the same mistakes, while performing the first task. In particular, these mistakes concerned the use of keyboard shortcut (not provided by Eclipse) that the subjects widely employed within Netbeans. The habit of software developers is an interesting point that requires a further empirical investigation.

Regarding the maintenance operations to perform in the second task, we can note that all the subjects were able to remove the majority of the errors (i.e., 4 mistakes of 5) both syntactic and semantic present in the understudy software system (see Table 9). Indeed, all the subjects except one, who identified all the errors, were not able to discover the same type of semantic error (i.e., a bug within an iterative cycle to scan a list). It is worth noting that all the syntactic errors were discovered by the subjects, thus indicating that the Eclipse framework effectively supports the developers in this phase. A further study is however needed to confirm or contradict this result.

3.3. Discussion

The general judgment of the Eclipse framework has been considered appropriate both to write source code and to comprehend and modify an existing one. Despite this encouraging result a further consideration is due. In particular, the subjects' satisfaction and performances have been probably influenced by the fact that they were very familiar with Netbeans and were not very familiar with the Eclipse framework. This concern is very interesting from the technology transfer point of view [10][12].

The subjects manifested the will of adopting Eclipse in the future (see box q21 in Figure 1) as it is easy to use and to learn. The fact of having an active community of developers was identified as a strength point to adopt it. Some issues have been

however aroused by the subjects on the graphical user interface of Eclipse, in general, and its perspectives, in particular. Once again, this could be due to the fact that the subjects unintentionally performing a comparison between the IDE they better known, i.e., Netbeans, with the Eclipse framework.

3.4 Threats to validity

The threats to validity that could affect both study (i.e. internal, construct, and external validity threats) are described in the following. The internal validity threats are relevant for our study as we aimed at concluding that Eclipse effectively supports novice developers. The internal validity threats are mitigated by the experiment design. The surveys also revealed that the subjects found clear everything regarding the tasks. This threat was also mitigated as the subjects knew neither the objective of the experiment nor its hypotheses.

The construct validity threats could be present in this study. However, the measurement of the variable was performed considering the times gathered by the supervisor. Finally, the survey questionnaire was designed using standard ways and scales [9].

External validity refers to the approximate truth of conclusions involving generalizations. This kind of threat is always present when students are used as subjects [3]. Nevertheless, before conducting the study, we identified the population we would like to investigate the Eclipse usage (i.e., novice software developer with some knowledge in a different IDE), and then we drew a fair sample from that population and conducted the study with subjects belonging to this sample. Moreover, none of the subjects abandoned the study. To confirm or contradict the achieved results replications using a larger dataset will be conducted. In the future we also plan to conduct a further investigation to compare the effectiveness and efficacy of Eclipse with respect to other widely used IDEs, e.g., Netbeans or Visual Studio.

It is worth noting that conclusion validity threats are not present in this study as statistical tests have not been performed to reject null hypotheses.

4 Conclusion

Software houses are generally conservative. They only act when they are forced to [12]. Such a position makes the adoption of a new technology quite challenging. This preliminary user study aims at providing some outcomes to increase the body of knowledge about the risks and benefit of introducing the Eclipse framework within the software industry. This was the main motivation for conducting our user study, based on the combined use of two techniques: a questionnaire-based survey and an empirical analysis. The study was conducted with novice developers (i.e., Bachelor students at the University of Basilicata) to evaluate their satisfaction when using the Eclipse framework and to assess the framework effectiveness and efficacy. To accomplish the study, the subjects were asked to develop a simple Java program and to maintain an existing Java software system. The survey shows a good satisfaction degree of all the involved subjects, while the empirical analysis reveals that the

effectiveness and the efficacy of the Eclipse framework can be considered appropriate. It is worth noting that at best of our knowledge this is the first study that aims at assessing the effectiveness and efficacy of the Eclipse framework to support novice developers in the development and maintenance of Java source code.

In the future we plan to conduct a further analysis on the gathered data using statistical tests. For example, we will investigate whether the programming experience and the known IDEs influence subjects' satisfaction and performances. Future work will be also devoted to replicate the study in different contexts with subject with different background. Finally, we aim at comparing the IDEs Eclipse and Netbeans. The motivation for conducting this comparison relies on the fact that they are widely adopted both in academic and industrial contexts.

References

1. L. Briand, K. E. Emam, D. Surmann, I. Wiekzorek, K. Maxwell, "An Assessment and Comparison of Common Software Cost Estimation Modeling Techniques". In Proc. of International Conference on Software Engineering, Los Angeles, USA, 16-22 May, 1999, pp. 313-322.
2. A. Dix, J. Finlay, G. Abowd, R. Beale, Human-Computer Interaction, 2nd edition, Prentice-Hall, 1994.
3. J.E. Hannay and M. Jørgensen, "The Role of Deliberate Artificial Design Elements in Software Engineering Experiments". IEEE Transactions on Software Engineering, vol. 34, no. 2, 2008, pp. 242-259.
4. ISO 9241-11 "Ergonomics of human-system interaction - Part 11 Guidance on usability", 1998.
5. A. Koohang, "Expanding the concept of usability". In Informing Science: The International Journal of an Emerging Transdiscipline, Eli Cohen Editor, vol. 7, 2004, pp. 129-141.
6. M. Matera, M.F. Costabile, F. Garzotto, and P. Paolini, "SUE Inspection: an Effective Method for Systematic Usability Evaluation of Hypermedia", IEEE Transactions on Systems, Man and Cybernetics- Part A, vol. 32, no. 1, 2002, pp. 93-103.
7. J. Nielsen. "Usability engineering". In New York: Academic Press, 1993.
8. J. Nielsen. "Usability testing", In Handbook of human factors and ergonomics, G. Salvendy Editor, New York: John Wiley, 1997, pp. 1543-1568.
9. A. N. Oppenheim, "Questionnaire Design, Interviewing and Attitude Measurement", Pinter Publishers, 1992.
10. S. L. Pfleeger and W. Menezes, "Marketing technology to software practitioners", IEEE Software, vol. 17, no. 1, 2000, pp. 27-33.
11. K. Ricks,B. A. Arnoldy, "How to conduct your own usability study". In Proceedings of International Professional Communication Conference, IEEE Computer Society Press, Portland, OR, USA, 2002, pp. 115-126.
12. E.M. Rogers. "Diffusion of Innovation", 4th edition, Free Press, New York, 1995.
13. B. Shneiderman, "Designing the User Interface", Third Edition, Addison Wesley Longman, Inc. 1998.
14. E. Stensrud, I. Myrtveit, "Human Performance Estimation with Analogy and Regression Models". In Proceedings of the METRICS 98 Symposium, 1998, Bethesda, Maryland, USA, pp. 205-213.

Exploring Eclipse possibilities to realize Mashups

Paolo Maresca[1], Giuseppe Marco Scarfogliero[2], Lidia Stanganelli[3],
Giacomo Franco[4], Gianfranco Nota[5]

[1] Università degli Studi di Napoli Federico II paolo.maresca@unina.it,
[2] Università degli Studi di Napoli Federico II, g.scarfogliero@studenti.unina.it
[3] Università degli Studi di Genova lidia.stanganelli@unige.it,
[4] IBM Italia S.P.A., giacomo.franco@it.ibm.com,
[5] Università degli Studi di Salerno nota@unisa.it

Abstract. In this paper we propose Eclipse platform as a great solution to the problem of designing and developing mashup applications. Thanks to its modular and pliable architecture, Eclipse can be used to satisfy all mashup's application needs, allowing developers to act at each level of a mashup application, from the simple action of retrieving, managing and mashing data to combine complex structures or architectures to obtain real mashup applications and to bring them to web, using all new Eclipse's web 2.0 features. The combined use of the Open Source technology of Eclipse and the mashup philosophy gives to enterprises, universities, research centres, freelancers, industries, and to each person that needs an application built to satisfy his own needs, the possibilities to obtain the application they need in a simple and quick way, with all the support of the Eclipse community. For enterprises this means building situational applications to manage all the enterprise business processes that the major Enterprise Applications cannot treat due to the particularities of them. The specific nature of these processes and their less relevance in the global mission of an enterprise make them less attractive for software houses and customers due to the high costs of designing and development. Costs that the proposed solution can cut easily. For universities and research centres this solution can be the right way to integrate and mash information and data belonging to these two worlds, often too less interconnected. For freelancers and other people this can be a real possibility to simplify their activities and improve productivity.
In conclusion we show also the aim the governs the Eclipse community and the constant rejuvenation process that gives us more trust on the future possibility in this way.

1 Introduction

Today, the information can be considered one of the main resources on which is concerned large part of the economical, political and social interests. Some theorist, looking at our society, speaks about the Information Age, basing on the idea that the current age is characterized by the ability of people to transfer information freely, and to have instant access to a shared knowledge. This idea is the direct consequence of the digital revolution in information and communication technologies, that has created the platform for a free flow of information, ideas and knowledge across the globe. This revolution has made a profound impression on the way the world functions. Mass media broadcast in real time information belonging to every zone of the globe, telecommunications interconnect people, the Internet has become an important global resource, on which are based many business and social relations.
So the main problems are to manage this enormous variety of information, to extract knowledge from data, to link together right information to build new knowledge, because data without a right contextualization cannot bring any useful information.
The problem of information management is very complex and cannot be treated exhaustively in this paper, but everyone will agree that in a global connected world there is the need to select, manage, use and mash all information belonging to the different information sources to obtain in a quicker and easily way the answer a person need.

The computer science, with its storage, elaboration, and communication capabilities, can help the human in this aim. The philosophy that governs mashups can be a possible solution to this problem: giving everyone the possibility of mashing data belonging to different information sources and use them to build his own web application that can help him to solve by himself his particular problem.

This approach can suit many different scenarios in which the aggregation and manipulation of data is needed. Let's think for example the simple case of a complex enterprise whose departments are geographically distant or less interconnected, many times managing information is very difficult. Or let's think about an educational scenario where some university, private institutes or other organizations produce some resources on the same argument, for a student or a researcher involved in the study of this theme, it would be very useful to mash them. Or again, let's think about a freelancer or a simple person that wants to make a personal activity *smarter* using personal and free information that can bring from the internet.

Another important consideration for the industry scenario, that adds value to the mashup solution, born from the fact that the Information Technology scenario is having a deep evolution, under the unceasing pressure of Market, that every days shows new needs. This change is led by technology evolution process, which offers innovative business opportunities due to new discoveries.

The Software production sector for enterprises is certainly one of the most interested scenarios by this changing: next to the Enterprise Applications, developed by IT as solution to the largest part of an enterprise business problems, there is the need for Situational Applications, software built ad hoc to manage particular business processes linked to the different realities. Very often the resources destined to the production of these applications are limited, because of the lower relevance that they have in the global mission. The tendency is to adopt low quality software or to use

non conventional alternatives, using software built for other purposes to achieve own goals.

The main difficulty to invest in the production of software of this kind is in the "artistic" and "social" nature of the business processes to model, in the sense that their particularity and specificity do not allow their implementation in Enterprise Applications.

So, the challenge is to provide very flexible, agile and low cost methods and processes to develop Situational Applications, in order to exploit the business opportunity represented by the "Long Tail".

The possibilities offered by web 2.0 technologies are some of the most accredited solutions to this problem. In this scenario Mashups have a great relevance.

2 Mashup

The word **"mashup"** recurs in some different context, assuming each time a different meaning; for example:

- *In music*, a mashup is a song or composition created by blending two or more songs, usually by overlaying the vocal track of one song seamlessly over the music track of another.
- *In cinematography*, a video mashup is the combination of multiple sources of video, which usually have no relevance with each other, into a derivative work often lampooning its component sources, or another text.
- *In digital production*, a digital mashup is a digital media file containing any or all of text, graphics, audio, video and animation drawn from pre-existing sources, to create a new derivative work.

From these definitions, although the clear difference among the objects they define, is evident that the basis concept is the same: to mix and mash some objects to obtain a new, derivative and complex one. Mashups arises exactly from this idea.

One of the first example in this way was the work done by John Snow during the colera outbreak in London during 1854. He placed on a city map all the deaths and all the water pumps and analyzing the information built with this mashup, he understood that the water was the main vehicle of cholera.

Within *the computer science* the word mashup assumes another new meaning:

A mashup is a lightweight web application, which allows users to remix information and functions belonging to different sources and to work with them to build software in a completely new, simple and quick way. The user can efficiently model their own business process under the own vision of the problem, achieving a result so particular and specific that is impossible to obtain with the older technologies.

Mashups stands on the fundamental concept of data and services integration; to operate in this way there are three main primitives: *Combination, Aggregation and Visualization.*

- The first allows to collect data from heterogeneous sources and to use them within the same application;
- the second primitive allows to operate on collected data having a measure and building new information starting from them;

– the last is used to integrate data in a visual way using maps or other multimedia objects.

In a technological view of the Mashup and of its data and services integration problem, the natural representation of the problem itself can be obtained using a level/pyramidal approach.

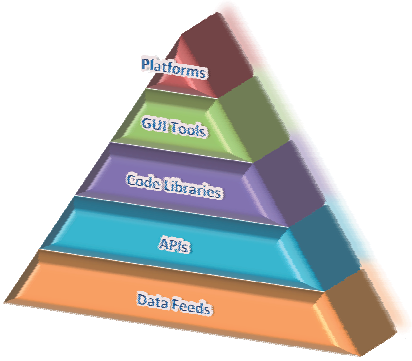


Fig. 1. The Mashup Pyramid

In the lowest abstraction layer there are *Data Feeds* and web technologies involved by them. They represent a good solution to access to updated data in a quick and secure way.

In the immediately superior level live the *APIs*, used to obtain data dynamically and on demand services.

A great level of abstraction is achieved by *Code Libraries*, that can be thought as application frameworks and API packages built to resolve some kinds of problems.

On the Code Library level stands the *Gui Tools* level, made of widgets and technologies related to the composition of small graphical applications to show data or to allow the access to a service.

On the top of the pyramid there's the *Platform level*, composed by all the tools and platforms that support mashup applications building, allowing to compose single graphical elements and lower level data.

3 Eclipse

The mashup pyramid shows well the complexity level of a mashup application, that sometimes cannot be clear, due to the common and diffused basic use of mashups in internet. To obtain the maximum results from mashup process, infact, there is the need to act on each of pyramid levels in the application building process in order to obtain a flexible and complete development process of a mashup application.

As direct consequence, there is the need of an integrated development environment, capable to adapt itself to each kind of need thanks to its modular and flexible architecture, allowing to face every aspect of the mashup problem and that drives the developer through all the production process till deployment and testing of the final application.

These requisites are well satisfied by the Open Source Development Platform "**Eclipse**" [1,2], that can greatly adapt itself to every scenario thanks to its modular architecture.

Eclipse in its last meaning *"an open, extensible platform for tool integration built by an open community of tool providers"* is perfect for the mashup aim, due to the multiple faces that it can show:

- A first usage that can be done is the simple use of Eclipse as IDE to build web 2.0 applications, using Web Tools Platform (WTP) [1] or PHP Development Tools [1] (PDT) to build specific mashup applications, following the traditional programming model, and then deploy them to an application server.
- A more interesting usage is to build mashup application using all possible feature of Eclipse architecture, leveraging Eclipse as Application and Runtime Framework, doesn't renouncing to the facility of using Eclipse as the development environment in which manage the entire production process.

Adopting the second option, great importance covers integration capability, that is one of the main directives of the Eclipse project from its birth: the platform architecture allows 5 different integration levels as represented in the following diagram [3].

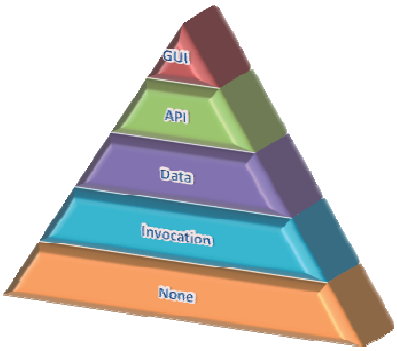


Fig. 2. The Eclipse Integration Pyramid

The lowest level is "*None – No Integration*" which represents the possibility to have no integration with other external tools if this integration is not needed.

The "*Invocation*" level represents the integration obtained by invocation of tools and services external to Eclipse within the platform itself. Services are executed as external processes distinct from the IDE one, using the same Eclipse resource manager to start them. Platform gives the possibility to manage a tool-resource association registry independent from the Operative System one.

"*Data*" level is certainly the one that offers the greatest level of integration. Eclipse platform, in fact, allows to collect data from heterogeneous sources, to give them a structure and to provide them to own applications, in a coherent and very flexible way.

The "*API*" integration level graft perfectly on Data level. The extreme flexibility of the Data level is balanced by the need of decode, understand and maintain integrity of

Data. Using APIs allow to access data in a coherent, secure and especially dynamic way, so the programmer can release the burden of dealing of the explicit manage of data. With APIs there is the introduction of the concept of service, intended as an on demand action on data. The modular structure of Eclipse allows each application to define its own APIs and services that become usable by the platform itself and by its components.

On the top of the pyramid there is GUI integration level, which allows many tools or application to share the platform Graphical User Interface becoming an unique application perfectly integrated in the IDE structure, starting from different applications.

4 Points of Convergence

The integration capabilities of Eclipse platform suggest to use Eclipse to build mashups [4,5]. This can be done if an association between the levels of the two pyramids will be find. There is a clear correspondence between the Mashup pyramid levels and the Eclipse Integration Pyramid ones:

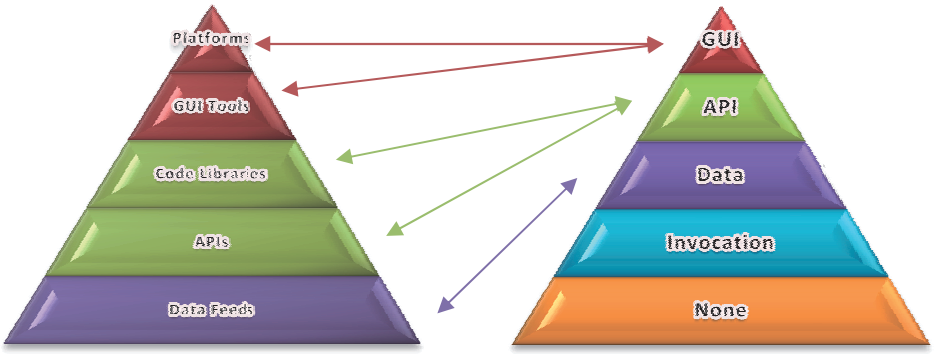


Fig. 3. Correspondence between Mashup-Eclipse pyramids

4.1 Data Level

The "Data" level of Eclipse Integration Pyramid allows to greatly manage Data Feeds, base of mashup pyramid, extending this possibility to all other structured Data belonging to other sources like heterogeneous Databases. This perspective appear very interesting in building enterprise mashups, in which often there is the convergence between data belonging to Enterprise Databases and data belonging to web services external to own enterprise infrastructure.

Eclipse Galileo offers many opportunities in this way, we focus our attention on "Data Tool Platform" (DTP) [6] project and the famous "Business Intelligence and Reporting Tool" (BIRT) [7].

4.1.1 Data Tool Platform

DTP [6] born form the need of having a good set of tools in a unique environment to develop or manage data-centric systems. So the project aim is very ambitious: to give a unique environment in which you can manage different kind of data sources.

This is achieved using the Open Data Access (ODA) framework, which realizes the connection with the most common data sources: XML, Web Services, CSV files and JDBC.

Actually the main developing efforts are in relational databases frameworks and tools, making DTP a very good environment to manage a large variety of Database Systems; SQL Development Tools component is one of the most powerful tool developed to this purpose. Interesting is the use of DTP made by “IBM pureQuery”, which furnishes an Eclipse environment where DTP features for accessing databases can be used in a very simple way by Java developers, during the application development activity, directly from the code.

In conclusion, DTP project seems a very good candidate to realize the correspondence between the two Data levels of the pyramids, but in the actual version it shows all its limits, due to a very poor support to ODA sources, very important for mashup applications. In our opinion, to begin to be very useful for mashups DTP must evolve to support ODA sources the same way it does with Databases.

4.1.2 Business Intelligence and Reporting Tool

BIRT (Business Intelligence and Reporting Tool) [7] is a top-project of Eclipse community [1]. Its main aim is to build reports doing business intelligence on web applications. We are not interested in BI capabilities, but the way a report is built is very interesting for mashups, in fact BIRT reports consist of four main parts: data, data transforms, business logic and presentation. Data and data transforms are perfect for our purposes.

As we can read on Birth Homepage [1]:

- *Data* - Databases, web services, Java objects all can supply data to your BIRT report. BIRT provides JDBC, XML, Web Services, and Flat File support, as well as support for using code to get at other sources of data. BIRT's use of the Open Data Access (ODA) framework allows anyone to build new UI and runtime support for any kind of tabular data. Further, a single report can include data from any number of data sources. BIRT also supplies a feature that allows disparate data sources to be combined using inner and outer joins.
- *Data Transforms* - Reports present data sorted, summarized, filtered and grouped to fit the user's needs. While databases can do some of this work, BIRT must do it for "simple" data sources such as flat files or Java objects. BIRT allows sophisticated operations such as grouping on sums, percentages of overall totals and more.
- *Business Logic* - Real-world data is seldom structured exactly as you'd like for a report. Many reports require business-specific logic to convert raw data into information useful for the user. If the logic is just for the report, you can script it using BIRT's JavaScript support. If your application already contains the logic, you can call into your existing Java code.
- *Presentation* - Once the data is ready, you have a wide range of options for presenting it to the user. Tables, charts, text and more. A single data set can appear in multiple ways, and a single report can present data from multiple data sets.

In conclusion, we can say that BIRT has very good data retrieving and transforming capabilities, which usage is very interesting in mashup building. Besides

the Business Logic and HTML Presentation capabilities gives to this project many possibilities to realize application logic and presentation of a some simple kinds of mashup. The limit is that the final product is not a real application, but only a report.

4.1.3 The importance of web services

One of the most interesting data and services source for mashups is represented by web services, because using them allows to link services belonging to Enterprise Service Oriented Architecture (SOA) and services belonging to an external Web Oriented Architecture (WOA). Actually service integration in Eclipse is managed by Data level through ODA drivers or by API level through specific plugins.

Another interesting possibility is the "Web Services" project that is a sub-project in the Eclipse Web Tools Platform Top-Level Project [8], composed by JST Web services component and WST Web services component. The first contains tools for developing and interacting with Java Web services, the latter contains tools for Web services development which is not Java specific.

Essentially WOA, that is a subset of SOA, describes a core set of Web protocols like HTTP and plain XML as the most dynamic, scalable, and interoperable Web service approach. The only real difference between traditional SOA and the concept of WOA is that WOA advocates REST, an increasingly popular, powerful, and simple method of leveraging HTTP as a Web service in its own right.

WOA architecture emphasizes generality of interfaces (UIs and APIs) to achieve global network effects through five fundamental generic interface constraints:

1. Identification of resources
2. Manipulation of resources through representations
3. Self-descriptive messages
4. Hypermedia as the engine of application state
5. Application neutrality

This generalization enable us to match easily WOA resources with Mashup Pyramid.

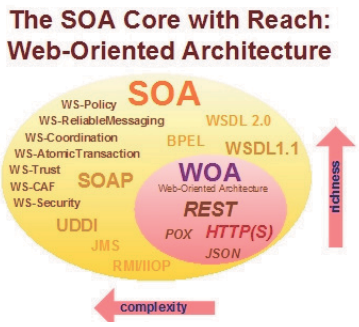


Fig. 4. SOA and WOA comparison architecture

4.2 API level

The "API" level allows to realize integration through platform API and Plugins that compose the particular installation. The modular structure of Eclipse makes easy to use external APIs or Code Libraries in a native manner or managing them through particular plugin. A famous example of the last possibility is offered by the "gdata-java-client-Eclipse-plugin" which, after installed, gives the opportunity to easily create Java application that uses the common Google APIs. These possibilities make the platform itself a natural candidate in realizing the right integration required from Mashup's "API" and "Code Libraries" levels.

4.3 GUI level

The “GUI” level is certainly one of the most powerful and tested integration level in Eclipse. The extreme simplicity that characterizes the extension of the development environment and its graphical personalization makes the platform adapt to realize any kind of application, beginning from different applications too, using perspectives, views and editors. So, Eclipse results to be the perfect environment in which integrate mashup application widgets directly in its architecture, with the whole flexibility, support and easiness of integration in the platform. Eclipse is a unique environment in which realize the development of the environment itself.

Plugin Development Environment (PDE) and Rich Client Platform (RCP) are two of the most famous projects that may help in this aim. The first allows to build components to add to Eclipse structure, the latter allows to leverage the structure of Eclipse to realize an RCP application.

4.4 Eclipse and Web 2.0

Last fundamental step is to bring the realized Eclipse mashup application on the web. Because of its genesis as stand-alone software development tool, sometimes are not clear the real possibilities of Eclipse in the web 2.0 field. There are many projects that allow the platform to be accessible and usable from the web using a common browser:

4.4.1 Eclifox

Among all these projects one of the most interesting is the “Eclifox” [9] plugin developed as IBM Alphawork; it makes available a remote Eclipse instance on the web through Jetty web server, transforming SWT based GUIs on XUL based GUIs. XUL is the famous language used by Mozilla products like Firefox.

4.4.2 Rich Ajax Platform

Another important perspective is brought by the project “Rich Ajax Platform” (RAP) [10]. This Project allows to design Ajax applications based on Eclipse in a simple way very similar to RCP Application building, substituting SWT widget library with RWT built for web. So RAP is a very good candidate to mashup application’s GUI building, because the entire application is transformed in a web 2.0 application, using the common Java technologies for server-side programming without the need for an Eclipse instance running on a server.

5 Eclipse and Jazz

Another great advantage in using Eclipse is the convergence in act between the Eclipse project and Jazz platform [11]: the introduction of Jazz candidates Eclipse as a complete tool which allows the collaborative development and the managing of the whole software life cycle. These innovations perfectly agree with mashup philosophy.

Jazz is an IBM initiative to help make software delivery teams more effective, Jazz transform software delivery making it more collaborative, productive and transparent.

The Jazz initiative is composed of three elements:

- An architecture for lifecycle integration
- A portfolio of products designed to put the team first

– A community of stakeholders.

5.1 An architecture for lifecycle integration

Jazz products embody an innovative approach to integration based on open, flexible services and Internet architecture. Unlike the monolithic, closed products of the past, Jazz is an open platform designed to support any industry participant who wants to improve the software lifecycle and break down walls between tools.

5.2 A portfolio of products designed to put the team first

The Jazz portfolio consists of a common platform and a set of tools that enable all of the members of the extended development team to collaborate more easily. The newest Jazz offerings are:

- Rational Team Concert is a collaborative work environment for developers, architects and project managers with work item, source control, build management, and iteration planning support. It supports any process and includes agile planning templates for Scrum and the Eclipse Way.
- Rational Quality Manager is a web-based test management environment for decision makers and quality professionals. It provides a customizable solution for test planning, workflow control, tracking and reporting capable of quantifying the impact of project decisions on business objectives.
- Rational Requirements Composer is a requirements definition solution that includes visual, easy-to-use elicitation and definition capabilities. Requirements Composer enables the capture and refinement of business needs into unambiguous requirements that drive improved quality, speed, and alignment.

Jazz is not only the traditional software development community of practitioners helping practitioners; it is also customers and community influencing the direction of products through direct, early, and continuous conversation. Fig.6 shows Db2 on campus project community monitored by using Jazz tools [5]. The project organization of the project was 130 students 4 thesis student about, and was stimulated by using team concert application. Jazz is also a process definition framework including agile and personalized processes.

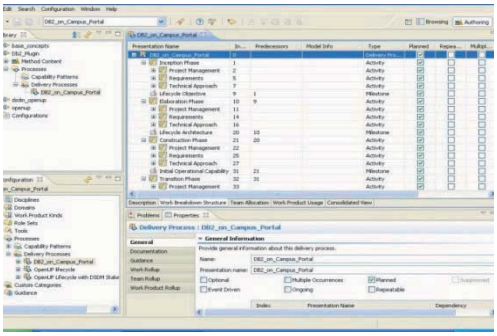


Fig. 5. Db2 on campus project – Jazz

The contribute that the adoption of Jazz can give to mashup development process is enormous, allowing to use mashups to realize a complex software system, thanks to its possibility of managing the lifecycle and the production team, in the same way it is done for a traditional software system.

6 CityInformation: a mashup example using BIRT

To underline the real possibilities of Eclipse in mashup developing, we developed CityInformation, a simple example on how Eclipse BIRT project can be used to realize a mix of data belonging to different data sources.

CityInformation shows to the user some information on an user chosen American City in the form of a BIRT HTML report.

When the application starts, it asks the user to insert the name of the city to display information (Fig. 7).

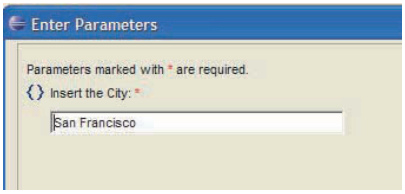


Fig. 6. Enter Parameters

Then the application invokes some free web services to retrieve some information on the city:

The webservice **WeatherForecast** [12] supplies weather forecast information for all the week and the geographic position of the city. Longitude and Latitude are used to display the city map by **Google Maps** [13] using a mashup with an external website. Under the map, forecast information are displayed grouped by day, showing an image and the expected temperatures.

The **Amazon** webservice [14] is used to obtain a list of the most sold Travel Guides of the City on Amazon.com; each book is displayed to the user with its own cover image. Fig.8 shows the report obtained requesting information on the city of San Francisco.

Fig. 7. Report on San Francisco

7 Conclusions and future development

In this paper we showed our belief in mashups as very good solution to many kind of problems and the need of an integrated environment in which exploit all the possibilities given by mashup philosophy. We believe that Eclipse platform is a great candidate for this purpose thanks to its modular and flexible architecture that allows to manage every abstraction level of the mashup pyramid [15,16].

As future development we aim at integrating first and second mashup pyramid with the corresponding two Eclipse levels. A common project is also growing grouping together Napoli and Salerno University with IBM and their business partner with the aims to research new mashup methodologies, technologies and best practices [17,18]. This collaboration is a great opportunity to integrate knowledge belonging to these different realities, mashing together open-source solutions, university's resources and technologies from enterprises development environments, and to have the possibility to prove that Eclipse and mashups can be the base to build solutions to many problems of modern enterprises and organizations.

References

1. Eclipse foundation http://Eclipse.org
2. Eclipse italian community at DIS of University Federico II of Naples http://Eclipse.dis.unina.it
3. Jim Amsden *Levels of Integration: five ways you can integrate with Eclipse platform* http://www.Eclipse.org/articles/Article-Levels-Of-Integration/levels-of-integration.html
4. Maresca P. (2009) *La comunità Eclipse italiana e la ricerca nel web3.0: ruolo, esperienze e primi risultati* Slide show for Mashup meeting at University of Salerno.
5. Maresca P. (2008) *Projects and goals for the Eclipse italian community* in Proceedings of Fourteenth International Conference on Distributed Multimedia Systems (DMS2008), Boston , USA, September 4 - 6, pp.112-117.
6. Eclipse Data Tooling Platform web site http://www.Eclipse.org/datatools/
7. Eclipse Business Intelligence and Reporting Tool web site http://www.Eclipse.org/birt/
8. Web Tools Platform web site http://www.Eclipse.org/webtools/
9. Eclifox web site http://www.alphaworks.ibm.com/tech/eclifox
10. Eclipse Rich Ajax Platform web site http://www.Eclipse.org/rap/
11. Jazz http://jazz.net
12. Weather Forecast webservice: http://www.webservicex.net/WeatherForecast.asmx?WSDL
13. Google Maps API: http://code.google.com/intl/it-IT/apis/maps/
14. Amazon webservices: http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl
15. P.Maresca, G.M.Scarfogliero, L. Stanganelli (2009) *Eclipse: a new way to Mashup*, in Proceedings of DMS2009, San Francisco Bay, USA, September 10-12, to be published.
16. L.Colazzo,A.Molinari, P. Maresca, L. Stanganelli (2009) *Mashup learning and learning communities*, in Proceedings of DMS2009, San Francisco Bay, USA, September 10-12, to be published.
17. IBM developerWorks Mashup section http://www.ibm.com/developerworks/spaces/mashups
18. Duane Merril (2006) *Mashups: The new breed of Web app* http://www.ibm.com/developerworks/web/library/x-mashups.html?S_TACT=105AGX01&S_CMP=LP

A Design Pattern Detection Plugin for Eclipse

Christian Tosi, Marco Zanoni, and Stefano Maggioni

Università degli Studi di Milano-Bicocca, DISCo - Dipartimento di Informatica, Sistemistica e Comunicazione, 20126 - Milan, Italy
{christian.tosi,marco.zanoni,maggioni}@disco.unimib.it

Abstract. It is well known that software maintenance and evolution are expensive activities, both in terms of invested time and money. Reverse engineering activities support the obtainment of abstractions and views from a target system that should help the engineers to maintain, evolve and eventually re-engineer it. An important task pursued by reverse engineering is design pattern detection, whose main objective is the identification of the design pattern instances that have been used in the implementation of a system, that let the practitioners focus on the overall architecture of the system without minding at the programming details it has been implemented with.
In this context we propose an Eclipse plug-in called MARPLE (Metrics and Architecture Reconstruction Plug-in for Eclipse), which supports the detection of design patterns through the use of *basic elements* that are mechanically extracted from source code. The development of this platform is mainly based on the exploitation of the Eclipse framework, relying also on two plugins: Java development tools (JDT) for code analysis and the Graphical Editing Framework (GEF) for results visualization.

Key words: Reverse Engineering, Design Pattern Detection, Static Analysis and Eclipse Plugin

1 Introduction

A software engineering research area that is getting more and more importance for the maintenance and evolution of software systems is reverse engineering [5, 16]. A relevant objective of this discipline is to obtain representations of the system at a higher level of abstraction and to identify the fundamental components of the analyzed system by retrieving its constituent structures. Getting this information should greatly simplify the restructuring and maintenance activities, as we obtain more understandable views of the system and the system can be seen as a set of coordinated components, rather than as a unique monolithic block. Considering these components, particular relevance is given to design patterns [10].

Hence the design patterns detection (DPD) activity is particularly relevant in the context of reverse engineering and program comprehension.

The main objective of DPD is to gain a better comprehension of a software system and on what kind of problems have been addressed during the development of the system itself. The presence of DP can be considered as an indicator

of good software design, as design patterns are reusable for their self definition. Moreover, they are very important during the re-documentation process, in particular when the documentation is very poor, incomplete or not up-to-date.

Different tools for DPD have been proposed in the literature (e.g. [12, 8, 18, 22, 25]). They usually have problems in finding all the design patterns of the GoF catalogue [10], some tools recognize only a small subset of these patterns, but the main problem is that the found results contain many false positive DP instances and moreover they usually don't scale well when trying to analyze medium/large systems.

The aim of this paper is to describe a project, on which we are currently working, named MARPLE (Metrics and Architecture Reconstruction PLug-in for Eclipse). MARPLE's architecture has been designed in order to be language independent, even if until now we have performed our analysis on Java systems.

Our approach to design pattern detection is based on the detection of design pattern subcomponents ([2, 1]), which can be considered indicators of the presence of patterns. We use static source code analysis: the Abstract Syntax Trees (ASTs) of the analyzed projects are parsed in order to obtain the structures we need for our elaboration, which we called *basic elements* (BE).

MARPLE is conceived as an Eclipse plug-in. This choice was supported by two main reasons: first of all, Eclipse is the most used open source development framework, it is supported by a wide community of developers, and it is strongly based on the concept of extending its functionalities through the implementation of plug-ins. The second reason resides in the fact that this platform encourages the strong interaction among the various components that constitute the framework; therefore, the implementation of our plug-in is based on the exploitation of the functionalities of other plug-ins and modules of the Eclipse framework. This reuse of components improves and speed up the development process.

The paper is organized as follows: Section 2 briefly introduces some tools for design pattern detection and other for software architecture reconstruction; Section 3 introduces the overall MARPLE architecture and goes into more details about the technologies, plug-ins and libraries used during its development; Section 4 presents some results about DPD; Section 6 gathers the conclusions and outlines possible future works.

2 Related works

Probably the only thing all research groups involved in reverse engineering agree on, is that manual inspection is not feasible or very expensive and tool support is necessary.

For example, the manual detection of DPs and reconstruction of the architecture of a system without the aid of automatic procedures require a preliminary study of the whole system, which can take, for large systems, a lot of time.

The time required to understand the system, to a sufficient extent and to become confident with it, is certainly not linear with its size, and might be so long that when the task is accomplished, the knowledge gained is already

outdated. This is due to the relations within the parts, which increase in number and complexity as long as the size of the system increases; other causes are emergency repairs and wild maintenance, which have made things even worse, by introducing additional coupling between the system's components.

In this section we give an overview on the main DPD tools and approaches.

2.1 Design pattern detection

SPOOL [14] stands for Spreading Desirable Properties into the Design of Object-Oriented, Large-Scale Software Systems. The authors outline three possible ways of detection: manual, automatic, and semi-automatic, the first two of which are supported in SPOOL. Automatic recovery is implemented through queries to a previously generated repository.

The Pat system [21] transforms C++ source code into PROLOG facts and matches them against pattern definitions given as PROLOG statements. The approach is based on first-order logic and constraint solving techniques. The authors claim that this system is able to detect many pattern instances without missing any, and with few false positives. Although we cannot verify the truth of these assertions, we can easily imagine the high computational costs of this approach. In addition, only header files are examined, so no behaviour is available for them.

PTIDEJ tool [12] uses constraint solving with explanation. Explanation consists in first detecting instances matching DP definitions exactly, and then, by relaxing some constraints, entities that are less and less similar to DPs.

The MAISA tool [20] uses a library of patterns defined as sets of variables, representing the patterns roles, and unary or binary constraints over them. A solution of the constraint satisfaction problem is a possible instantiation of those variables. To be able to detect instances which do not correspond exactly to the definition, it is possible to relax it by removing some constraints, but the number of candidates tends to increase quickly. A similar approach has been used in the Columbus tool [4], in which patterns are defined by using an XML based language called Design Pattern Markup Language (DPML) and searched for in an Abstract Semantic Graph (ASG) generated by the tool itself.

Web of Patterns [6] uses an approach to the formal definition of design patterns based on the web ontology language (OWL). This prototype accesses the pattern definitions and detects patterns in Java software. The tool connects to a pattern server, downloads and scans the patterns, translates them into constraints, and resolves these constraints with respect to the program to be analysed.

Pinot [23] is a modification of Jikes [13], IBMs Java compiler, developed to detect various design patterns based on static rule-based analysis. The authors present an interesting, but arguable, reverse engineering oriented reclassification of the GoF design patterns into different categories: patterns provided by the programming language, syntax-based patterns, semantic-based patterns, domain-specific patterns and patterns that aren't but mere generic concepts.

Fujaba [17] uses fuzzy logic combined with Fujaba Abstract Syntax Graphs to cope with two different types of pattern variations, design variants and implementation variants. They address the former with ASGs, by modeling various design variants with different graphs, and then handle implementation variants by defining a set of fuzzy rules together giving the degree of belief that a pattern is found at a certain location in the program.

SPQR [25] uses Elemental Design Patterns (EDPs) and a system for logical calculus, the ρ-calculus, to detect DPs. The authors claims that the tool can detect design patterns, but at present no evidence is provided and it is not possible to verify the results.

3 An overview on MARPLE

An overview of the overall architecture of MARPLE is depicted in Figure 1 that shows the general process of data extraction, of design pattern detection and of results visualization.

The information that is used by MARPLE is obtained by an Abstract Syntax Tree (AST) representation of the analyzed system through the JDT plugins.

DPD receives the set of *basic elements* that have been found inside the system, collected in an XML file. By *basic elements* we mean a set that is formed by Elemental Design Patterns (EDPs) [24], design pattern clues [15] and micro patterns [11], that are intended as the basic information we exploit as hints for the presence of design patterns inside the code, and as basic relationships that may connect two or more classes in terms of object creation, method invocation or inheritance.

The architecture is constituted by five main modules, that interact with one another through XML data transfers. The four modules are the following:

- The *Information Detector Engine* which collects both *basic elements* and metrics starting from an AST representation of the source code of the analyzed project;
- The *Joiner*, that extracts architectures from the project that could match those of design patterns, basing on the information extracted by the *Information Detector Engine*;
- The *Classifier*, which tries to infer whether the architectures detected by the *Joiner* could effectively be realizations of design patterns or not. This module helps to detect possible false positives identified by the *Joiner* and to evaluate the similarity with the canonical design patterns by assigning different confidence values;
- The activity of *Results visualization* provides an organic view of the project analysis results. Through this activity, the user will see the results produced by the detection of design patterns .

As we have outlined, the MARPLE project leans on the Eclipse framework and hence many functions did not need to be rewritten, but have been implemented by extending the core concepts provided by the platform. Obviously,

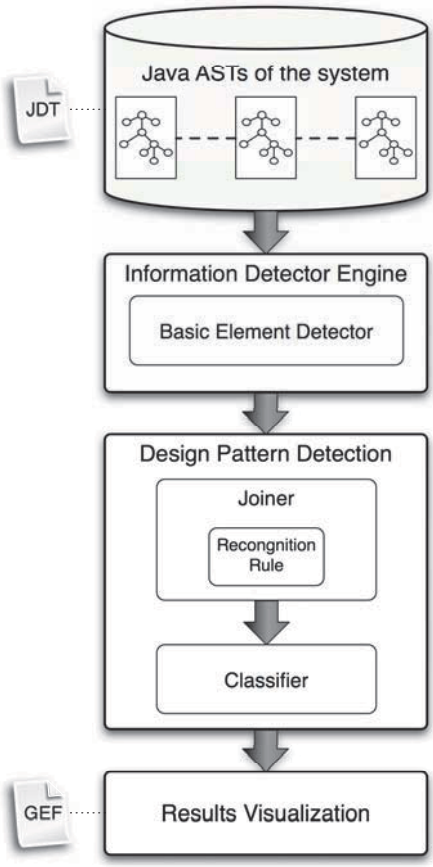


Fig. 1. The architecture of MARPLE.

many functionalities have also been implemented by using third party libraries, like XML data access or graph representations. All the modules work on XML files that come from some previous modules. Each of these modules works on these files, both for reading and writing, with the Apache XMLBeans library [9].

In the following, we will discuss the components we exploited in the implementation of each module constituting MARPLE, discussing the reasons about the choices we made and the effectiveness of these components in pursuing our objectives.

3.1 The Information Detector Engine Module

Currently, the *Basic Elements Detector* (BED) sub-module has been completely developed and tested. The *basic elements* are extracted by visitors that parse an AST representation of the source code, each of them returning instances of the

basic elements if the analyzed classes or interfaces actually implement them. The information is acquired statically and is characterized by 100% rate of precision and recall. This value is due to the fact that these kinds of structures are meant to be *mechanically recognizable*, i.e. there is always a 1-to-1 correspondence between a basic element and a piece of code. In other words, the *basic elements* are not ambiguous (as on the contrary design patterns may be), and once a basic element has been specified in terms of the source code details that are used to implement it, the *basic element* can be detected without any problem.

We didn't implement an AST structure from scratch, but we used the *JDT* library that provides all the classes and interfaces that can be used to access a project's ASTs. Moreover, it provides the class *ASTVisitor*, that is used to visit the nodes constituting the AST according to the Visitor design pattern [10]. Therefore, one visitor for each *basic element* has been extended from *ASTVisitor*. These visitors are invoked sequentially on the ASTs of the classes constituting the project and visit only those nodes that may contain the information they are able to detect (for example, the visitors that look for *method call* EDPs only analyze nodes that represent a method invocation, i.e. instances of the *MethodInvocation* class).

The results coming from the visitors, i.e. the instances of *basic elements* that have been found inside the project, are then stored in an XML file.

The BED module has been developed also for the .NET environment [7].

3.2 The Joiner Module

As far as the *Joiner* module is concerned, no particular third party technologies have been used. Nonetheless, this module does not handle the system through its AST representation, but it manages it as a graph *Attributed Relational Graph* (ARG) , where the set of vertices corresponds to the set of types (i.e. classes and interfaces) the project is constituted by, while the edges are the set of *basic elements* that connect the types with one another. In fact, each *basic element* can be seen as a relationship between a type and another one (therefore depicted as an edge between two nodes of the graph), or as a relationship between a class and itself (depicted as a self loop on a graph node). The system graph representation is directly derived from the output generated by the *Information Detector Engine*. As we have briefly anticipated at the beginning of this section, this module tries to extract architectures that match a target structure, defined in terms of *Joiner rules*. A *Joiner rule* is a graph that collects roles and *basic elements* (edges) that must be present among the roles in order to satisfy the rule. In particular, we have to define rules to extract candidate design pattern instances. In this way the roles in the rule are the roles of the target design pattern. For example, if we want to extract the couple of roles $(R1, R2)$, where $R1$ has a *create object* and a *delegate method call* to $R2$, we may represent this rule as shown in Figure 2.

The *Joiner* module tries to match and extract this kind of architectures from the graph representing the system through an ad-hoc graph matching algorithm. The algorithm has been demonstrated to have linear complexity in the number

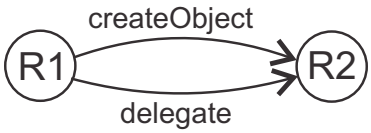


Fig. 2. An example of Joiner rule.

of the classes of the system. All the details of the algorithm and the complexity demonstration can be found in [27].

The extracted architectures are then inspected by the *Classifier* module which tries to infer whether they can represent instances of design patterns or not.

3.3 The Classifier Module

The *Joiner* output is the input for the next analysis step performed by the *Classifier* module. This module takes all the candidate design pattern instances and tries to evaluate their grade of similarity to the searched design pattern in order to be able to rank the instances given as output. Figure 3 shows an example of the classification process. Through our current approach, we generate every possible valid mapping $\{(R1, C1), (R2, C2), \ldots, (Rn, Cn)\}$ for each pattern instance, where each Ci is the class that is supposed to play the role Ri inside the pattern. These mappings are all of fixed size (an element for each pattern role) and each class has a fixed number of features, where the features are the *basic element* retrieved in the class. In this way each mapping can be represented as a vector of features whose length is given by $(num\_features * num\_roles)$. These vectors are grouped by a clustering algorithm, producing k clusters; each pattern instance is represented as a k-long vector, having in each position i the absence/presence of the i-th mapping. Since we know that an instance is a DP or not directly from the training set, we can enqueue to each vector the class attribute and use the resulting dataset for the training of a supervised classifier.

In the *Classification* Module we used the clustering and classification algorithms provided in Weka [19]. All the detail of classification process can be found in [26].

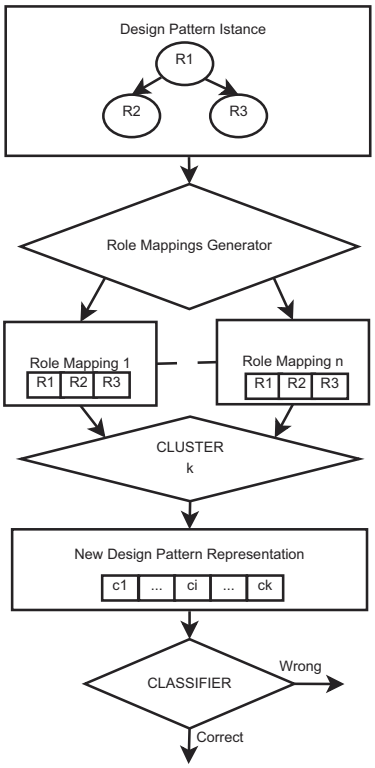


Fig. 3. Classification process.

3.4 Results visualization

This module, using the GEF framework, depicts the found patterns in order to allow users to better understand them. It allows to visualize the results in two way:

Joiner model: this visualization (as depicted in Figure 4) organizes the patterns following their definition;

Graph: this visualization (as depicted in Figure 5), organizes the pattern as a graph where the nodes are the classes involved in the pattern, the colors of the nodes are the roles in the pattern and finally the edges are the basic elements connecting two classes.

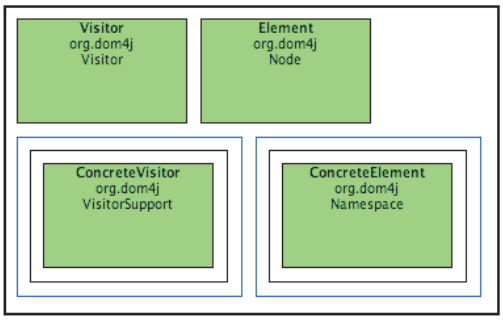


Fig. 4. Joiner model visualization

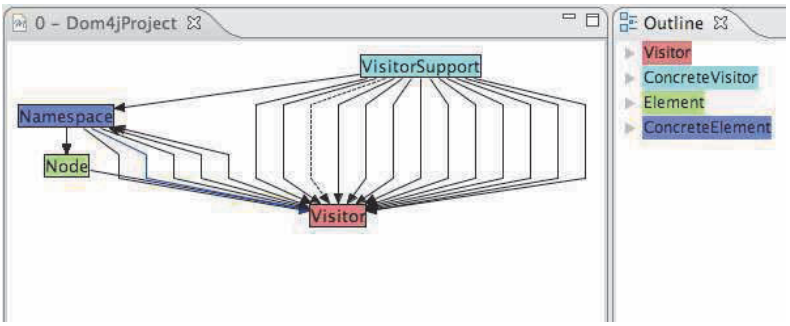


Fig. 5. Graph visualization

4 Conclusions and future works

In this paper we have presented MARPLE, a tool for design pattern detection that is being developed as an Eclipse plug-in. We are very interested in using such capabilities also in the context of system modernization and in particular for what concerns systems migration to SOA. In this context, we have also explored if detecting design patterns in a system can give useful information towards the migration of the system [3].

Currently, some modules have been completely implemented and we are currently working on others.

The *Information Detector Engine* has been completely developed as far as the *Basic Elements Detector* is concerned. It detects all of the elemental design patterns, clues and micro pattern we think are useful as hints for design pattern detection.

As far as the *Joiner* is concerned, we have defined rules for the extraction of DP instances for the Abstract Factory, the Builder, the Factory Method, the Prototype, the Singleton, the Adapter (both based on classes and on objects), the Composite and the Proxy design patterns. Rules for the remaining patterns are under development.

We have developed the *Classifier* module, which proved us that we can successfully extract information from our representation of the problem, as the performance values are higher than the apriori ones. We are currently analysing if, adding or removing some *basic elements*, we could improve the performances.

Future works are related to complete the detection of all the Design Patterns of [10], to add new views based on both metrics and *basic elements* and to better integrate all the modules: we started the implementation of MARPLE through the development of separated modules, but now we need them to cooperate in order to enhance the user experience and to let the tool to be more effective. Moreover we would like to complete the benchmark for the comparison of design pattern detection tools.

Since the design of MARPLE architecture has been done in order to be language independent, future works will consider other languages as for example C++.

References

1. F. Arcelli, S. Masiero, C. Raibulet, and F. Tisato. A comparison of reverse engineering tools based on design pattern decomposition. In *ASWEC '05: Proceedings of the 2005 Australian conference on Software Engineering*, pages 262–269, Washington, DC, USA, 2005. IEEE Computer Society.
2. F. Arcelli and C. Raibulet. The role of design pattern decomposition in reverse engineering tools. In *Pre-Proceedings of the IEEE International Workshop on Software Technology and Engineering Practice (STEP 2005)*, pages 230–233, Budapest, Hungary, 2005.
3. F. Arcelli, C. Tosi, and M. Zanoni. Can design pattern detection be useful for legacy systemmigration towards soa? In *SDSOA '08: IEEE Proceedings of the 2nd international ICSE Workshop on Systems development in SOA environments*, pages 63–68, New York, NY, USA, 2008. ACM.
4. Z. Balanyi and R. Ferenc. Mining design patterns from c++ source code. In *ICSM '03: Proceedings of the International Conference on Software Maintenance*, page 305, Washington, DC, USA, 2003. IEEE Computer Society.
5. E. J. Chikofsky and J. H. C. II. Reverse engineering and design recovery: A taxonomy. *IEEE Software*, 7(1):13–17, 1990.
6. J. Dietrich and C. Elgar. Towards a web of patterns. *Web Semant*, 5(2):108–116, 2007.
7. D. F.Arcelli, L.Cristina. nmarple: .net reverse engineering with marple. In *Proceedings of WOOR 2007, ECOOP Workhop*, Berlin, 2007.
8. R. Ferenc, F. Magyar, A. Beszedes, A. Kiss, and M. Tarkiainen. Columbus - reverse engineering tool and schema for c++. In *ICSM '02: Proceedings of the International Conference on Software Maintenance (ICSM'02)*, page 172, Washington, DC, USA, 2002. IEEE Computer Society.
9. T. A. S. Foundation. Xmlbeans. Web site. `http://xmlbeans.apache.org/`.
10. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.
11. J. Y. Gil and I. Maman. Micro patterns in java code. *SIGPLAN Not.*, 40(10):97–116, 2005.

12. Y.-G. Guéhéneuc. Ptidej: Promoting patterns with patterns. In *Proceedings of the 1st ECOOP Workshop on Building a System using Patterns*. Springer Verlag, 2005.
13. IBM. Jikes. Web site. `http://jikes.sourceforge.net`.
14. R. K. Keller, R. Schauer, S. Robitaille, and P. Pagé. Pattern-based reverse-engineering of design components. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 226–235, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.
15. S. Maggioni. Design pattern clues for creational design patterns. In *Proceedings of the DPD4RE Workshop, co-located event with IEEE WCRE 2006 Conference*, Benevento, Italy, 2006.
16. H. A. Müller, J. H. Jahnke, D. B. Smith, M.-A. Storey, S. R. Tilley, and K. Wong. Reverse engineering: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 47–60, New York, NY, USA, 2000. ACM.
17. U. Nickel, J.Niere, J. Wadsack, and A. Zundorf. Roundtrip engineering with fujaba. In *Proc of 2nd Workshop on Software Reengineering (WSR)*, Germany, 2000.
18. J. Niere, W. Schäfer, J. P. Wadsack, L. Wendehals, and J. Welsh. Towards pattern-based design recovery. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 338–348, New York, NY, USA, 2002. ACM.
19. U. of Waikato. Weka. Web site. `http://www.cs.waikato.ac.nz/ml/weka/`.
20. J. Paakki, A. Karhinen, J. Gustafsson, L. Nenonen, and A. I. Verkamo. Software metrics by architectural pattern mining. In *Proceedings of the International Conference on Software: Theory and Practice (16th IFIP World Computer Congress)*, pages 325–332, Beijing, China, 2000.
21. L. Prechelt and C. Kramer. Functionality versus practicality: Employing existing tools for recovering structural design patterns. *Universal Computer Science*, 4(12):866–883, December 1998.
22. N. Shi and R. A. Olsson. Reverse engineering of design patterns for high performance computing. In *Proceedings of the 2005 Workshop on Patterns in High Performance Computing*, 2005.
23. N. Shi and R. A. Olsson. Reverse engineering of design patterns from java source code. In *ASE '06: Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering*, pages 123–134, Washington, DC, USA, 2006. IEEE Computer Society.
24. J. Smith. An elemental design pattern catalog. Technical Report 02-040, Dept. of Computer Science, Univ. of North Carolina - Chapel Hill, December 2002.
25. J. M. Smith and P. D. Stotts. Spqr: Flexible automated design pattern extraction from source code. In *ASE '03: Proceedings of the 18st IEEE/ACM International Conference on Automated Software Engineering*, pages 215–224, 2003.
26. C. Tosi. Marple: classification techniques applied to design pattern detection. Master's thesis, Università degli studi di Milano-Bicocca, 2008.
27. M. Zanoni. Marple: discovering structured groups of classes for design pattern detection. Master's thesis, Università degli studi di Milano-Bicocca, 2008.

XTGT: un tool estensibile per la generazione di test suite

Laura Bottanelli

Università degli studi di Bergamo, Facoltà di Ingegneria

Sommario Il plugin XTGT è stato sviluppato per aiutare il tester durante la fase di testing del software, trovando una suite di test composta dal minor numero possibile di casi test, permettendo di ridurre il tempo e le risorse utilizzate per il test. XTGT offre l'opportunità al tester di scegliere la copertura combinatoriale e il model checker da utilizzare per ricavare i casi di test. Per il progetto è stata creata una piattaforma estensibile sfruttando la piattaforma di Eclipse.

1 Introduzione

La fase di test del software ricopre una parte molto importante nel processo di sviluppo del software, e, spesso, richiede molto tempo e molte risorse per la sua esecuzione. Il Model Based Testing è un metodo di testing del software molto utilizzato, nel quale i casi di test derivano, in tutto o in parte, da un modello che descrive gli aspetti più significativi del sistema sottoposto a test (System Under Test - SUT).

Il plugin è stato sviluppato per aiutare chi deve effettuare il testing del software con il Model Based Testing, creando la test suite con il minior numero di casi di test da eseguire, risparmiando così tempo e risorse. Il tester non deve far altro che scrivere il modello del software (specifiche), scegliere il tipo di copertura e il model checker, uno strumento in grado di risolvere problemi di logica, utilizzato in XTGT per ricavare i casi di test necessari. Il risultato sarà una suite di casi di test ridotta, da utilizzare per eseguire il testing del software.

2 Plugin XTGT

Il problema descritto nel paragrafo precedente è stato risolto creando una piattaforma estensibile, utilizzando la piattaforma di Eclipse, che permette di introdurre diversi tipi di copertura e diversi model checker. Il tester potrà scegliere la copertura e il model checker più adatto alle sue esigenze.

XTGT si basa sull'idea di ATGT [1], uno strumento per la generazione automatica di sequenze di test a partire da un modello del sistema.

Il plugin sfrutta le potenzialità degli Extension Points, per aggiungere ulteriori funzionalità al plugin. In particolare, nel progetto di tesi, sono stati aggiunti due model checker: Yices ed Hysat, in grado di aumentare la velocità di

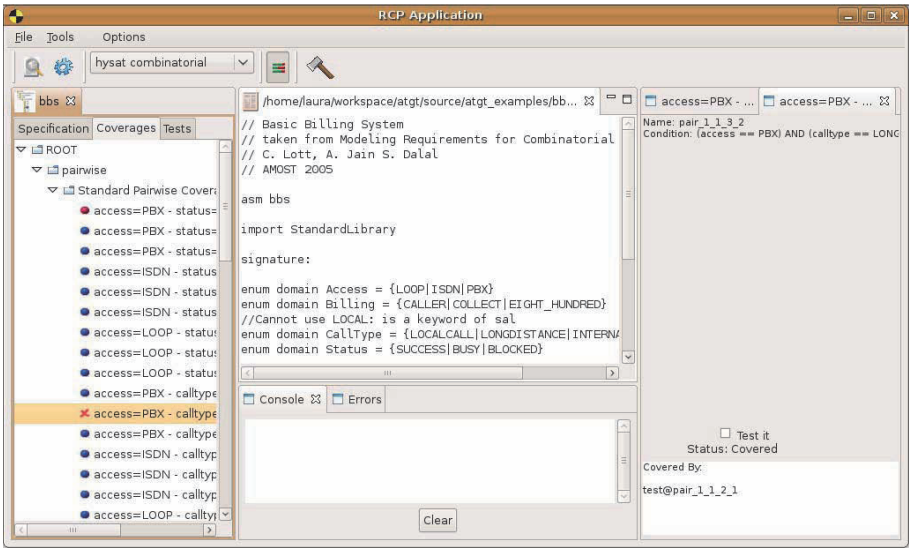


Figura 1. Esempio di utilizzo del plugin XTGT.

risoluzione del problema (Yices) [2], e di aumentare la potenzialità di descrizione del software, consentendo di utilizzare anche delle specifiche non lineari (Hysat).

In figura 1 è illustrato un esempio di utilizzo di XTGT. Nella finestra centrale di XTGT è presente il modello del software, in quella di sinistra il tipo di copertura scelto e le variabili coperte, mentre nella finestra di destra si possono notare i dettagli del test che si vuole coprire (test predicate).

Riferimenti bibliografici

1. Andrea Calvagna and Angelo Gargantini. *A Logic-Based Approach to Combinatorial Testing with Constraints.* In Bernhard Beckert and Reiner Hähnle, editors, *Tests and Proofs, Second International Conference, TAP 2008, Prato, Italy, April 9-11, 2008. Proceedings*, volume 4966 of *Lecture Notes in Computer Science*, pages 66–83. Springer, 2008.
2. Andrea Calvagna and Angelo Gargantini. *Combining Satisfiability Solving and Heuristics to Constrained Combinatorial Interaction Testing.* In *Tests and Proofs*, volume 5668 of *Lecture Notes in Computer Science*, pages 27–42. Springer, 2009.

Awareness in un plug-in per Shared Editing

Annunziato Fierro, Ilaria Manno, Pasquale Vitale

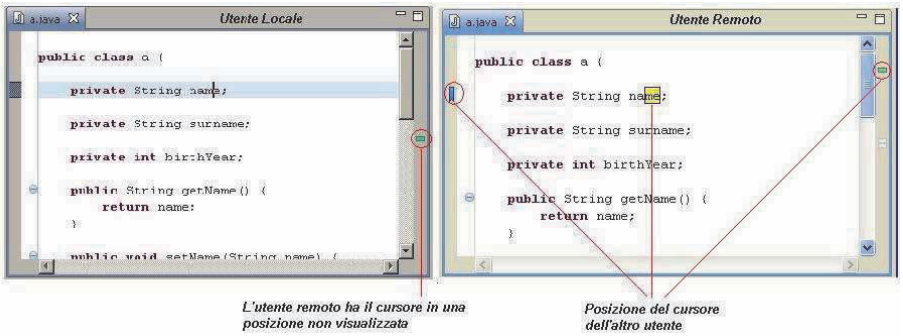
ISISLab, Dip. Informatica ed Applicazioni,
University of Salerno, Italy
{annunziato.fierro@email.it, manno@dia.unisa.it, paco86v@gmail.com}

L'editing collaborativo è una attività sincrona e distribuita che consente ad un gruppo di utenti di modificare uno stesso documento contemporaneamente e ottenere in real time le modifiche apportate dagli altri utenti. Le informazioni circa la posizione in cui sta scrivendo un utente, dove si trova il suo cursore o quale porzione di testo sta selezionando rappresentano informazioni di *consapevolezza* del processo di shared editing. In questa demo viene presentato il lavoro svolto per aggiungere informazioni di consapevolezza al un plug-in Real Time Shared Editor (RT Shared Editor) [5] sviluppato nell'ambito del progetto Eclipse Communication Framework (ECF) [3] della comunità di Eclipse.

RT Shared Editor offre funzioni di editing collaborativo per due utenti: essi saranno in grado di lavorare contemporaneamente su un unico file sorgente visualizzando le modifiche apportate dall'altro utente in tempo reale. RT Shared Editor affronta il problema della consistenza del contentuto del file tramite una implementazione dell'algoritmo Cola [4]. Oltre alla visualizzazione delle modifiche apportate, RT Shared Editor non offre informazioni di consapevolezza circa la posizione all'interno del file in cui stava lavorando l'altro utente.

Il lavoro che stiamo presentando aggiunge a RT Shared Editor tali informazioni, consentendo agli utenti di sapere non solo le modifiche apportate dall'altro utente, ma in generale su quale porzione di testo si trova il cursore o la selezione dell'altro utente. Tali informazioni di consapevolezza sono state aggiunte allo Shared Editor utilizzando i vertical ruler della componente grafica dell'editor e modificando le caratteristiche di visualizzaione del testo per evidenziare il testo su cui sta operando l'altro utente. I vertical ruler sono i 'righelli' laterali che vengono visualizzati a destra e sinistra degli editor di eclipse; su di essi è possibile applicare dei marker; un marker è rappresentato da una icona informativa e può essere posizionato lungo un vertical ruler in corrispondenza di una posizione del file. Il vertical ruler a sinistra contiene i marker associati alle posizioni del file che sono visibili all'interno dell'area dell'editor (visione locale), mentre il vertical ruler a destra contiene i marker associati alle posizioni dell'intero file (visione globale), anche quelle non attualmente visibili nell'area dell'editor. I vertical ruler e i marker sono stati utilizzati in modo da indicare la posizione dove sta lavorando l'altro utente: lungo i vertical ruler viene inserito dinamicamente un marker in corrispondenza della riga in cui si trova il cursore o la selezione dell'altro utente.

Oltre alle informazioni visualizzate tramite i marker sui vertical ruler, vengono fornite informazioni più dettagliate sulla posizione di ciascun utente all'interno del file grazie alla possibilità di modificare la visualizzazione del testo: viene



modificato il colore di sfondo del testo selezionato dall'altro utente (awareness della selezione) o dei caratteri adiacenti al cursore (awareness del cursore).

RT Shared Editor è stato progettato per avere esattamente due utenti che collaborano. Uno sviluppo interessante riguarda la possibilità di avere più utenti che partecipano all'editing. Questa modifica è stata implementata portando il plug-in con le funzioni di awareness all'interno di CoFFEE [6–8]. CoFFEE è un *Collabortive Face to Face Educational Environment* basato su Eclipse, utilizzato in classe per supportare l'apprendimento collaborativo tramite computer. Le applicazioni principali sono il Session Player ed il Session Client; sviluppate come Rich Client Application, esse consentono di integrare nel sistema tool collaborativi. Poiché CoFFFE prevede la collaborazione di n utenti, abbiamo implementato una versione del plug-in sviluppato a partire da RTShared Editor come tool di CoFFEE. In questa versione abbiamo modificato l'editor in modo che oltre ai due utenti che possono scrivere, fosse possibile avere n utenti in grado di leggere ed effettuare selezione del testo. Non è stato possibile modificare il numero di utenti che possono scrivere (due) poiché questo è un limite di Cola, l'algoritmo che gestisce la consistenza del file. Il passaggio a n utenti 'lettori' consente a tutto il gruppo di utenti che partecipa alla sessione collaborativa di vedere le modifiche apportate al file. Questa versione del plug-in è stata modificata in modo da ottenere la consapevolezza circa la posizione di tutti gli utenti 'lettori', oltre che degli 'scrittori'.

References

1. Annunziato Fierro: Editor Cooperativi in CoFFEE. Master Thesis, 2009.
2. Pasquale Vitale: Editor Cooperativi in Eclipse con la consapevolezza delle operazioni. Master Thesis, 2009.
3. Eclipse Communication Framework: `www.eclipse.org/ecf`
4. Cola: `http://wiki.eclipse.org/RT_Shared_Editing#The_Cola_Source`
5. Real Time Shared Editor: `http://wiki.eclipse.org/RT_Shared_editing`
6. De Chiara, Di Matteo, Manno, Scarano: CoFFEE: Cooperative Face2Face Educational Environment. In Proc. of the 3rd Int. Conf. on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2007), New York, 2007.
7. Manno, Belgiorno, De Chiara, Di Matteo, Erra, Malandrino, Palmieri, Pirozzi, Scarano: Collaborative Face2Face Educational Environment (CoFFEE). In Proc. of 1st Int. Conf. on Eclipse Tecnhologies (Eclipse-IT 2007), 2007.
8. CoFFEE: `http://coffee-soft.org`

Interactive Graphical Maps for Infocenter via Model to Model Transformation

Enrico Oliva

ALMA MATER STUDIORUM–Università di Bologna, Cesena, Italy

Abstract. In this work we discuss an Eclipse based Model to Model (M2M) transformation to generate interactive graphical maps related to Darwin Information Typing Architecture (DITA) and delivered by the Eclipse *Infocenter*. The maps are shown and manage with a extension of Prefuse run-time interpreter that accepts in input instance of GraphXML model. The obtained system is called **GIMI** (Graphical Interactive Map into Infocenter). M2M transformation is realized by mapping rules among the DITA Schema and GraphXML expressed in Xtend, a model transformation language provided by the *openArchitectureWare* (oAW) framework. The M2M approach allows us to reach a clear separation between structures and interpreters and a more maintainable generation of code. Orthogonally, the realization of functionalities specific to domain-related user actions is leaved to extensible run-time interpreters.

1 GIMI Overview

The goal of our project is exploit the Darwin Information Typing Architecture (DITA) [1] to design an build interactive graphical map to be delivered by the Eclipse Help System *Infocenter* - the Internet-based scenario used by IBM to support its online Manual Management System (MMS). The resulting navigation system is called **GIMI** (Graphical Interactive Map into Infocenter).

Graphical maps are used to give visual supports for understanding and representing knowledge. In fact, different map styles can be exploited to emphasize different aspects of the information given by the *eContents* and to allow users to perceive concepts in different ways. In our project, the maps are show and managed by a run-time interpreter which is an extension of the Prefuse toolkit [2], that accepts in input models instances of GraphXML and provides interactive functionalities such as zooming, filtering and detailing-on-demand.

To provide the input of the graphical maps we realize a model to model transformation [3] that statically links domain parts (DITA) to presentation parts (maps in GraphXML) by running mapping rules defined among both meta models. The rules are expressed in xTend a model transformation language provided by the *openArchitectureWare* (oAW) framework.

The logical architecture of the generation system is shown in the picture 1. The content model (written with the help of a DITA editor) is given as input to the transformation process together with the transformer specification M2MSpec expressed in Xtend. The model (instance of the NavMap model) obtained as result of the transformation becomes the input of the runtime rendering library (Prefuse).

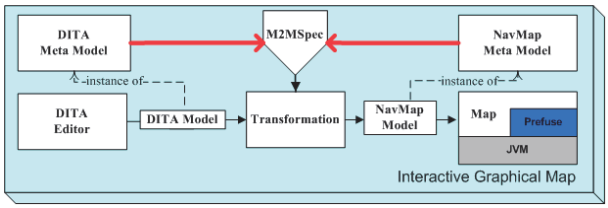


Fig. 1. Model to model transformation process.

Our final intent is to extend the Infocenter visual interface in order to allow learners to use a map as a new form of navigation inside the contens as manual. To this end we introduce in the Infocenter the graphical map provided by a Java Applet included in a new frame (`NavmapFrame`) to be shown at the client site. The task of the new frame is to load the applet with the right configuration file generated from the model transformation. The result is depicted in figure 2.

In order to improve the use of the map also as the input device to add annotation and to organize navigation into customizable *reading paths*. We have defined a Domain Specific Language (DSL) [4] by introducing a new formal model for content organization [5] from which to start the map generation process.

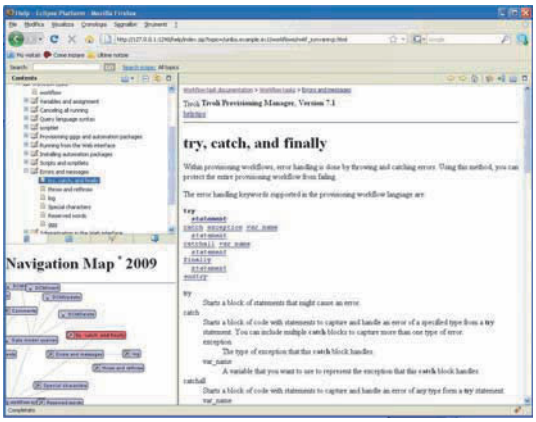


Fig. 2. GIMI resulting user interface.

1.1 Model to Model Transformation

The use of model to model transformation has several architectural and development advantages. First of all it introduces a clear separation point in the overall architecture between structures and interpreters with the advantage of system modularization. Moreover, in case of refactoring, this transformation is more maintainable being directly

connected with both meta model (source and target). This type of solution promotes a strong reuse of code, by inserting maintainable transformation.

The transformation system is based on a set of Eclipse plugins, by exploiting the openArchitectureWare (oAW) framework [6], which is a modular MDA/MDD generator suite implemented in Java. oAW has strong support for EMF-based models [7] but can work with other models, too (e.g. UML2, XML, XSD or simple JavaBeans) through the specific mapping with the Ecore independent model.

Mapping rules are expressed in oAW Xtend, which a general purpose transformation language that enables to build functions over meta model elements, by means of a compact syntax and powerful supports for model management. Xtend has special operators to work on collection; the major drawback is that it is not compliant with the standard QVT and has no support for debugging.

1.2 Conclusions

The M2M approach was successful because it was available the Eclipse and oAW framework that provide the right abstraction to easily implement Model Driven Software Development techniques as the use of static-mapping. The final attended result was reached, the GIMI system improves Infocenter navigability by showing interactive graphical map. A working example of the map is available on the web at the link: lia.deis.unibo.it/ Staff/Enrico.oliva/WebMap/Map_wrapper.htm.

Acknowledgement

The authors are grateful for technical assistance and collaboration received from the IBM Tivoli Lab in Rome.

References

1. Harrison, N.: The Darwin Information Typing Architecture (DITA): Applications for globalization. Professional Communication Conference IPCC **10-13** (2005) 115 – 121
2. Heer, J., Card, S.K., Landay, J.A.: Prefuse: a toolkit for interactive information visualization. In: CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems, New York, NY, USA, ACM (2005) 421–430
3. Brown, A.W., Iyengar, S., Johnston, S.: A rational approach to model-driven development. IBM Systems Journal **45**(3) (2006) 463–480
4. Natali, A., Oliva, E., Bonanni, C.: Model-driven generation of graphical maps for e-contents. In Lanubile, F., ed.: Eclipse-IT 2008. 3rd Italian Workshop op on Eclipse Technologies. (nov 2008) 48–57
5. Natali, A., Del Cinque, A., Oliva, E.: Using eclipse in building model-driven e-learning supports. In Maresca, P., ed.: Eclipse: a Great Opportunity for Industry and Universities in Italy. Volume 1., Cuzzolin Editore (October 2007) 27–42 1st International Conference on Eclipse Technologies (Eclipse - I 2007), Napoli, Italy.
6. itemis AG: openarchitectureware 4.3. http://www.openarchitectureware.org/
7. Budisnsky, F., Steinberg, D., Merks, E., Ellersick, R., Grose, T.: Eclipse Modeling Framework. (2004)

Eclipse-L: Un ambiente integrato open source per la didattica universitaria mobile

Mauro ROCCO
Università di Napoli "Federico II" - mauro.rocco@gmail.com

1. Introduzione

Gli studenti, che escono dai licei, arrivano ai corsi di base universitari con una scarsa conoscenza degli elementi della logica e dei concetti di analisi e sintesi, ma soprattutto con una scarsa abitudine a cooperare costruttivamente e ad organizzare il proprio lavoro soprattutto se innestato in un progetto.
Con queste premesse, si è sviluppato un ambiente integrato di strumenti che possono essere utilizzati per le attività didattiche (ma non solo) dei corsi di primo livello universitario. Gli strumenti sono tutti open source ed ospitati su una architettura più generale ed estendibile quale quella di Eclipse che incoraggia e favorisce la costruzione di soluzioni innovative.

2. Struttura del sistema Eclipse-L

Il sistema Eclipse-L (Eclipse Learning) nasce nel 2008 composto da tre sottosistemi:

1. **LCE** (Learning and cooperative environment),
   Esso offre a tutti gli studenti una valida piattaforma per la cooperazione durante lo svolgimento di corsi didattici, degli homeworks e per effettuare esercitazioni e attività di laboratorio, il tutto in un unico ambiente continuamente migliorato dai preziosi suggerimenti degli studenti e continuamente aggiornato.
2. **IT-SL** (Italian community with second life project),
   Permette a tutti gli studenti di beneficiare dei privilegi offerti dal sottosistema "LCE" tramite l'interattività con piattaforme esterne all'architettura Eclipse-L. E' stato realizzato un canale di comunicazione con "Secondo Life" [14], ambiente virtuale sempre più conosciuto ed usato dai giovani.
3. **L&E** (Lab & exams).
   Realizzato per focalizzare l'attività di natura didattica finalizzata all'acquisizione dei CFU, consentendo agli studenti di condurre attività di laboratorio anche da remoto su multi-piattaforma e multi-dispositivo.

Nel 2009 la piattaforma è stata migliorata ed arricchita con l'aggiunta di:

4. **IDS** (Interactive Didactical support),
   Pensato per offrire un collegamento diretto tra gli studenti e il docente. Questo strumento può essere utilizzato sia dal docente, per organizzare

una video lezione (in diretta o in differita), sia dagli studenti per richiedere al docente spiegazioni e/o chiarimenti avvalendosi di una pratica lavagna virtuale, di videocamera e microfono, nonché di canale per la condivisione di file.

5. **LS** (Logic simulator),
   Realizzato per l'attività di natura didattica finalizzata all'acquisizione dei CFU, offrendo agli studenti uno strumento di simulazione grafica per i circuiti logici combinatori.
6. **S3** (System support for student)
   Per offrire agli studenti una supporto/guida alle ormai tante funzionalità implementate nell'architettura Eclipse-L, con la vantaggiosa possibilità cui ogni studente ha nel poter inserire, aggiornare e correggere le informazioni pubblicate, garantendo il servizio sempre aggiornato e utile.

L'importanza del progetto consiste nell'offrire allo studente la possibilità di usare gli strumenti messi a disposizione in modo completamente facile ed efficace, privo di ogni limitazione dettata dalla piattaforma e/o dal dispositivo.
Lo studente non è più legato alla piattaforma usata per accedere a tali servizi, al tipo di collegamento utilizzato, al luogo in cui si trova, né al dispositivo che ha a disposizione.
Cambia la metodologia di insegnamento: non più incentrata attorno al docente (cui prima era l'unico ente per la divulgazione delle informazioni e nozioni), ma sempre più a contorno della figura dello studente, che può strutturare l'offerta formativa con estrema facilità e come più ritiene opportuno.
Nasce la figura del **Mobile-Student**: uno studente libero dai vincoli geografici ed economici, utilizzatore di applicativi software di tipo "cloud-software".
Questi vantaggi, in ambito universitario, sono visti con maggior interesse dallo studente che può muoversi liberamente tra le varie strutture del campus, usare i servizi didattici nel tempo a disposizione e quindi lontano dai vincoli stringenti dovuti allo studio delle numerose materie che ci possono essere all'interno di un qualunque corso di laurea.

3. Eclipse IDS – Videocomunicazione per lezione e supporto

La piattaforma di comunicazione aggiunta al sistema Eclipse-L offre, sia agli studente quanto ai docenti, potenti strumenti di comunicazione. "**IDS**" è un modulo di interazione tra Moodle e OpenMeetings.
L'uso di "Red5", un server video per Flash e su Openlaszlo, applet Java per realizzare applicazioni Flash a partire dal codice Laszlo, ha permesso la creazione di stanze virtuali in cui è possibile organizzare videoconferenze, con l'ausilio di webcam, microfoni e di una lavagna virtuale "Whiteboard" su cui è possibile proiettare delle presentazioni (file), oppure usare la lavagna virtuale come proiettore del desktop del docente.

4. Eclipse L&S – Eclipse Lab & Exams

L'interazione tra Eclipse e Moodle avviene per mezzo del plug-in appositamente creato e che quindi permette, in un ambito didattico, di offrire servizi utili alla formazione e alla verifica dell' utente verso un particolare tipo di corso.
Il plug-in fornisce una serie di strumenti necessari all' interazione tra utente e piattaforma Eclipse, la quale potrà essere utilizzata con un qualunque tipo di linguaggio di programmazione, di sviluppo o di modellazione (java, python, php, UML, XML, etc..).

5. Eclipse LS – Eclipse Logic Simulator

LS è un simulatore di circuiti logici per uso didattico. Questa applicazione è fruibile agli studenti per mezzo di un framework (GMF & EMF) di base che traduce ogni oggetto iconografico in pratiche stringhe XML.
E' prevista una futura possibilità di affiancare all'applicazione un Server Web AJAX, per rendere l'applicativo fruibile tramite rete, nonché l'introduzione del "concetto tempo" per gestire anche le condizioni temporali e dunque permettere allo studente di poter simulare anche circuiti logici sequenziali.

6. Eclipse S^3 – System Support for Student

La piattaforma **S3** (System Support for Students) è rivolta allo studente bisognoso di informazioni, guide e dimostrazioni. Realizzata con l'uso di un motore MediaWiki, completamente gratuito e open-source, è sempre più utilizzata da tutti gli studenti che hanno espresso meno indugi durante le fasi d'esame nonché un maggiore interesse verso il mondo accademico.

7. Conclusioni

In questo lavoro si è mostrato l'integrazione fra tecnologie allo scopo di costruire un ambiente multipiattaforma e multi dispositivo nonché un valido supporto informativo utile per il mobile-student.
L'ambiente Eclipse-L è oggi utilizzato sperimentalmente presso i laboratori del D.I.S della facoltà d ingegneria dell'ateneo Federico II di Napoli.
Il progetto è stato realizzato dal punto di vista e necessità dello studente che affronta per la prima volta l'ambiente didattico universitario, offrendo un supporto sempre attivo, pronto, efficace e conveniente.
L'uso dell'intera piattaforma attraverso un qualunque Browser Web (come Firefox) garantisce un elevato livello di compatibilità per gli utilizzatori, nonché svincola lo studente dai macchinosi metodi di studio, rendendolo sempre più libero e dinamico.

EifFE-L® incontra Eclipse

D. Brondo, L. Stanganelli
DIST - Dipartimento di Informatica Sistemistica e Telematica – University of Genoa
{diego.brondo, lidia.stanganelli}@unige.it

Abstract. In questo lavoro viene descritto il progetto di integrazione della piattaforma open source per la formazione in rete EifFE-L (Environment for Freedom in E-Learning) all'interno della piattaforma open source Eclipse, al fine di creare un ambiente di sviluppo aperto per l'e-learning e il knowledge management modulare e facilmente estendibile.

Keywords. e-learning environment, eclipse environment, cooperative environment, knowledge management, knowledge sharing.

1 Introduzione

In questo lavoro viene presentato il progetto di integrazione dell'ambiente open source EifFE-L (Environment for Freedom in E-Learning) [www.eiffe-l.org] all'interno della piattaforma open source Eclipse [eclipse.org], al fine di creare un ambiente di sviluppo aperto per la formazione e la gestione della conoscenza, cercando in questo modo di rispondere alle crescenti esigenze di formazione e formazione continua. EifFE-L è un LCMS (Learning Content Management System) realizzato all'interno del Progetto CampusOne, promosso dalla CRUI, ed è una applicazione open source costruita in linguaggio Free Pascal [1,2]. La possibilità di usufruire della piattaforma Eclipse, costituisce il vero e proprio motore di innovazioni in quanto consente di realizzare applicazioni estensibili ed indipendenti dalla piattaforma hardware e software utilizzata. Il progetto si propone di sviluppare un ambiente integrato di strumenti che possono essere utilizzati per le attività didattiche (ma non solo) di corsi universitari e per il long-life learning. Il nuovo strumento si avvale dell'infrastruttura tecnologica offerta da EifFE-L per la gestione della conoscenza la profilatura e gestione degli utenti e le protezioni sugli accessi e sfrutta il framework estendibile di Eclipse che incoraggia e favorisce la costruzione di soluzioni innovative. Un altro aspetto significativo del progetto di integrazione di EifFE-L all'interno dell'ambiente Eclipse (che citiamo da questo punto in poi come EMEL: *EifFE-L meets Eclipse*) riguarda la creazione di ambienti simulativi da utilizzare all'interno di percorsi e-learning.

2 L'architettura di sistema

La comunità Eclipse [3] ha mostrato un forte interesse per l'e-learning sviluppando, diversi plugin inerenti a questa tecnologia. In questo lavoro si vuole descrivere la realizzazione di un macro-sistema contenente diversi strumenti per la didattica e, nell'ottica di offrire un servizio sempre più completo all'utente (discente), si è pensato di integrare un sistema già esistente e funzionante quale EifFE-L all'interno di Eclipse Gavab (che è una particolare versione di Eclipse). Quindi è stato realizzato un wrapper, ovvero, l'intero sistema EifFE-L è stato integrato all'interno della piattaforma Eclipse. In figura 1 viene rappresentata tale integrazione.

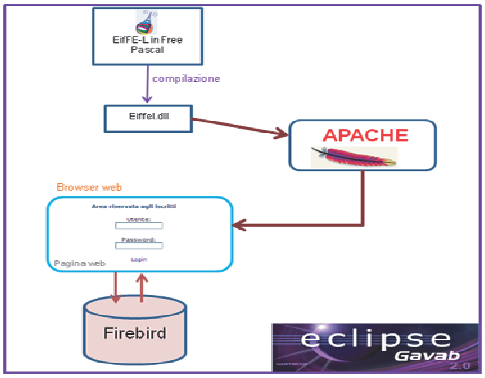


Fig.1 **Integrazione di EifFE-L in Eclipse Gavab**

Si è partiti dalla versione di EifFE-L in Free Pascal funzionante e si è proceduto alla sua integrazione all'interno di Eclipse Gavab, da qui vengono compilati i sorgenti ottenendo il file .dll che verrà caricato in Apache per l'effettiva esecuzione. La caratteristica fondamentale del wrapping è quella di sfruttare al massimo tutte le proprietà del vecchio sistema in modo semplice e fornire una nuova interfaccia per la gestione e lo sviluppo del sistema. In questo modo si potranno aggiungere nuove funzionalità o semplicemente rendere disponibile la vecchia applicazione nel nuovo ambiente. Eclipse Gavab è una distribuzione di Eclipse, multipiattaforma, realizzata dal gruppo di ricerca GAVAB del Dipartimento di Informatica dell'Università Rey Juan Carlos di Madrid. EifFE-L, la cui architettura è illustrata in figura 2 [1,2], ha la funzione di un LCMS (Learning Content Management System) con funzionalità di VLE (Virtual Learning Environment) classico ed è uno dei nuclei della formazione tradizionale attualmente utilizzato presso l'Ateneo di Genova. Il lavoro è stato sviluppato nell'ambito di un progetto comune fra il DIS dell'Università di Napoli Federico II, il Laboratorio di E-Learning & Knowledge management del DIST dell'Università di Genova [www.elkm.unige.it], e la comunità italiana di Eclipse, Eclipse italian community [eclipse.dis.unina.it]. Di fatto il livello di integrazione raggiunto è giusto al di sotto della API. L'applicazione viene evocata dalla piattaforma Eclipse connettendosi al database remoto esistente, Firebird.

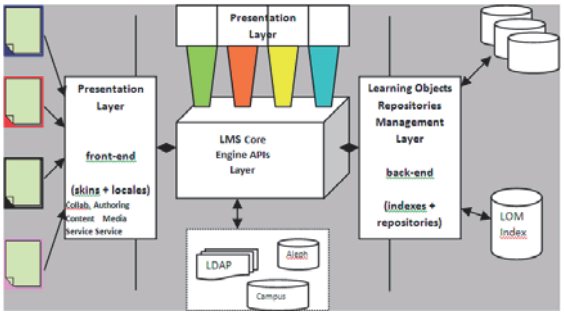


Fig. 2 Architettura del sistema EifFE-L

4 Conclusioni

L'integrazione di EifFE-L all'interno di Eclipse Gavab offre diversi vantaggi legati alla virtualizzazione attraverso un browser dell'intera piattaforma di e-learning che ne garantisce espandibilità, facilità di accesso, di uso e di installazione. Inoltre, attraverso il progetto EifFE-L incontra Eclipse, la comunità internazionale open source del Free Pascal "incontra" la comunità open source internazionale Eclipse.

L'esperienza è stata anche un'utile sperimentazione delle possibilità dell'architettura Eclipse e della sua comunità di pratica sul versante del riuso sia di applicazioni legacy che di database open source. Lo sviluppo si intravede in uno scenario di servizi web dove anche l'utente, non necessariamente uno studente nella accezione classica del termine, fruisce di un servizio personalizzabile sia nel processo di formazione che nelle risorse delle quali può fruire. L'uso di tali applicazioni sono svariate: dall'uso nella formazione universitaria all'uso nella formazione degli studenti re-immessi nei percorsi formativi dopo esserne usciti, dalla costituzione di classi eterogenee con obiettivi didattici personalizzabili, infine nella formazione di comunità di pratica aperte.

Bibliografia

1. Adorni G., Premuda G. , The e-learning environment of the University of Genoa: An open source SCORM and W3C compliant platform, Didamatica 2004
2. Adorni G., Sugliano A.M , Buone pratiche per l'E-Learning all'Università: Insegnamenti blended e corsi on line, CRUI, Roma, 2005, pp.191-226.
3. Maresca P., "Projects and goals for the eclipse italian community", in Proceedings of Fourteenth International Conference on Distributed Multimedia Systems (DMS2008), Boston , USA, September 4 - 6, 2008, pp.112-117.

Model Driven Software Development con Eclipse, StatechartUMC *

Aldi Sulova

Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo - CNR"
Via G. Moruzzi 1, 56124 Pisa, Italy
aldi.sulova@isti.cnr.it

Sommario StatechartUMC è un editor grafico che semplifica la generazione del codice testuale associato ad un'UMC statechart. Tale codice rappresenta l'input per il model checker UMC [1]. StatechartUMC applica la tecnica MDSD (Model Driven Software Development) ed il lavoro per la sua costruzione si suddivide in varie fasi: definizione del modello di dominio, costruzione di un'editor grafico per creare istanze del modello, trasformazioni da modelli predefiniti standard (UML) e scrittura di generatori di codice. Tutto questo è realizzato mediante tools open source basati sulla piattaforma Eclipse [2]: Eclipse RCP [3], Eclipse Modeling Framework (EMF) [4], Graphical Modeling Framework (GMF) [5] e Open Architecture Ware (oAW4) [6].

Key words: MDSD, Eclipse, EMF, GMF, oAW4, RCP

1 Introduzione

L'obiettivo principale di StatechartUMC è di semplificare la creazione dell'istanza di input per il model checker UMC. Senza approfondire le conoscenze su UMC, possiamo dire che un'istanza di input viene espressa come un modello a nodi ed archi basato su statecharts UML. A questo punto si può pensare ad uno strumento che modella graficamente le istanze e a partire da queste genera il codice testuale eseguibile da UMC. L'idea di base è quella di definire un modello attraverso diagrammi dai quali poter eseguire operazioni di verifica e validazione e naturalmente ottenere completamente o parzialmente il codice. Sicuramente creare un diagramma è più semplice che scrivere il corrispondente codice testuale. L'applicazione dell'approccio MDSD allo strumento risulta essere la scelta più ragionevole. L'MDSD è un tentativo di portare il processo di sviluppo software verso un livello più alto eliminando quasi completamente la scrittura del codice e inserendo un concetto nuovo di "sorgente": il modello.
Per realizzare gli obiettivi sono utilizzati tre frameworks open source della piattaforma Eclipse: EMF, GMF ed oAW4.
La funzionalità principale di EMF è di ricevere in input il modello di dominio (Domain Model, il meta-modello) e di fornire come output una serie di classi

* Parte del Progetto EU Sensoria IST-2005-016004 e del progetto PRIN 2007 D-ASAP

Java completamente implementate, che realizzano i vincoli, le relazioni e le associazioni descritte nel modello di partenza. Il primo passo nel processo di sviluppo dell'applicazione è la definizione del meta-modello, quindi l'identificazione delle entità principali del dominio che naturalmente, parlando di statecharts, saranno gli stati e le transizioni. Per modellare il dominio, EMF mette a disposizione un'editor grafico con una notazione molto simile all'UML.
Il passo successivo la connessione tra EMF e GMF. Lo scopo di GMF è facilitare lo sviluppo di istanze grafiche del meta-modello, creando editor grafici dotati di funzionalità quali drag & drop, copia/incolla, undo e redo: una classica applicazione GMF, ad esempio, è un editor che consente di disegnare diagrammi di vario tipo, con la possibilità di collegare tra loro le figure, ridimensionarle e spostarle. In sostanza con GMF le entità identificate nella prima fase hanno anche una rappresentazione grafica, dove le transizioni connettono tra loro gli stati. Un diagramma con stati e transizioni definisce un istanza di input per UMC.
Dalla rappresentazione grafica si può arrivare al codice testuale utilizzando oAW4. oAW4 [6] è un framework che definisce varie funzionalità per i modelli generati a partire da un meta-modello EMF. Sostanzialmente fornisce 3 linguaggi testuali che sono utili in diversi contesti: Check (.chk), per la verifica della correttezza del modello, Xpand (.xpt), per controllare l'output del processo di generazione ed Xtend (.ext), per definire librerie con operazioni generali, utilizzabili da Check ed Xpand. Xtend viene utilizzato anche in un contesto di trasformazione di modelli. La verifica della correttezza risulta essere un attività molto utile nel processo di definizione del modello. In una situazione normale, un'istanza di input, può avere un numero elevato di entità grafiche, quindi è molto importante mantenere una certa coerenza con il modello di dominio.

2 Funzionalità

Attualmente il tool fornisce tre funzionalità:

- validazione dei modelli, verifica della correttezza rispetto alla definizione del meta-modello,
- generazione del codice testuale dell'istanza di input per UMC,
- importazione e trasformazione di modelli da UML a UMC, nel paragrafo precedente abbiamo detto che il meta-modello UMC è molto simile al meta-modello UML per le macchine a stati. In questo caso altri tool UML (MagicDraw) si possono utilizzare per definire il modello UMC.

Riferimenti bibliografici

1. Franco Mazzanti. *Designing UML Models with UMC (ref. UMC V3.6 build p-April 2009)*, http://fmt.isti.cnr.it/umc/V3.6/umc.html.
2. Eclipse Project. http://www.eclipse.org/
3. EclipseRich Client Platform. http://wiki.eclipse.org/index.php/Rich_Client_Platform
4. Eclipse Modeling Framework. http://www.eclipse.org/modeling/emf/
5. Graphical Modelling Framework. http://www.eclipse.org/modeling/gmf/
6. Open Architecture Ware project. http://www.openarchitectureware.org/

Author Index