Digital Business Ecosystem

Contract n° 507953

Workpackage 26: DBE Studio/DBE Portal

Deliverable D26.5:

Final Release of the DBE Studio

Information Society Technologies

Project funded by the European Community under the “Information Society Technology” Programme

Contract Number: 507953

Project Acronym: DBE

Title: Digital Business Ecosystem

Deliverable N°: 26.5

Due dates: 12/2006

Delivery Date: 12/2006

Short Description:

The DBE Studio is an Integrated Development Environment (IDE) for the Digital Business Ecosystem (DBE). It includes Eclipse plugins that allow business services to be analysed, and corresponding software services to be defined, developed and deployed. The DBE Studio Integration task requires the ongoing managing of component integration and release builds, along with core plugin development.

This document outlines these ongoing tasks, and complements the actual software and websites which also form part of this deliverable. A developer's integration guide is reproduced in an appendix.

Author: Intel Ireland Ltd.

Partners contributed: Intel Ireland Ltd.

Made available to: Public

Versioning

Version	Date	Author, Organisation
D26.3 (0.1)	04/02/2006	David McKitterick, Intel Ireland
D26.3 (0.2)	22/02/2006	David McKitterick, Intel Ireland
D26.3 (0.3)	06/03/2006	David McKitterick, Intel Ireland
D26.3 (0.4)	15/03/2006	David McKitterick, Intel Ireland
D26.4 (0.1)	17/07/2006	David McKitterick, Intel Ireland
D26.5 (0.1)	16/11/2006	David McKitterick, Intel Ireland
D26.5 (0.2)	11/12/2006	David McKitterick, Intel Ireland

Quality check:

1st Internal Reviewer : Pierfranco Ferronato, Soluta.Net

2nd Internal Reviewer: Fotis G. Kazasis, TUC

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 2.5 License. To view a copy of this license, visit : http://creativecommons.org/licenses/by-nc-sa/2.5/ or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

creative commons

COMMONS DEED

Attribution-NonCommercial-ShareAlike 2.5

You are free:

- to copy, distribute, display, and perform the work
- to make derivative works

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor.

Noncommercial. You may not use this work for commercial purposes.

Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

TABLE OF CONTENTS

Illustration Index

Executive Summary

This document describes the final deliverable for task C32, DBE Studio Integration. This deliverable outlines the ongoing tasks of managing component integration and release builds for the DBE Studio SourceForge project [1]. The development of the DBE Studio is a collaborative effort by many DBE partners, which requires effective coordination of this integrated process. Although the work under this task is ongoing and recurring in nature, some significant achievements have been accomplished.

Chapter 2, 'DBE Studio on SourceForge.net', describes the efforts involved in setting up the DBE Studio project within the SourceForge domain. In addition to the features provided by SourceForge, such as bug trackers, mailing lists and a file release system, other contributions from this task to this project are outlined, including providing user documentation on a project home page and wiki site. Chapter 3 and 4, 'Component Integration' and 'Build and Release Management', outline the work achieved in the management and coordination of component integration and release builds. Chapter 5, 'Core Plugin Development', describes the set of Eclipse plugins that were developed as part of this task and which provide some core functionality to the DBE Studio distribution. User support and community involvement are discussed in Chapter 6. To conclude this deliverable, an integration guide for DBE Studio developers, which was created as part of the component integration effort, is presented in the appendix.

1. Introduction

The DBE Studio is an Integrated Development Environment (IDE) for the Digital Business Ecosystem (DBE). The DBE is a semantically-aware, service oriented architecture for Small and Medium sized Enterprises (SMEs). The architecture of the DBE is divided into three main logical components: Service Factory, Execution Environment (ExE) and Evolutionary Environment (EvE). The Service Factory contains a set of tools and core services which enable the creation and modeling of business and software services. The ExE is a peer-to-peer runtime environment where services can be deployed and executed by users and other services. The EvE uses advanced algorithms to help optimise the results of DBE queries by intelligently migrating service representations. It also optimises composed service workflows autonomously.

The DBE Studio is the implementation of the Service Factory. It is built on top of the Eclipse platform [6] where each tool is implemented as an Eclipse plugin. These tools allow business services to be analysed and modeled based on BML (Business Modeling Language) [9], and is where the corresponding software services can be defined, developed and deployed using a set of editors and wizards provided by the aggregation of plugins. The Eclipse platform provides an extensible pluggable architecture which ideally suits the requirements of an evolving development environment. The DBE Studio requires additional Eclipse frameworks (e.g. graphical and modeling frameworks) to the standard Eclipse distribution and it is intended that users will make use of the standard Java development tools also available with Eclipse.

The target users of the DBE Studio are advanced end users and software developers. These may vary from business analysts, who need to model their business and software services using the business modeling tools, to SME software developers, who need to expose existing legacy systems as a software service using the platform independent development and deployment tools.

The development of the DBE Studio is a collaborative effort by many DBE partners. This requires effective coordination of this integrated process along with the additional interconnections with other DBE environments such as the DBE Servent. The DBE Studio Integration task requires the ongoing management of component integration and release builds along with core plugin development. This task was created to provide a means to efficiently establish and coordinate this open source project within the SourceForge domain.

This document complements the software deliverable for the DBE Studio Integration task.

Although the work under this task is ongoing and recurring in nature, some significant achievements have been accomplished. These include the initial creation of the DBE Studio SourceForge project, extensive coordination of component integration and release builds, setting up of a build system and source repository, development of some core plugins and the ongoing delivery of help and support to developers and users of the DBE Studio.

2. DBE Studio on SourceForge.net

The DBE Studio project [1] was first hosted the SourceForge.net website [2] in July 2005. SourceForge is an open source software development web site, and with over 100,000 projects and over 1 million registered users, is the largest of its kind. SourceForge provides online resources for managing open source projects, including bug trackers, mailing lists, forums, and a file release system. As the largest open source code repository, SourceForge also hosts the largest amount of open source development products and applications available on the Internet.

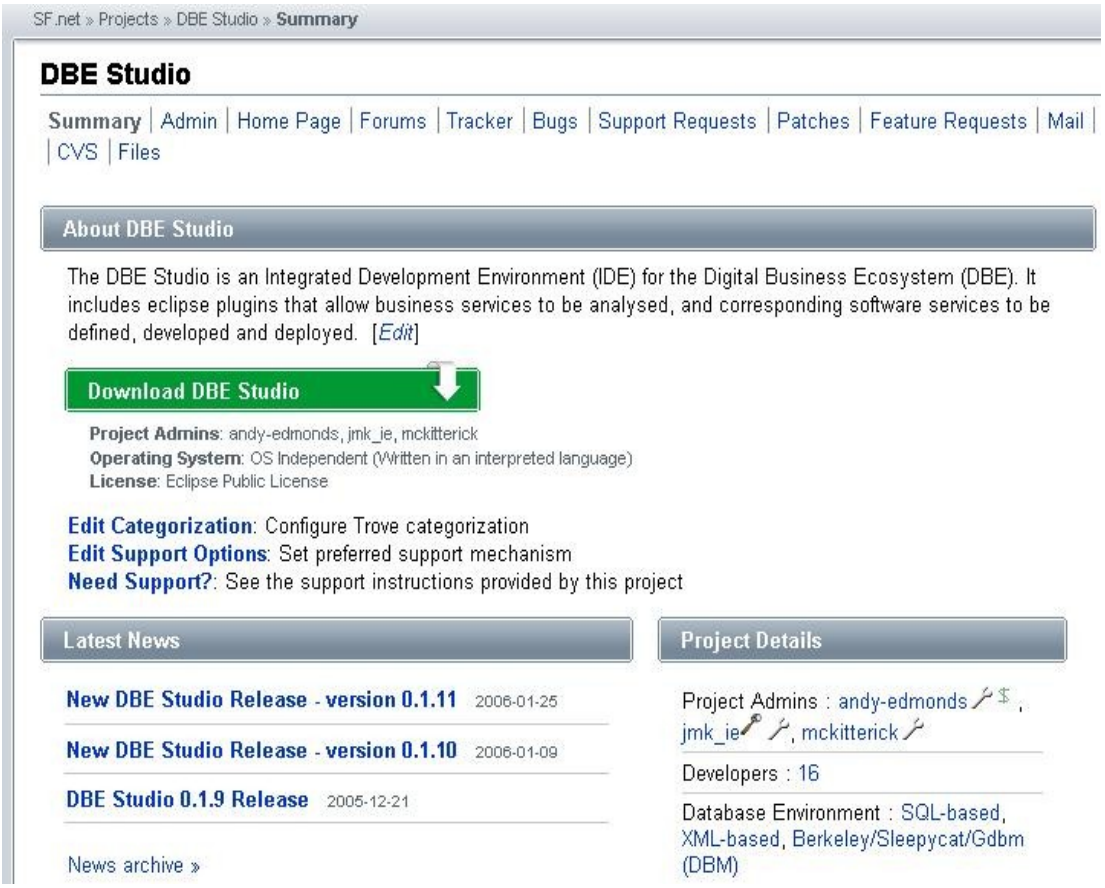

Figure 1: http://sourceforge.net/projects/dbestudio/

The DBE Studio SourceForge project was setup as part of the DBE Studio Integration task (see Figure 1 for the DBE Studio project on SourceForge). One of the initial tasks when creating this project was to decide upon an open source license which best suited the long term goals of the DBE project. After extensive consultation with both project partners and open-source legal experts, the Eclipse Public License (version 1.0) [3] was selected as the open source license for all DBE Studio source code and software releases. This license

reflects the close dependency with the Eclipse platform and associated frameworks, and also does not restrict the further development of DBE Studio based tools by external parties.

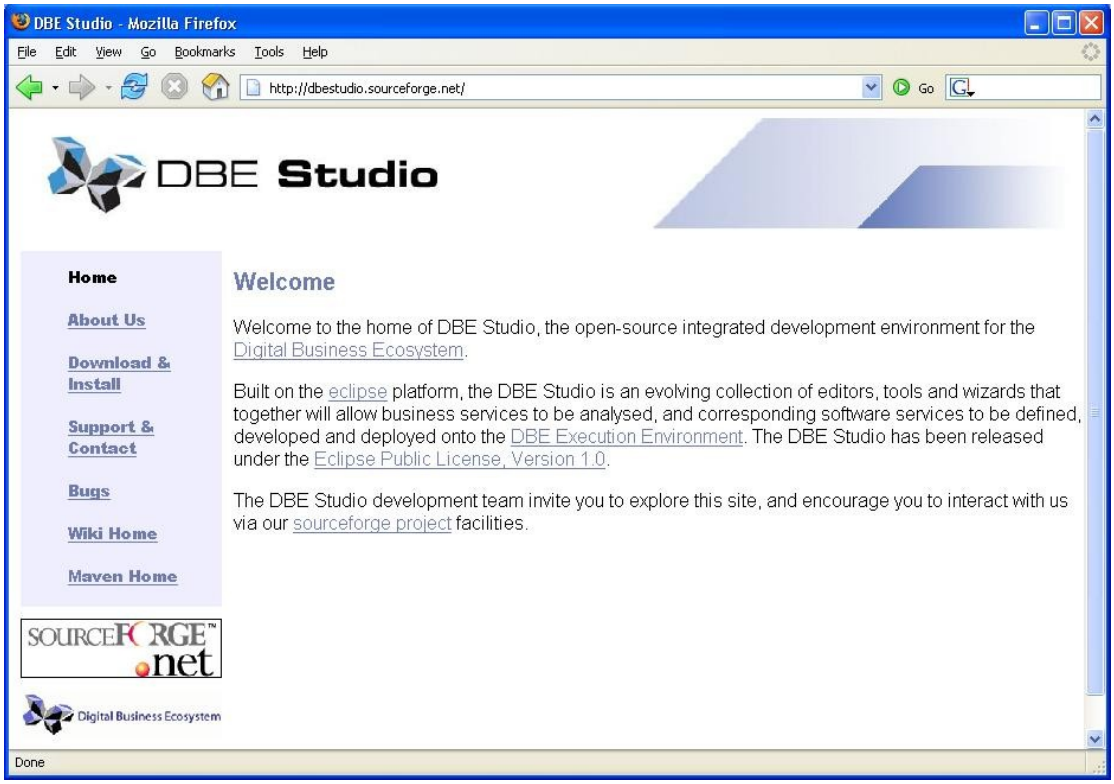

Figure 2: http://dbestudio.sourceforge.net

Using the project facilities provided by SourceForge, a project home page [4] was created (see Figure 2). This informs visitors and active users of the project's installation procedures and development status. The web site also provides access to the support features of this project, such as mailing lists, forums and bug trackers. Given that numerous components are being developed by different people within the DBE Studio project, users can target their bug, feature or support requests directly at the developer(s) responsible for the relevant plugin. There is also a link on the home page to our integrated build system web site which will be explained in more detailed in chapter 4, 'Build and Release Management'. SourceForge provides a CVS (Concurrent Versions System) [10] repository for each project - a necessary facility for integrated open source projects. All DBE Studio source code is available for browsing and downloading by anyone, but CVS commit rights are restricted to allow only active project developers to modify and add files. CVS administration for this project will be explained further in chapter 3, 'Component Integration'.

In additon to a project website, a wiki site [23] was created for the developers and users of the DBE Studio (see figure 3). The wiki provides integration and code contribution information for developers of plugin projects. These guidelines were created to assist developers in

contributing to the project. This will be discussed further in chapter 3, 'Component Integration'. The wiki also provides users with download and install guidelines. As a main feature of wiki sites, most information can be easily modified and updated by any user as the project matures. Additional, any user or interested party can add new wiki pages with any information related to the project. This should help to drive more involvement from the open source community. More community related work will be discussed further in chapter 6, 'Community Support'.

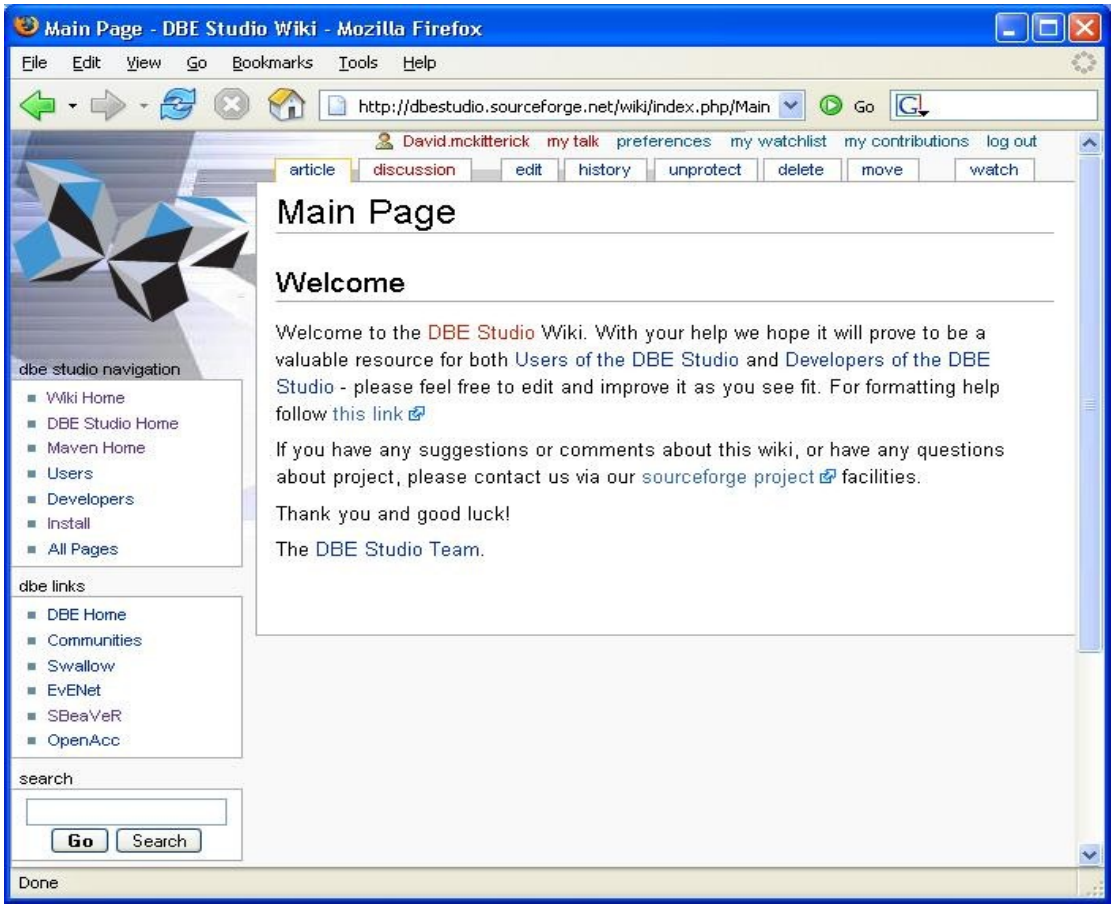

Figure 3: http://dbestudio.sourceforge.net/wiki/index.php/Main_Page

3. Component Integration

The DBE Studio currently comprises of 1 Eclipse feature and 26 separate plugins (taken from the DBE Studio release, version 0.3.0) being developed by 16 developers from 7 different partners within the DBE consortium. This large mix of components and developers requires highly coordinated integration procedures. As part of the DBE Studio Integration task, an integration guidelines document for DBE Studio project developers was released. This document outlines the recommended procedures for creating a plugin project within the project's CVS repository, integrating a plugin project into the self-contained Maven build system, using CVS with Eclipse and using common versioning techniques. The latest version of this document can be found in appendix A, 'DBE Studio- Developer's Integration Guide', and on the DBE Studio wiki site [23].

The component structure of the DBE Studio has been divided into three sections: core, editors and tools. This structure is reflected in the CVS repository and the naming convention used for plugin projects, e.g. '*org.dbe.studio.editors.bml*'. A summary of components within the DBE Studio are listed in Table 1. Each component group is described as follows:

- A **Core** component is generally a plugin which is central to the DBE Studio distribution and is used by some other DBE Studio plugins. Core plugins include *DBE Studio Preferences* and *DBE Studio Perspectives*.

- An **Editor** component is a plugin which provides an Eclipse editor view for creating and editing files and models, such as the *BML* (Business Modeling Language) [9] and *SDL* (Service Description Language) [11] editors.

- A **Tool** component is any plugin which does not provide an editor view or is not considered a core plugin within the DBE Studio. For example the *Service Exporter* tool.

Once the above structure was established, help was provided to most of the relevant partners with the task of creating their plugin projects. This assistance included such things as setting up the correct structure of their projects in conformance with the integrated build system, help with CVS operations and usage, advice on plugin development and UI techniques, and help with integration to other plugins (Eclipse and DBE Studio). A set of UI icons was also created and provided for the majority of plugin wizards and editors. This was accompanied with advice on best known approaches for UI tool design.

During the development of plugins for the DBE Studio, ongoing analysis was performed on the functionality provided by all plugins. The overall testing process involved testing individual plugins, and if problems were observed, a communication process was initiated with the partner or particular developer responsible for this plugin. This communication initially was done by email but following the early stages of the SourceForge project, most bug related issues were posted on the online bug tracker provided by SourceForge.net.

Component Groups	Component Name	Plugin ID	Contributing Partner(s)
Core	Feature	*org.dbe.studio.core.feature*	Intel Ireland
	Perspectives	*org.dbe.studio.core.perspectives*	Intel Ireland
	Preferences	*org.dbe.studio.core.preferences*	Intel Ireland
	Studio Examples	*org.dbe.studio.core.examples*	Intel Ireland
	Studio Help	*org.dbe.studio.core.help*	Intel Ireland, Others
	UI	*org.dbe.studio.core.ui*	Intel Ireland
	Connection Manager	*org.dbe.studio.core.connmgr*	Intel Ireland
	Security	*org.dbe.studio.core.security*	Intel, TCD, WIT
	Servent Toolkit	*org.dbe.studio.core.serventtoolkit*	Intel Ireland
Editors	BML Editor [13]	*org.dbe.studio.editors.bml*	TUC
	BML Data Editor [20]	*org.dbe.studio.editors.bmldata*	UCE
	BPEL Editor [12]	*org.dbe.studio.editors.bpel*	TCD
	Ontology Editor	*org.dbe.studio.editors.odm*	TUC
	SDL Editor [16]	*org.dbe.studio.editors.sdl*	Soluta
Tools	KB Toolkit	*org.dbe.studio.tools.kbtoolkit*	TUC
	Metering Wizard [21]	*org.dbe.studio.tools.accounting.metering*	WIT
	Ontology Viewer [15]	*org.dbe.studio.tools.ontologyviewer*	TUC
	QF-SDT [17]	*org.dbe.studio.tools.qfsdt*	TUC
	Recommender [18]	*org.dbe.studio.tools.recommender*	TUC
	SDL2Java Compiler [16]	*org.dbe.studio.tools.sdl2java*	Soluta
	SDL Storage	*org.dbe.studio.tools.sdl.storage*	Soluta
	Service Exporter	*org.dbe.studio.tools.exporter*	Intel Ireland
	SM Creator [15]	*org.dbe.studio.tools.smcreator*	Soluta
	SSL2SDL Compiler [14]	*org.dbe.studio.tools.ssl2sdl*	Soluta
	Test Case Generator [22]	*org.dbe.studio.tools.testcasegenerator*	UniS
	Identity Manager	*org.dbe.studio.tools.identity*	TCD
	UI Generator	*org.dbe.studio.tools.uigenerator*	Intel Ireland

Table 1: Component List for DBE Studio

To further improve the integration between DBE Studio plugins, some core plugins, as previously mentioned, were developed and integrated into this increasing set of tools. These plugins provided core functionality such as aggregating user preferences in a common location for all DBE Studio plugins, providing a general user help guide where all plugin user guides could be located, and creating a consistent set of task-oriented perspectives where related tools were visually connected. These core plugins will be explained in more detail in chapter 5, ‘Core Plugin Development’.

4. Build and Release Management

One of the most important tasks of DBE Studio Integration was to create a unified mechanism to build regular releases of the DBE Studio. This task can be separated into two parts:

- **Build management** involves integrating and administrating an effective build system for all component projects in the CVS repository, enabling a unified build and distribution of the software product.

- **Release management** involves creating and administrating a mechanism which supports regular user-friendly install and update procedures for the released software.

4.1 Maven as an Integrated Build System

As already used in the previous phase of project, it was decided to continue using Maven [5] as the open source build system for the DBE Studio. In addition to a build system, Maven provides software project management and inspection tools. Maven helps project integrators to build project components as part of unified builds, create project reporting information and present this information using an automatically generated web site. Extra Maven components were modified and added to the standard build system, including functionality to improve the building of Eclipse plugins automatically from Maven projects.

Figure 4: http://dbestudio.sourceforge.net/maven-site/main/docs/

Maven provides a selection of generated project reports such as dependency convergence and quality reports. An example of these quality reports are Checkstyle reports [8]. By using the in-built Maven Checkstyle plugin, developers can analyse their plugin projects for code convention compliance. These reports can be generated for each plugin project. Detailed guidelines have been added to the wiki site (as part of the developer's integration guide) to help developers write well-formatted code or even reformat existing code to reflect the code convention, as shown in Figure 5.

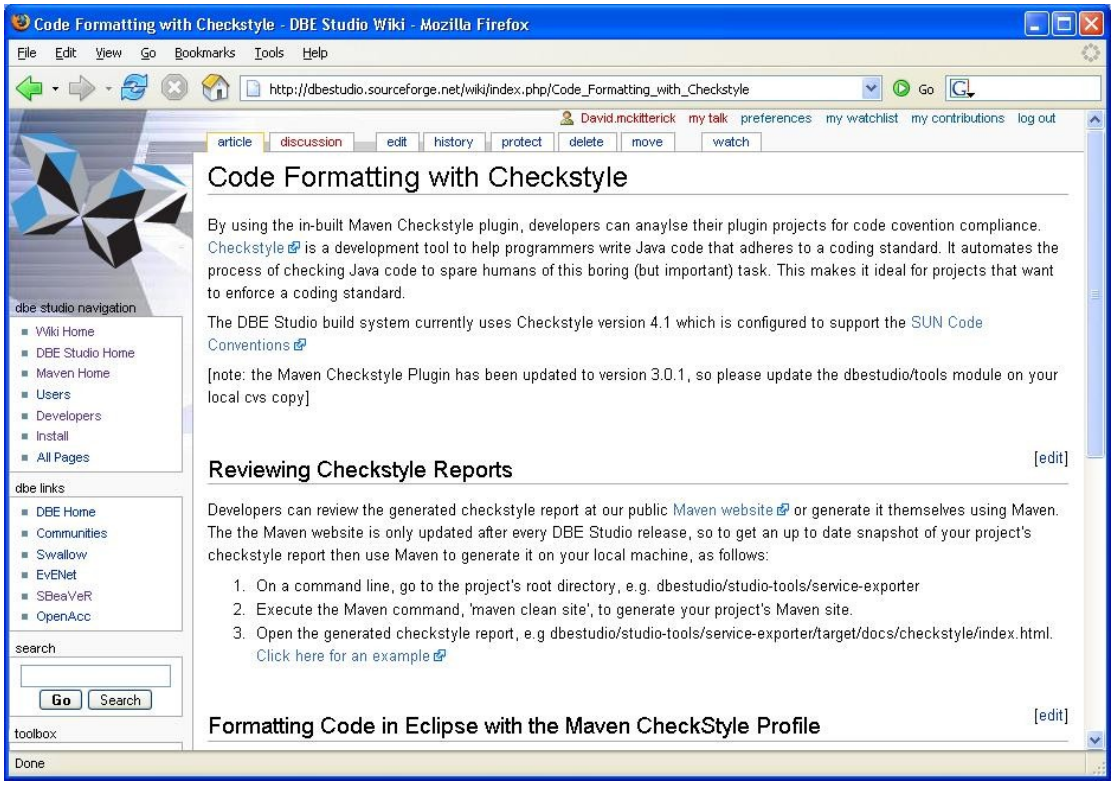

Figure 5: http://dbestudio.sourceforge.net/wiki/index.php/Code_Formatting_with_Checkstyle

The integrated build system provided by Maven was uploaded to the CVS repository so that all developers could have a standalone build system for the DBE Studio on their local machines. Each DBE Studio plugin project was created as a Maven project within the CVS structure, as described in chapter 3. Therefore by building each Maven project it was possible to automatically build an Eclipse plugin which was ready for distribution. Each plugin project requires a project.xml file which declares the project's id, version number and dependencies, along with other general information. Various Maven commands are available to compile the Java source code, run unit and component tests, build an Eclipse plugin and also generate a project web site. A build script was created so that a developer can simply run this script to build all components and copy the generated Eclipse plugins to a common directory location. The generated web site, as shown in Figure 4, provides information such as plugin project

dependency lists, code convention correctness reports, test results, versioning and change logs.

A Maven repository was setup on the DBE Studio SourceForge project web site to contain all versioned jars of the DBE Studio plugin classes. This repository also contains third-party open source jar dependencies which are used by the DBE Studio Maven projects. The build system also uses external Maven repositories to download third party jars during component builds. As an additional improvement to the build system, global dependency variables for common project jar dependencies were declared within the master (root) properties file. This enables developers to insert these variables into the dependency list within each project.xml file, therefore automating any future dependency version upgrades. This improvement allows for new builds of the DBE Studio to be created if common dependencies upgrades are required while reducing the effort required by individual plugin developers.

4.2 DBE Studio Install and Upgrade Mechanism

SourceForge provides a file release system where files can be uploaded and mirrored across the SourceForge network. These file releases are available for download by users from the project's SourceForge site. As the DBE Studio is built on top of the Eclipse platform and each component within the DBE Studio is distributed as an Eclipse plugin, it seemed obvious to follow the Eclipse install and update mechanism with which existing Eclipse users would be already familiar. This required the creation of an update site which specified the DBE Studio Feature version and the required plugins for this feature. For both first time users and existing users of the DBE Studio, a simple procedure enables them to manually (or automatically) install or update the latest version of the DBE Studio Feature from within their Eclipse installation. Some dependencies are required before installing the DBE Studio including Eclipse, EMF (Eclipse modeling Framework) and GEF (Graphical Editing Framework). Ensuring that the required dependencies were installed, a first time user can search for a DBE Studio Feature using the Eclipse 'Software Updates, Find and Install' mechanism. After entering the update site URL for the DBE Studio, i.e. http://dbestudio.sourceforge.net/install, they can select the latest feature and install it within their existing Eclipse installation.

Another available procedure involves downloading an archive of the DBE Studio distribution from the SourceForge file release system and then using the Eclipse software update mechanism to install this local archive. See Figure 6 for the Eclipse software update wizard displaying the DBE Studio Feature. For existing users of the DBE Studio, it is possible to either manually search for new feature releases or to set up automatic updates which will

automatically search for new feature releases and install the update if desired by the user.

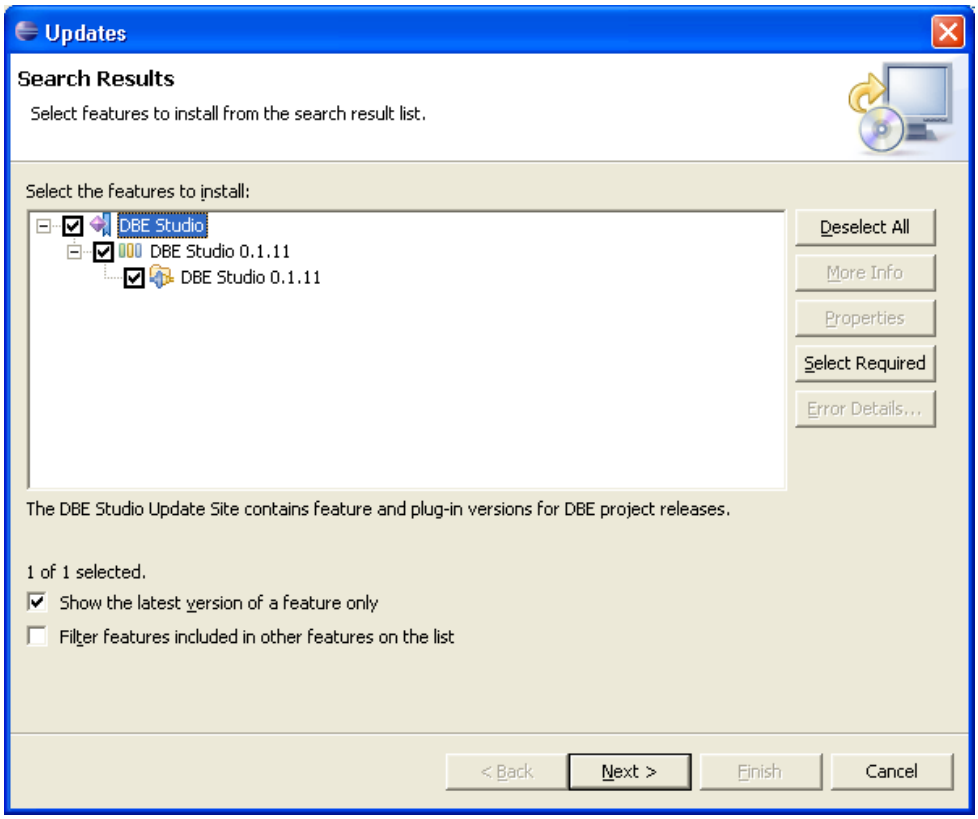

Figure 6: DBE Studio Install/Update Mechanism

5. Core Plugin Development

As part of the DBE Studio Integration task, several core plugins have been developed and contributed to the DBE Studio distribution. These plugins provide functionality for feature and UI support, user preferences, user help guides and examples, and a tool to deploy DBE services to the DBE Execution Environment. The following sections describe these plugins in more detail.

5.1 DBE Studio Feature

Most products based on the Eclipse platform are distributed as features. A feature describes the distribution of the product by listing the plugins included with this feature and the required dependencies of the feature and its plugins. The feature enables efficient versioning of different distributions. See Figure 7 for the DBE Studio Feature within Eclipse.

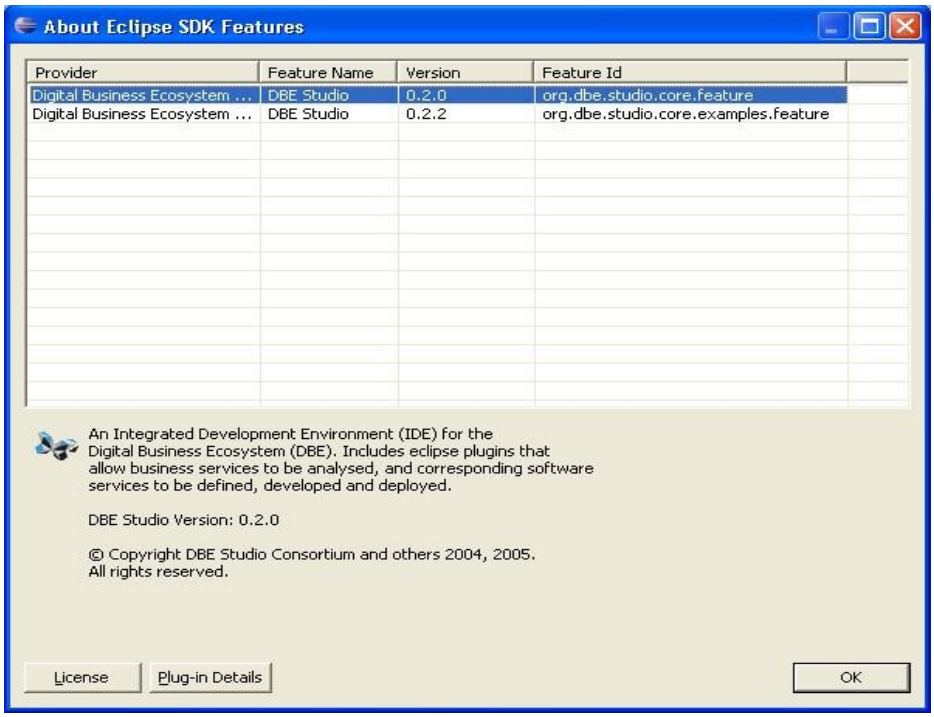

Figure 7: DBE Studio Feature within Eclipse

5.2 Examples

The Example Feature, which is distributed separately to the DBE Studio distribution, provides DBE projects containing example DBE services. These projects contain the relevant source

files and models for analysing, building and deploying the services. The latest examples feature also includes user interfaces for the DBE services based on the Open Laszlo UI technology. The example services implement potential DBE business scenarios in such domains as tourism and wholesaling. Help cheatsheets are provided for each example project. Figure 8 shows some of the example DBE projects open within the DBE Studio.

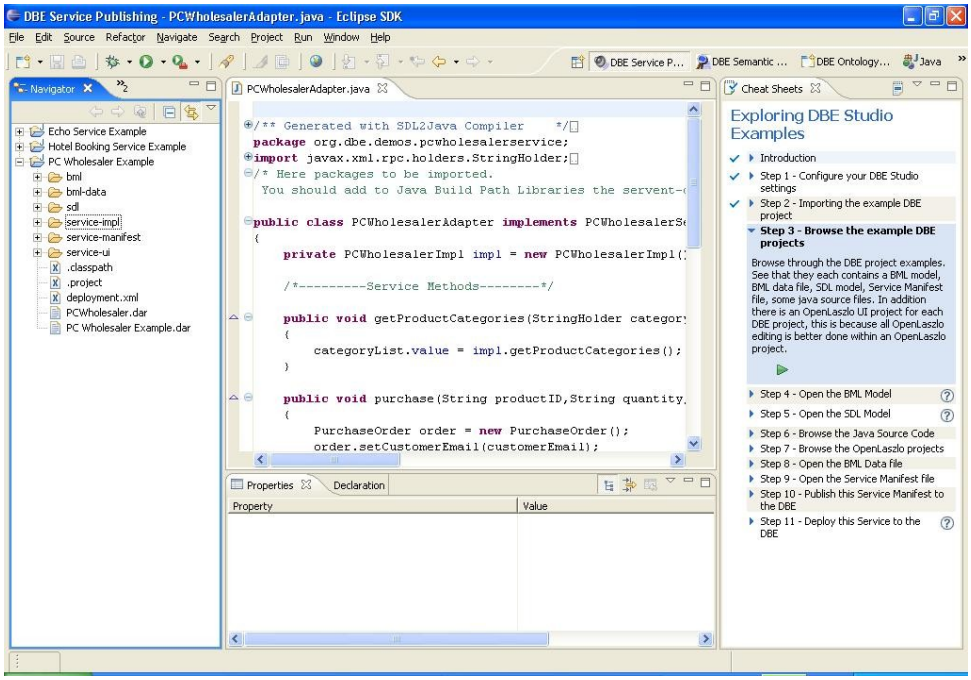

Figure 8: Using Examples within the DBE Studio

5.3 Perspectives

This plugin provides a set of perspectives for the DBE Studio. Examples of perspectives included are *DBE Service Publishing* and *DBE Service Development*. A perspective declares an arrangement of a set of related views and editors which are combined together to perform specific tasks. The DBE Studio perspective plugin also provides the concept of a DBE project. A user can create a new DBE project which provides a directory structure for all DBE files. This structure is also used by the *Service Exporter* tool for deploying services to a DBE Servent. Figure 9 shows a list of all the DBE Studio perspectives within Eclipse.

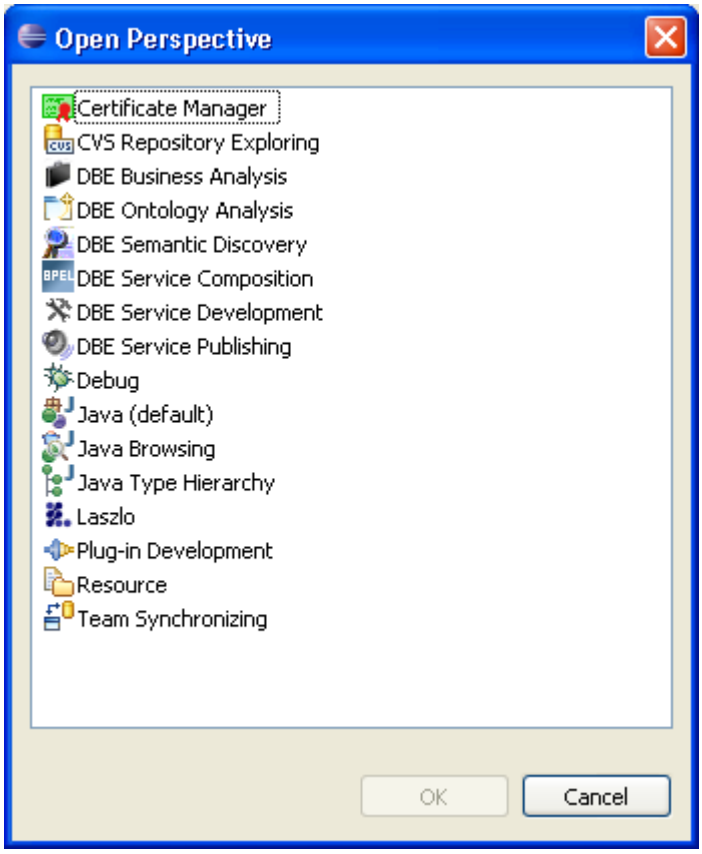

Figure 9: DBE Studio Perspectives

5.4 Preferences

The preferences plugin provides an entry in the Eclipse workspace preference menu for general DBE Studio preferences. This plugin also provides an extension point for plugin developers to add tool related preference pages to the DBE Studio preference list. Figure 10 shows the DBE Studio general preferences page with sub preference pages for specific plugins.

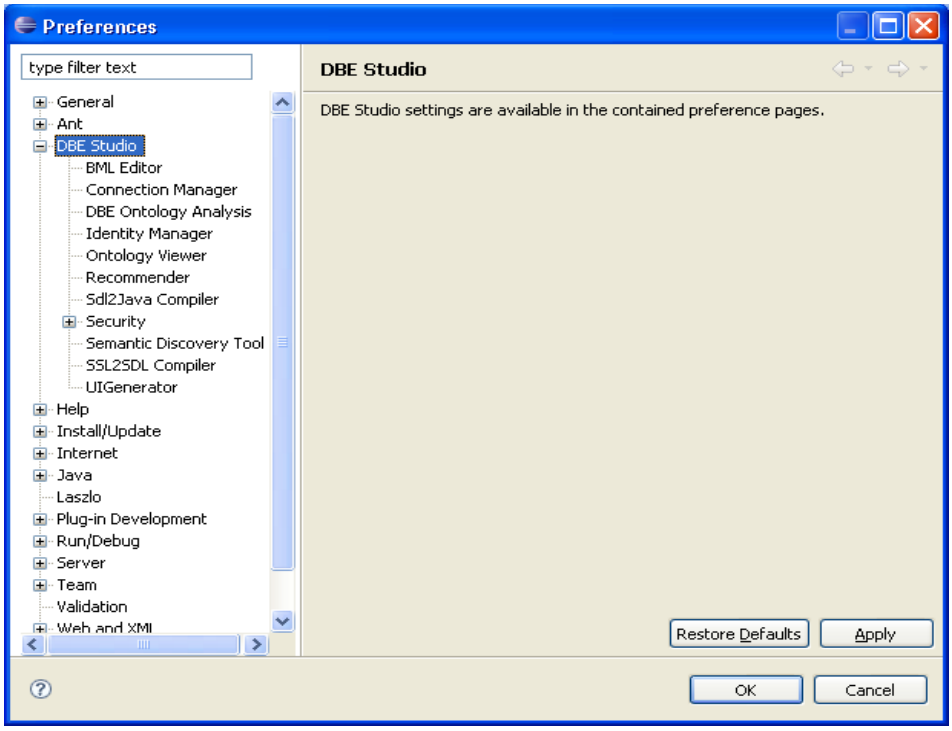

Figure 10: DBE Studio General Preferences Page

5.5 Connection Manager

The Connection Manager plugin was created and integrated to introduce a common point of contact between all DBE Studio plugins and the core services provided by the ExE. Previously, each plugin communicated directly to the ExE via the Servent's client side helper class. This required users to enter a ExE url into multiple preference pages. The connection manager plugin provides a single preference page for all ExE configuration. All DBE Studio plugins can read the saved user settings in this preferences page, therefore only plugin specific settings are required in the other preference pages. This introduces a common messaging and error handling procedure for all connections to the ExE. The user will be prompted if they wish to connect to a particular ExE node or they can choose to work offline, although some actions will always require a connection to be created. These preferences will be persisted to avoid repeated entries by the user. HTTP proxy settings have been included to enable users to interact with the DBE from behind a network proxy. Figure 11 shows the connection manager preference page.

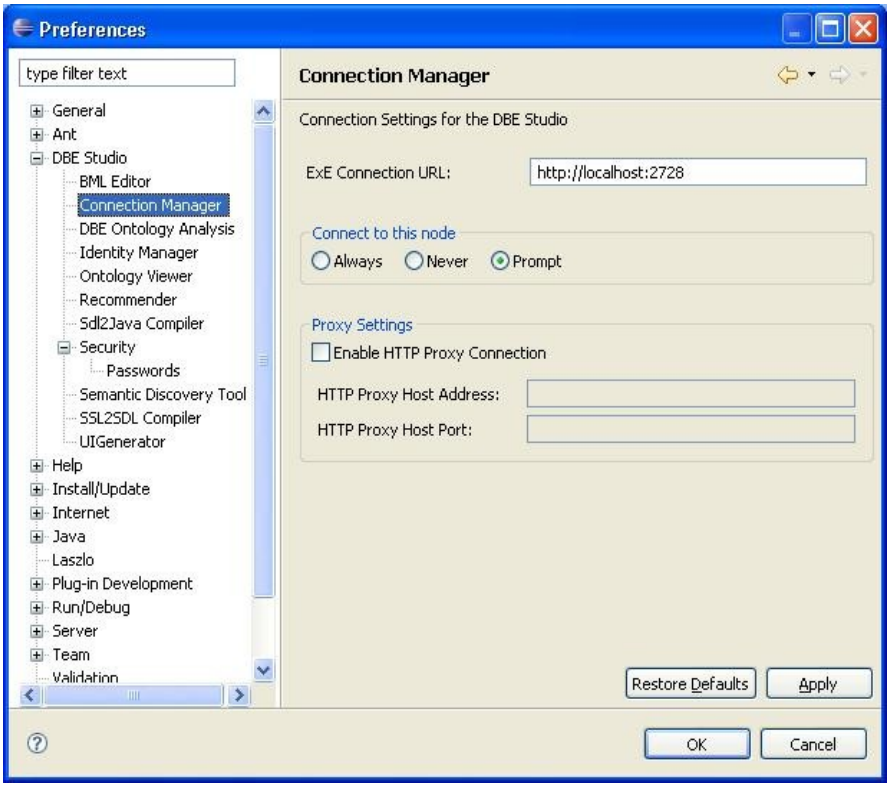

Figure 11: Connection Manager Preference Page

5.6 Service Exporter

The Service Exporter tool is implemented as an export wizard and allows a selected DBE project to be exported to the DBE. The wizard is comprised of two wizard pages. When a DBE project is selected, the service name and service description are automatically extracted from the service manifest file, stored within the project, and then presented to the user on the first wizard page (see Figure 12). The SMID is also extracted from the service manifest file and included in the generated deployment.xml file when the service exporter wizard is finished. The user must select the adapter class for the DBE service so that the service can be properly deployed within the Servent.

The Service Exporter also supports multiple UI types, composed DBE services, and adding Servent filters and properties to services. See Figure 13 for added service properties and filters using the second wizard page. This plugin is also integrated with the Accounting Metering wizard which enables users to specify metering filters for their deployed services. If the 'service requires metering' option is selected on the first wizard page, then the accounting metering wizard is added as the third wizard page to the service exporter wizard set.

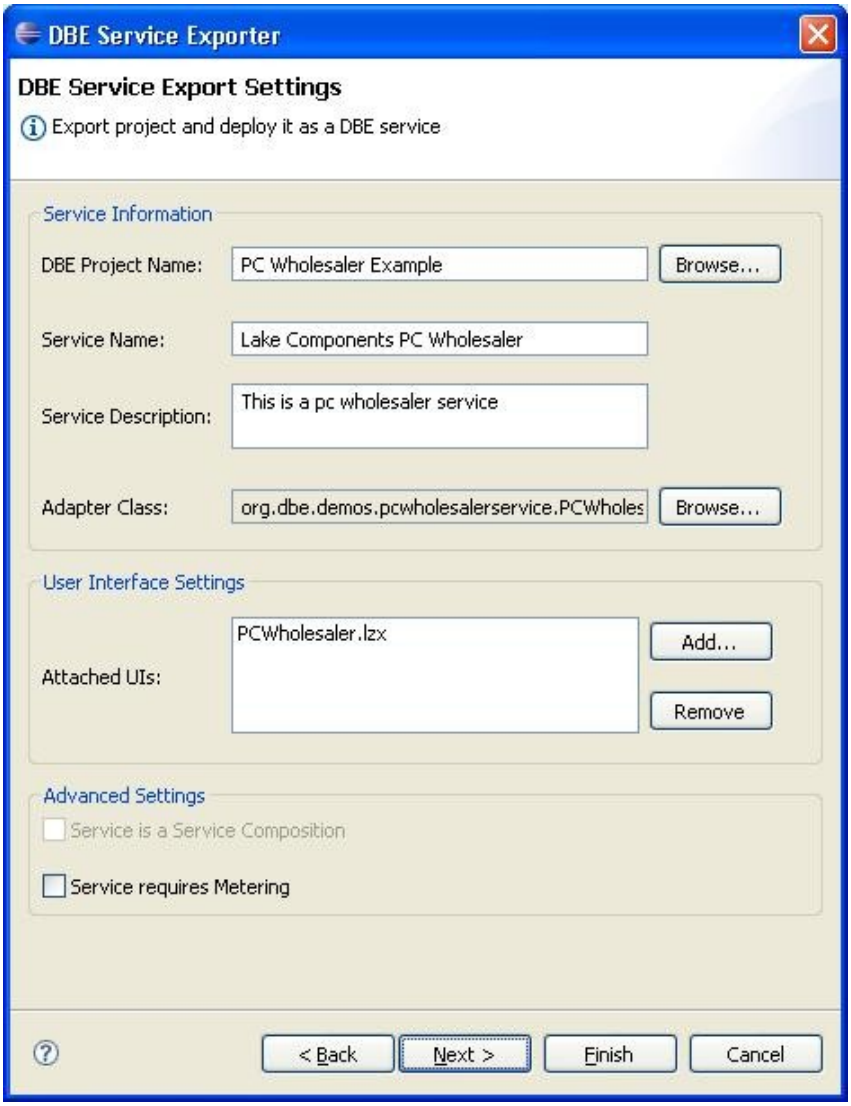

Figure 12: DBE Service Exporter, wizard page 1

When the service exporter wizard is finished, the service will be deployed to the specified Servent url, as declared in the connection manager preferences. This involves the generation of a deployment descriptor file, the deployment.xml. This file includes all important information required for successful deployment. After the generation of the deployment descriptor, all specified resources (included classes, libraries, deployment.xml, service manifest file and any UI resources) are added to a DBE archive (DAR) file. The DAR file is deployed to a specific Servent. The created deployment.xml and DAR files are also saved to the local project so that the deployment information can be loaded for redeployments with the service wizard or the DAR can be automatically deployed without using the service exporter wizard. The user will be prompted if these files exist within the project.

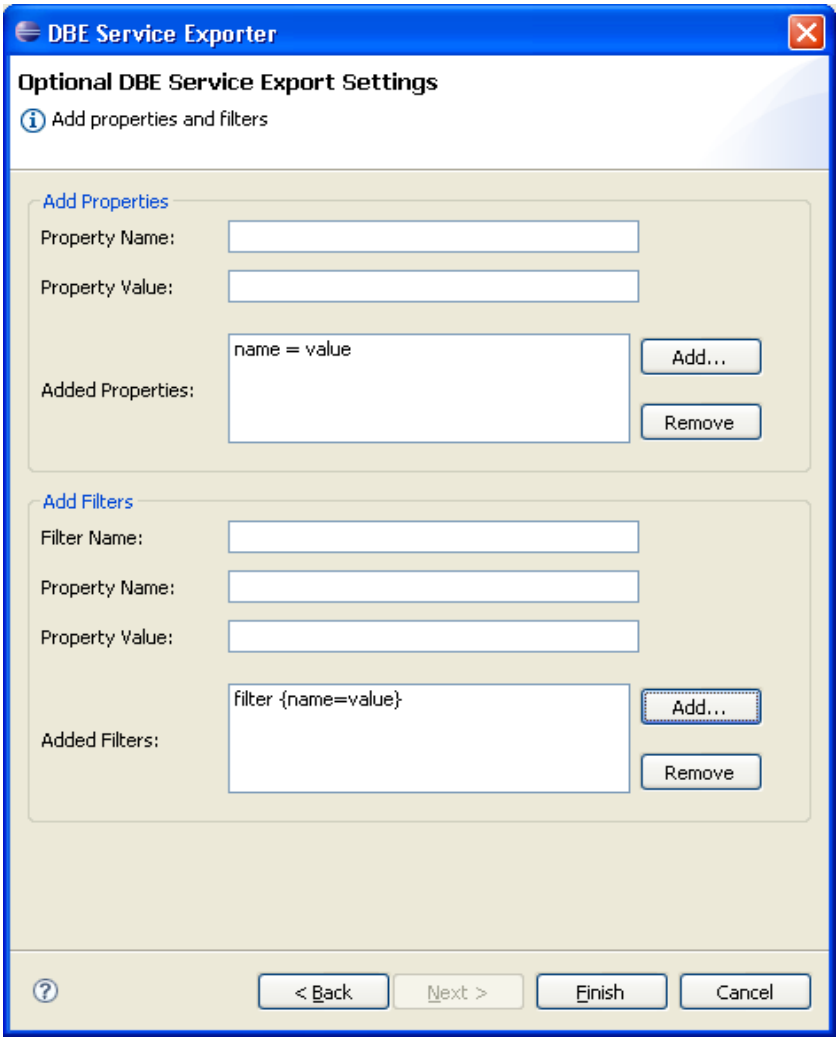

Figure 13: DBE Service Exporter, wizard page 2

5.7 User Interface (UI)

The UI plugin is a branding plugin for the DBE Studio Feature which provides branding details related to the feature and its plugins, such as licensing and contributing partner information. It is intended that this plugin will be extended to provide additional UI support functionality to the DBE Studio.

5.8 Help

The help plugin contains a set of user guides for most tools within the DBE Studio which can easily be accessed via the main Eclipse help menu. These user guides were provided by the contributing partners within the DBE Studio project. All documentation was written in DocBook [7], an XML based document type, which enabled easy integration and the generation of both html and pdf formated files. In addition, a Cheat Sheet is provided with

this plugin. This Cheat Sheet is an interactive tutorial which opens a view on the right hand side of the screen outlining steps on how to get started with the DBE Studio tools and therefore creating and deploying a DBE Service. Figure 12 shows the DBE Studio User Guide.

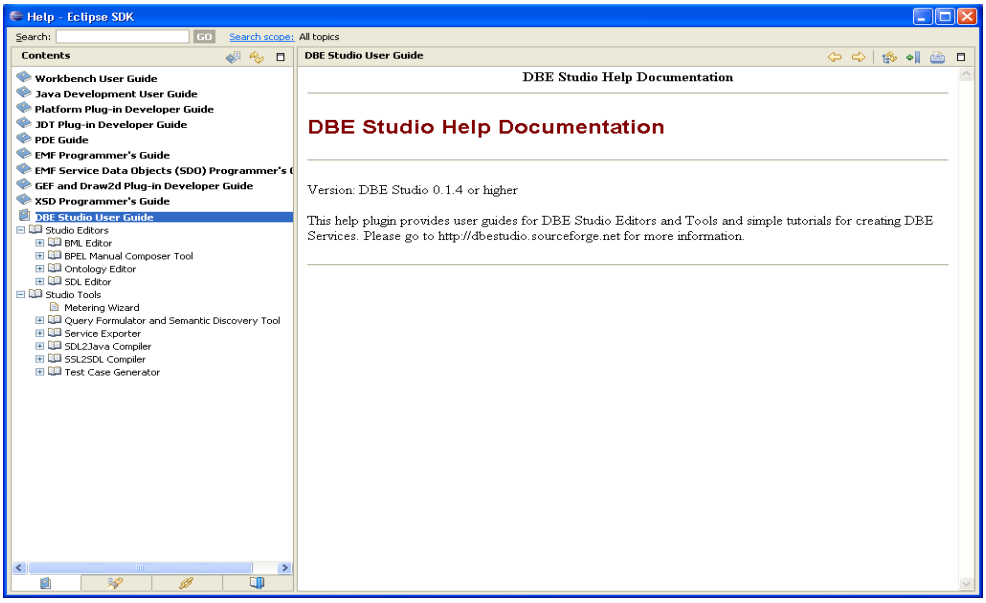

Figure 14: DBE Studio User Guide within Eclipse

5.9 UI Generator

The UI Generator plugin was developed as part of the 'User Interaction Design for DBE Services' task , D20.7 [24], by Intel Ireland. This plugin uses XSLT transforms to generate OpenLaszlo UI files from SDL models. OpenLaszlo was chosen as the default UI technology for DBE services [25]. UI generator plugin was integrated into the DBE Studio to improve the process of creating UIs and therefore making it easier for users to rapidly develop their DBE services. Figure 15 shows the first wizard page of the UI Generator plugin.

5.10 Security

This security plugin was a shared development task contributed by TCD, WIT and Intel Ireland. The purpose of this plugin is to provide common identity preferences for all DBE Studio plugins and to enable digital signing of DBE (xml based) models and files. This plugin was integrated into DBE Studio and is used by the Identity Manger and Contract Editor plugins. The Contract Editor has not been distributed with the DBE Studio at the time of writing this document, but this integration will conclude before the end of the project.

6. Community Support

As the users of any open source project are very important to the success and sustainability of its software, the DBE Studio team has made our users a priority during the ongoing development of the project. The main facility for user and community support is through the support features provided by SourceForge, as briefly outlined in chapter 2, 'DBE Studio on Sourceforge'. This chapter will explain these features in more detail. Another support aspect is the coordination and support of numerous SME and user code camps. These provided face-to-face interaction with the initial adopters of the DBE Studio.

6.1 DBE Studio Project Support

As the DBE Studio project is hosted on SourceForge, many support features have been provided for our users, including mailing lists, discussion forums, support requests , feature requests and bug reporting. Three mailing lists currently exist: a users mailing list for any user related topics, a developers mailing list for development and integration topics, and an announce mailing list for the communication of new distribution releases and any project news. Currently, three message forums exist: developers, help and open discussion. These allow users and developers of the project to openingly discuss any questions, usage problems and technical fixes. Support requests provides a mechanism where users can log a request for help and support relating to a particular tool or component. This request can be assigned to the responsible developer with expert knowledge on this plugin. Similarly, users can log feature requests and bug reports. Feature requests are where users can log a request for new functionality or features to be added to the next release. These requests are very useful as they provide invaluable feedback from our users and enable us to target future feature improvements to suit our users needs. Bug reports have an important part to play in on-going development of the software. They greatly help in improving the quality of future releases.

6.2 SME Code Camps

As integrators and administrators of the DBE Studio project, we have supported five SME code camps within the last year. These include sessions in Tampere (Finland), Zaragoza (Spain) and Birmingham (UK). In each code camp, we guided the attending driver SMEs through the installation and usage of the DBE Studio in a tutorial style session. The face-to-face interaction has not only helped our users but also provided the project developers with useful feedback, which has been added into subsquent development efforts.

7. Conclusion

As part of the DBE Studio Integration task a SourceForge project has been setup and administered, a project web site has been created, and a project structure and build system has been defined. An integration guideline document has been developed for contributors, and assistance has been provided throughout the lifecycle of DBE Studio plugin development to the contributing partners. The building, testing and release of the DBE Studio as well as the definition of two installation procedures has also been accomplished as part of this task. Finally, core DBE Studio plugins have been developed, including Examples, Perspectives, Preferences, a Connection Manager and a Service Exporter plugin.

This task is ongoing in nature, and it is foreseen that numerous releases of DBE Studio will be supported during the remainder of the project. These releases will include newly integrated DBE Studio components (including a Contract Editor), as well as support for any relevant upgrades of the DBE Studio dependencies, including Eclipse and Java. Several initiatives are also planned to enhance the build system and support for developers, including the further population of the DBE Studio Wiki to which anyone in the DBE Studio community can contribute.

Further work that will be conducted before this integration task concludes, but is not documented in this deliverable, will include the investigation of some improvements to the look-and-feel of the DBE Studio with the intention to build on the user experience within Eclipse. Such efforts may involve adding an integrated welcome page for the DBE Studio and improved icons and images for all tools.

8. Glossary

BML	Business Modelling Language
BPEL	Business Process Execution Language
CVS	Concurrent Versions System**s**
DAR	DBE Archive
DBE	Digital Business Ecosystem
EMF	Eclipse Modelling Framework
EvE	Evolutionary Environment
ExE	Execution Environment
GEF	Graphical Editing Framework
HTTP	Hyper Text Transport Protocol
IDE	Integrated Development Environment
KB	Knowledge Base
QF-SDT	Query Formulator – Semantic Discovery Tool
SBVR	Semantics of Business Vocabulary and Business Rules
SDL	Service Description Language
SM	Service Manifest
SME	Small and Medium Enterprises
SSL	Semantic Service Language
TCD	Trinity College Dublin
TUC	Technical University of Crete
UCE	University of Central England
UniS	University of Surrey
UI	User Interface
WIT	Waterford Institute of Technology
XSLT	XML Stylesheet Language Transforms

9. References

1. DBE Consortium, DBE Studio SourceForge Project, http://sourceforge.net/projects/dbestudio.
2. SourceForge.net, https://sourceforge.net/docs/about
3. OpenSource.Org, Eclipse Public License, http://www.opensource.org/licenses/eclipse-1.0.php
4. DBE Consortium, DBE Studio Home Page, http://dbestudio.sourceforge.net
5. Apache.org, Maven home page, http://maven.apache.org
6. Eclipse Open Source Community, http://www.eclipse.org/
7. DocBook Technical Committee Document Repository, http://www.oasis-open.org/docbook/
8. Checkstyle Project, Home Page, http://checkstyle.sourceforge.net/
9. ISUFI, DBE Deliverable: D15.1 - BML First Release
10. GNU, Concurrent Versions Systems, http://www.nongnu.org/cvs/
11. SOLUTA.net, DBE Deliverable, D16.1 - Service Description Models and Language Definition
12. TCD, DBE Deliverable, D17.2 - Manual Composer Tool
13. TUC, DBE Deliverable, D15.2 - BML Editor, 2nd Release
14. Soluta, DBE Deliverable, D16.4 - SSLCompiler
15. TUC, DBE Deliverable, D15.4 - Ontology Creator/Importer/Viewer
16. Soluta, DBE Deliverable, D18.5 - Implementation of the SM Editor
17. Soluta, DBE Deliverable, D16.2 - SDL Specification for the DBE Platform
18. TUC, DBE Deliverable, D24.6 - Query Formulator and Semantic Discovery Tool, Final Release
19. TUC, DBE Deliverable, D14.6 - Final Release of the Recomender
20. UCE, DBE Deliverable, D20.8 - GUI for M1 Authoring Tool/Wizard
21. WIT, DBE Deliverable, D36.4 - A set of Accounting Software Building Blocks for Automatically Composed Services
22. University of Surrey, DBE Deliverable, D10.2 - Complete Test Automation on DBE system
23. DBE Consortium, DBE Studio Wiki Home, http://dbestudio.sourceforge.net/wiki/index.php/Main_Page/
24. Intel Ireland, DBE Deliverable, D20.7 – User Interaction Design for DBE Services
25. Laszlo Systems, OpenLaszlo home page, http://www.openlaszlo.org/

10. Appendix A: Developer's Integration Guide

DBE Studio – Developer's Integration Guide

Digital Business Ecosystem

Intel Ireland

Versioning

Date	Author	Version	Changes
12/07/2005	Intel Ireland Ltd.	0.1	Initial Draft
12/07/2005	Intel Ireland Ltd.	0.2	Added CVS Guidelines
16/08/2005	Intel Ireland Ltd.	0.3	Added process diagrams, First Release
12/09/2005	Intel Ireland Ltd.	0.4	Added eclipse CVS configuration. Added section on Maven
04/02/2006	Intel Ireland Ltd.	0.5	Additional upgrade
26/11/2006	Intel Ireland Ltd.	0.6	Added new section, Building Plugin Projects

1. Introduction

This document is intended to outline some guidelines and rules for developers/partners contributing to the DBE Studio SourceForge.net project. The project can be accessed from:

- http://dbestudio.sourceforge.net/
- http://sourceforge.net/projects/dbestudio.

Details on becoming a SourceForge user or developer or any other specific SourceForge issues are out of the scope of this document. Please refer to the SourceForge website [1] for more information. This document will be divided into the following sections:

- Partners, Roles and Components.
- CVS Module Structure.
- CVS Procedures with Eclipse.
- Studio Plugin and Naming Conventions.
- Building Plugin Projects
- Versioning.
- Code Formatting with Checkstyle.

As a prerequisite to reading this document, all developers are advised to review a previous released document, the DBE Developers' Guide [2. We make the assumption that all developers are familiar with this document, especially with regards to code conventions, CVS (Concurrent Versions System) [8], and the Maven [3] build system. We use the Maven build system to manage the builds within this project and to conform to the conventions declared in the DBE developers' guide.

2. Partners, Roles and Components

In Figure 1, an organisational overview of the DBE Studio SourceForge project is presented (although this represents the initial status of the DBE Studio project. Additional partners and components have been included since this initial stage). At the lowest layer are the components (Eclipse [5] plugins), comprised in the DBE Studio. Moving up the diagram, the partners that are responsible for implementing these components are shown. At the next level, Intel is listed as the integrators of these components supplied by partners. The integrators have the responsibility of distributing a seamless and unified DBE Studio distribution to users using SourceForge. This involves the administration of the whole CVS [4] root and Maven builds, development of a Studio installation and update mechanism, and any other integration tasks needed. Please note, Intel is not responsible for managing and administrating each individual component; this is the responsibility of the implementing partner.

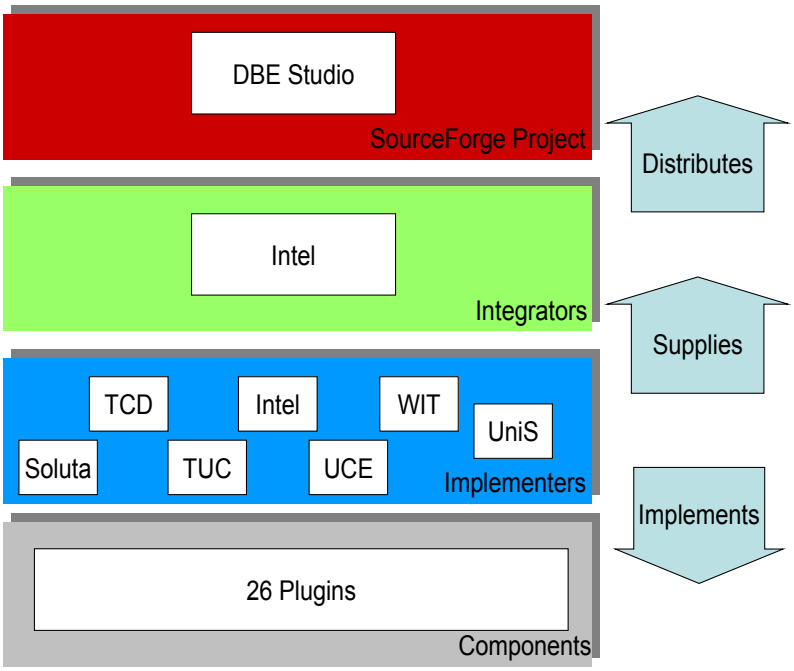

Figure 1: DBE Studio SourceForge Initial Project Overview (more components and implementers have been added since this digram was created)

3. CVS Module Structure

From the organisational diagram in Figure 1, the following CVS root hierarchy (Figure 2) has been devised. The main CVS root structure has been divided into two main sections, Maven build system directories and DBE Studio source directories. More information about the Maven build system directories can be found in the DBE developers' guide and will not be discussed in this document. The DBE Studio source directories are split into three main modules:

- **`studio-core`** is a module for any core components that are common to all studio plugins. This may include perspective plugins, a DBE studio Eclipse feature, branding and DBE Studio help.
- **`studio-editors`** is a module for any component that provides an editor view within the Studio. Each editor component will only have one plugin. These conventions for plugins will be discussed later in this document.
- **`studio-tools`** is a module for any plugin which provides a tool and does not provide an editor view within the Studio.

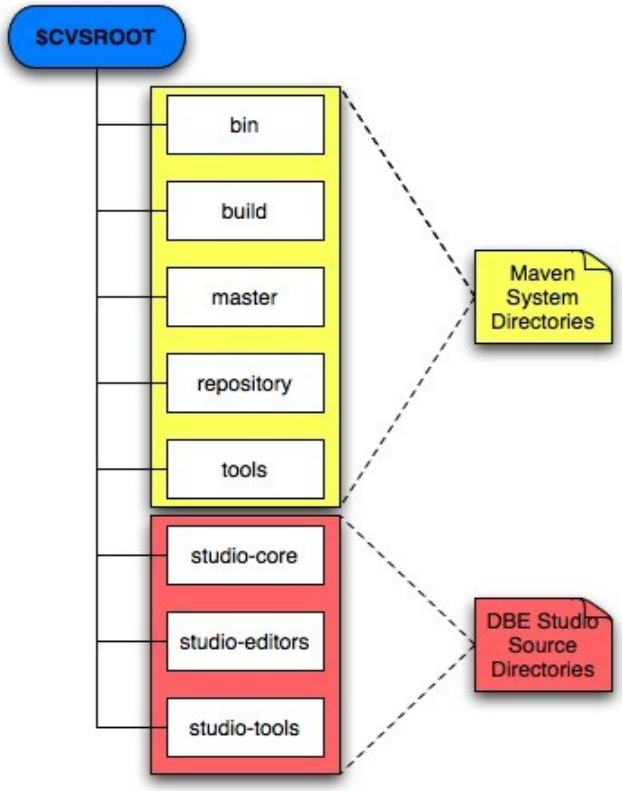

Figure 2: DBE Studio CVS Structure Overview

From the above CVS root structure the following table has been compiled. In table 1, the responsible partner for implementing the named component and their role, regarding that component, is listed. In the column named "**Location**" is the location, relative to the CVS root directory where the partner's component will be committed.

DBE Studio			
Partner	**Component**	**Role**	**Location**
Soluta	SDL Editor	Implementer	`$CVSROOT/studio-editors/sdl`
	Service Manifest Creator	Implementer	`$CVSROOT/studio-tools/sm-creator`
	SDL2Java Compiler	Implementer	`$CVSROOT/studio-tools/sdl2java-compiler`
	SSL2SDL Compiler	Implementer	`$CVSROOT/studio-tools/ssl2sdl-compiler`
	SDL Storage	Implementer	`$CVSROOT/studio-tools/sdl-storage`
TCD	Manual Composer (BPEL Editor)	Implementer	`$CVSROOT/studio-editors/bpel`
	Identity Manager	Implementer	`$CVSROOT/studio-tools/identity-manager`
	Security	Implementer	`$CVSROOT/studio-core/security`
TUC	Ontology Editor	Implementer	`$CVSROOT/studio-editors/odm`
	BML Editor	Implementer	`$CVSROOT/studio-editors/bml`
	Ontology Viewer	Implementer	`$CVSROOT/studio-tools/ontology-viewer`
	Query Formulator	Implementer	`$CVSROOT/studio-tools/qf-sdt`
	KB-toolkit	Implementer	`$CVSROOT/studio-tools/kb-toolkit`
	QF-SDT	Implementer	`$CVSROOT/studio-tools/qf-sdt`
	Recommender	Implementer	`$CVSROOT/studio-tools/recommender`
Intel	Studio Help	Integrator & Implementer	`$CVSROOT/studio-core/studio-help`
	Perspectives	Implementer	`$CVSROOT/studio-core/perspectives`
	Studio Examples	Implementer	`$CVSROOT/studio-core/studio-examples`
	UI	Implementer	`$CVSROOT/studio-core/ui`
	Feature	Implementer	`$CVSROOT/studio-core/feature`
	Service Exporter	Implementer	`$CVSROOT/studio-tools/service-exporter`
	Servent Client	Implementer	`$CVSROOT/studio-core/servent-client`
	Preferences	Implementer	`$CVSROOT/studio-core/preferences`
	UI Generator	Implementer	`$CVSROOT/studio-tools/ui-generator`
	Connection Manager	Implementer	`$CVSROOT/studio-core/connection-manager`
	Servent Toolkit	Integrator	`$CVSROOT/studio-core/servent-toolkit`
	Security	Integrator	`$CVSROOT/studio-core/security`
UCE	BML Data Editor	Implementer	`$CVSROOT/studio-editors/bml-data`
WIT	Metering Wizard	Implementer	`$CVSROOT/studio-tools/metering-wizard`
	Security	Implementer	`$CVSROOT/studio-core/security`
	Contract Editor	Implementer	`$CVSROOT/studio-editors/contract-editor`
UniS	Test Case Generator	Implementer	`$CVSROOT/studio-tools/test-case-generator`

Table 1: Responsible Partners and Components. (This table represents the cvs component structure on the DBE Studio SF project and does not directly map to the released plugins within the DBE Studio)

4. CVS Procedures with Eclipse

For most development tasks the preferred method to access code via CVS is through the use of the Eclipse built-in CVS feature, plugins and perspective. To access and manage CVS tasks using Eclipse follow the proceeding steps, as described below.

4.1 Eclipse CVS Configuration Pre-requisites

To guarantee correct operation with Eclipse it is necessary to configure how files are checked out of the CVS repository. In this case it means how Eclipse deals with CVS keywords. By default Eclipse will **not** expand the keywords and as such it is necessary to configure Eclipse to do this. To configure Eclipse: open the preferences page and find the preference page shown below (Figure 3: Eclipse CVS Preferences). Ensure that the "`Default text mode`" is set to "`ASCII with keyword expansion (-kkv)`"

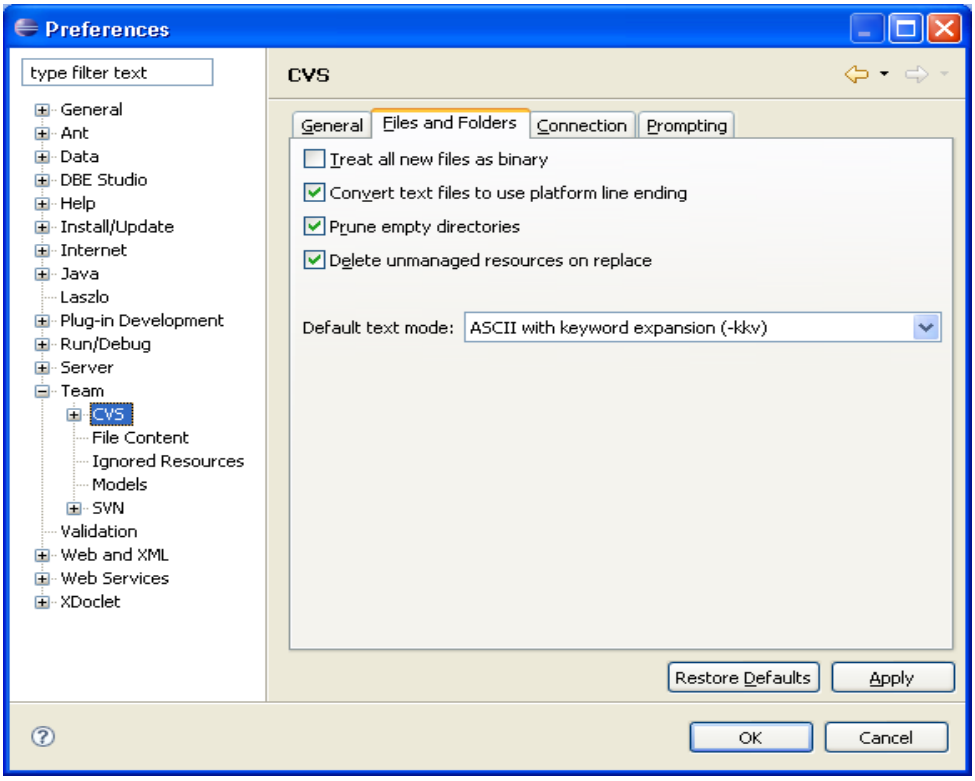

Figure 3: Eclipse CVS Preferences

4.2 CVS Workspaces

It is recommended to use separate Eclipse workspaces for CVS and plugin development tasks. This is due to the maven structure of the studio repository. The maven sub-projects, which provide the individual plugins for the studio, must be imported at the root Eclipse level for easy development. Therefore to retain the hierarchical structure of the CVS repository, we propose that developers have two workspaces, a CVS workspace and a development workspace. CVS tasks like checking out modules, branching and merging should be done in the CVS workspace and plugin development should be done in the development workspace. Workspaces can be easily switched by selecting "`File -> Switch Workspace`". This procedure with working with CVS, Eclipse and Eclipse workspaces is shown in Figure 4.

Figure 4: Overview of CVS Development Process with Workspaces

4.3 CVS Procedure Summary

This is a brief summary of basic CVS procedures within Eclipse, which are explained in more detail in the following sections.

1. Switch to CVS-workspace
2. Checkout the `dbestudio` Mainline (HEAD)
3. Create a branch on the `dbestudio` project
4. Switch to standard development-workspace
5. If necessary, create the Eclipse project (e.g. on the editor or tool plugin that you are developing) by using the Maven command, "`maven eclipse`", and import into your development-workspace
6. Develop and make the required changes. Note changes and new versions in the `templates\changes.xml` file within the plugin project.
7. Commit to the branch as required. (Implies CVS connection must be setup inside

standard development-workspace.)

8. Switch to CVS-workspace when ready to merge with Mainline (Head)
9. Perform CVS Merge on `dbestudio`

This process is illustrated in Figure 5.

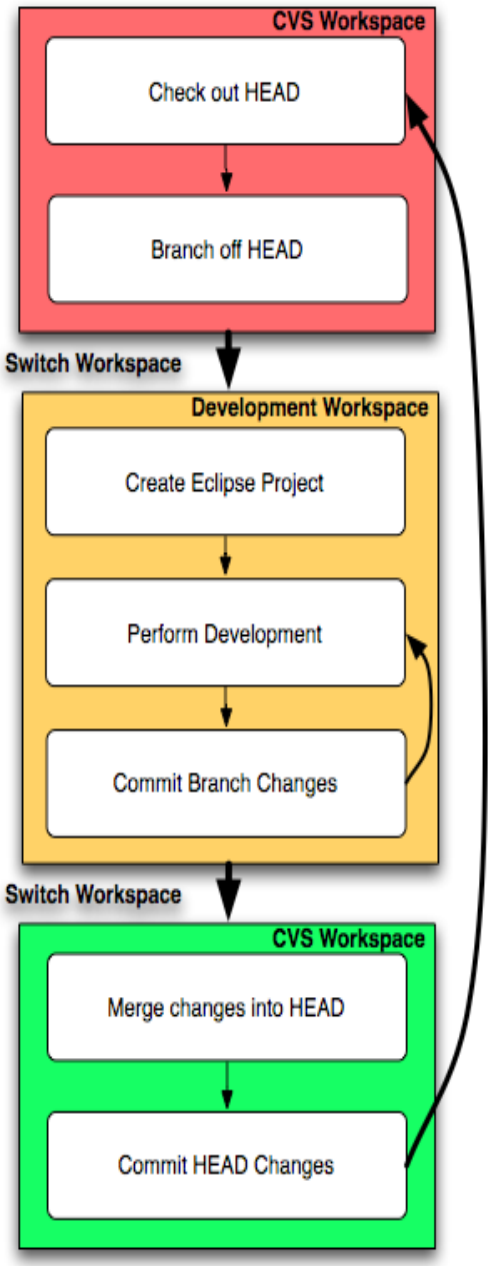

Figure 5: Detailed CVS Development Process with Workspaces

4.4 Mainline Checkouts in your CVS-workspace

If we want to check out the entire source that comprises the DBE studio the following steps will accomplish this.

1. Inside Eclipse, use "`File -> Switch Workspace`" (if necessary create it) and switch to your CVS-workspace.

2. Enter the CVS perspective within your Eclipse installation.

3. Right click anywhere in the CVS Repositories view, selecting "`New -> Repository Location…`".

4. An "`Add CVS Repository`" wizard pops up (see Figure 6).

5. Here, you need to enter the host and repository path, while also including your username and password. "`extssh`" should be chosen for the connection type. Click Finish. Depending on your network / firewall connection the connection may or may not work. A SSH2 connection proxy can be defined if necessary by going to "`Window -> Preferences -> Team -> CVS -> SSH2 Connection -> Proxy`".

6. Next, enter the Resource perspective.

7. Right click, selecting "`Import -> Checkout Projects from CVS`".

8. Select the "`dbestudio`" repository location (see Figure 7), click "`Next`".

9. Select the "`dbestudio`" module (see Figure 8), click "`Next`".

10. As shown in Figure 9, select "`Check out as a project in the workspace`", and then click "`Next`".

11. Select your CVS workspace location (see Figure 10), click "`Next`". It is recommended to use separate workspaces for CVS and plugin development tasks.

12. Then select "`HEAD`" for tagging, and click "`Finish`" (see Figure 11).

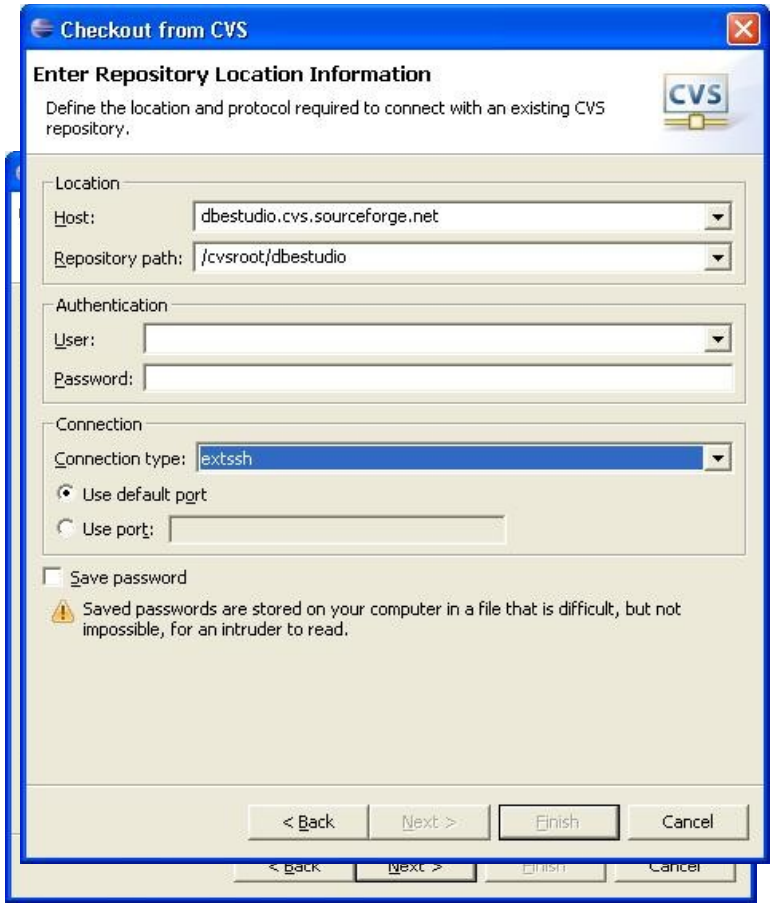

Figure 7: CVS Checkout, Selecting a Repository

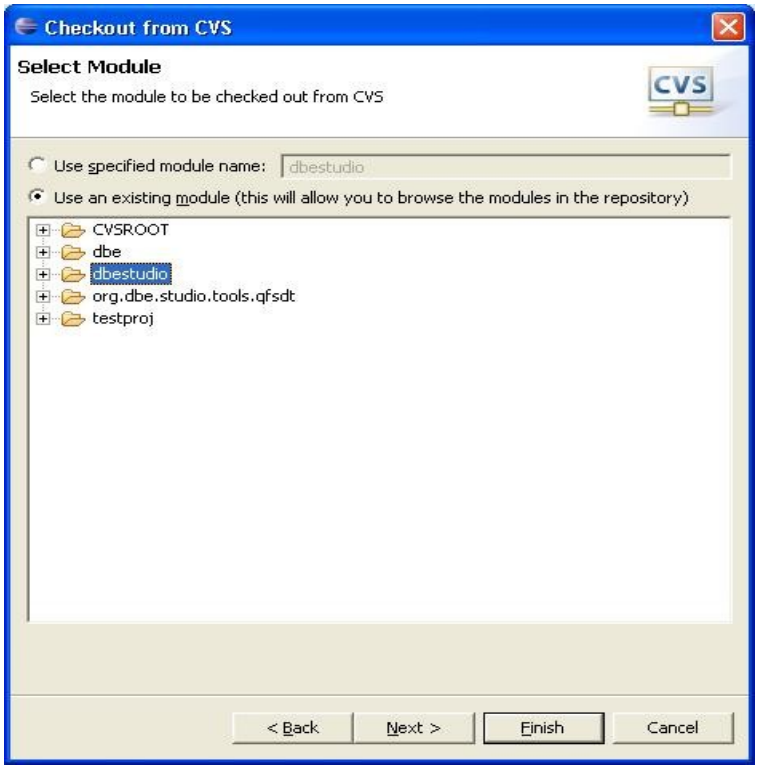

Figure 8: CVS Checkout, Selecting a Module

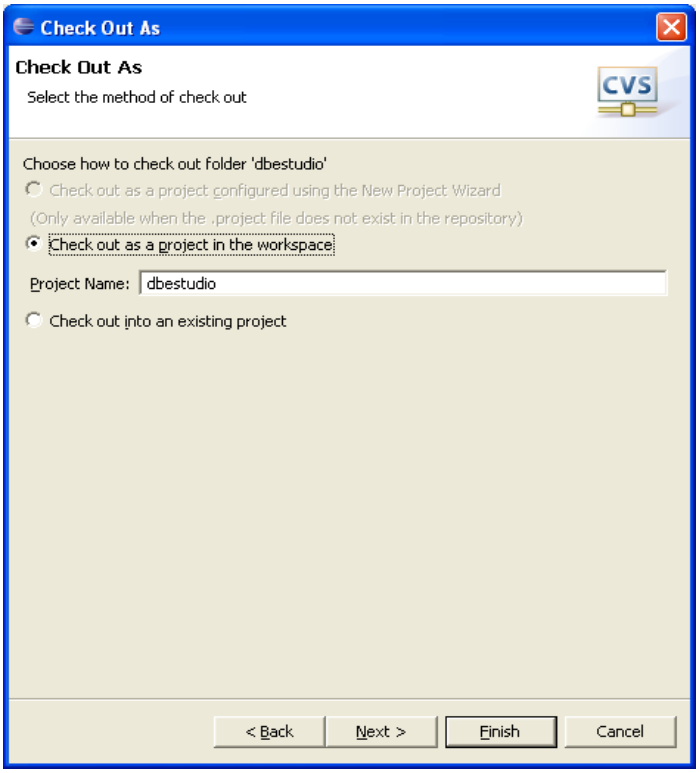

Figure 9: CVS Checkout As

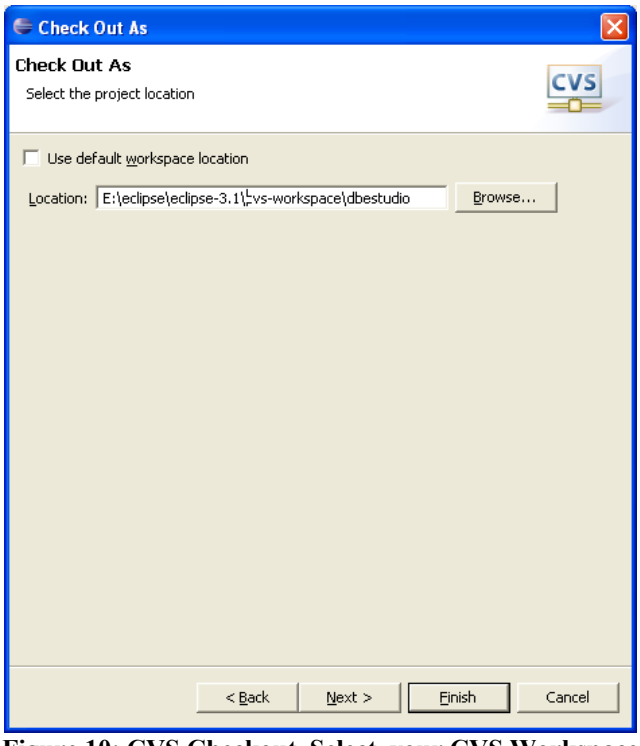

Figure 10: CVS Checkout, Select your CVS Workspace

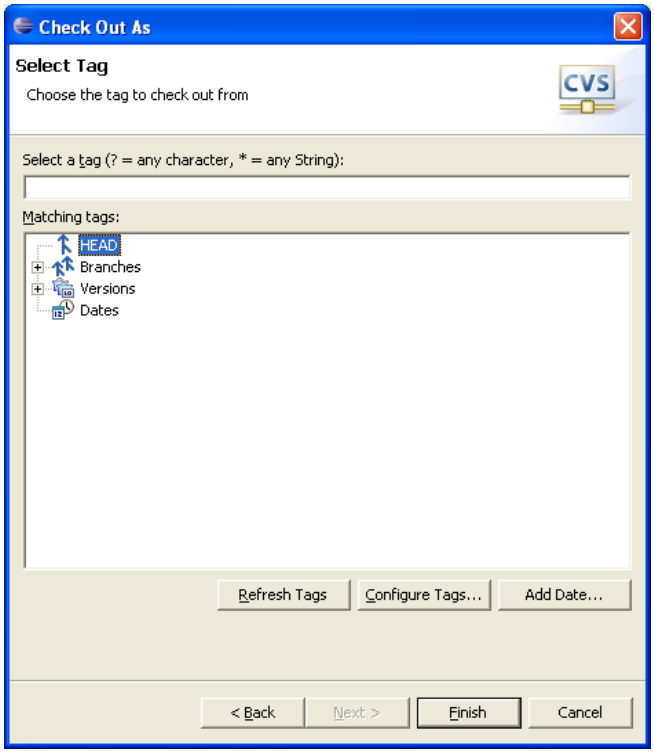

Figure 11: CVS Checkout, Select Head for tagging

4.5 Branching in your CVS-Workspace

Creating branches is done within the CVS-workspace. To create a branch off the mainline follow the proceeding steps:

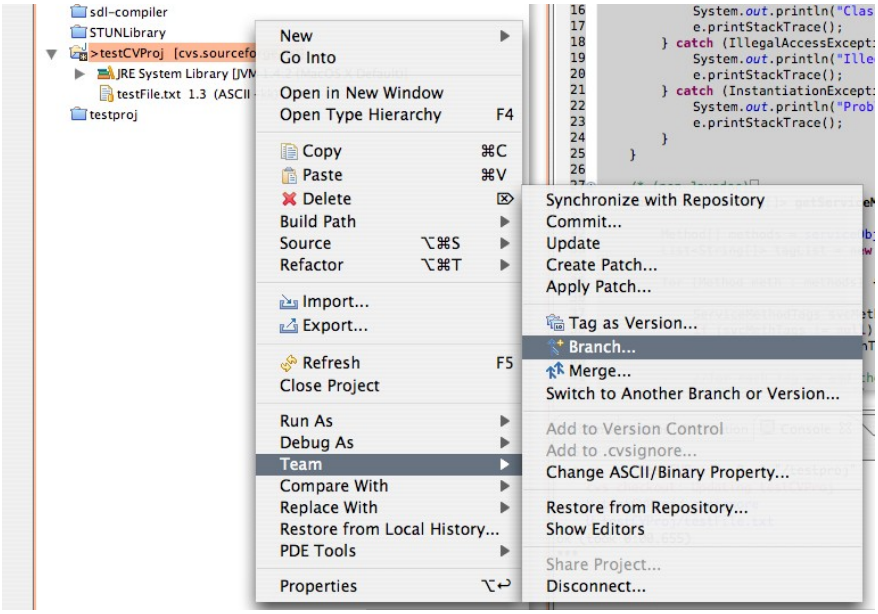

Figure 12: Branching in Eclipse (within the CVS-workspace)

Right click on the project root (Mainline HEAD) and select team and click on the entry "`Branch`" as displayed above. This will open the proceeding dialog.

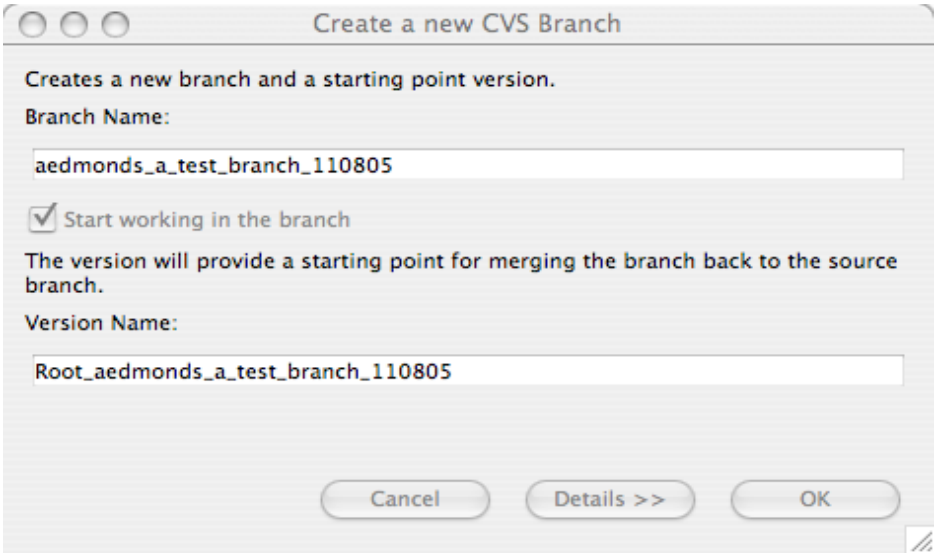

Figure 13: Naming a Branch

In this dialog, enter in the name of the branch in the format:

`<username>_<taskDescr>_<date>` e.g.
`krasik_updating_my_component_080805`

Where date is of the format `DDMMYY`.

There is no need to modify the Version name (see Figure 13). Click "`OK`" and Eclipse will begin to create a branch. Once the branching has completed, you will notice that your project root is tagged with the name of the branch (see Figure 14).

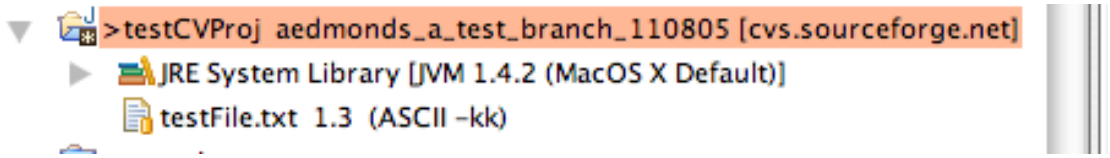

Figure 14: Eclipse Project Tagged with Branch Name

4.6 Developing in your Development-Workspace with Eclipse Project Imports

Once a branch is created, development can begin by importing an Eclipse project into your development-workspace. The following steps outline the procedure for this:

1. Use "`File -> Switch Workspace`" to switch to standard development-workspace
2. Create the Eclipse project files via Maven. E.g. on windows:
3. Open up a command prompt
4. Run `cvs-workspace\dbestudio\bin\set_env.bat` (or equivalent UNIX script)
5. Change to the directory containing the project to be worked on
6. Run "`maven eclipse`"
7. Use "`File -> Import... -> Import Existing Project into Workspace`" (i.e. development-workspace) to import the development project. Browse to the project location under the cvs-workspace directory.

4.7 Committing Changes in your Development-Workspace

Committing your changes to your branch can be done within your development-workspace, by right clicking on your project and selecting "`Team -> Commit...`" (see Figure 15).

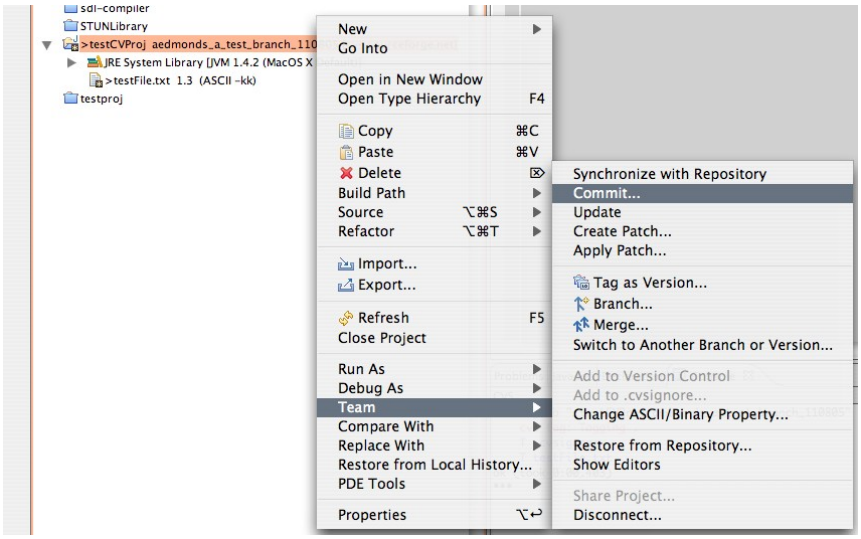

Figure 15: Committing Branched Changes in Eclipse

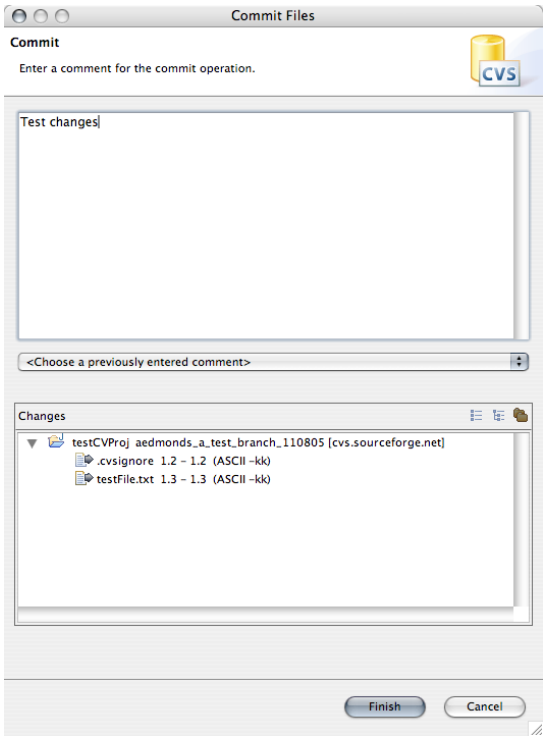

Figure 16: Commenting a Branch Commit in Eclipse

4.8 Merging your CVS Workspace

After making sure that all your code changes are committed to your branch, you must then switch back to your CVS-workspace. Now, in your CVS-workspace, it is time to merge those code changes from your branch. To do this, select "`Switch to Another Branch or Version`" from the Team menu.

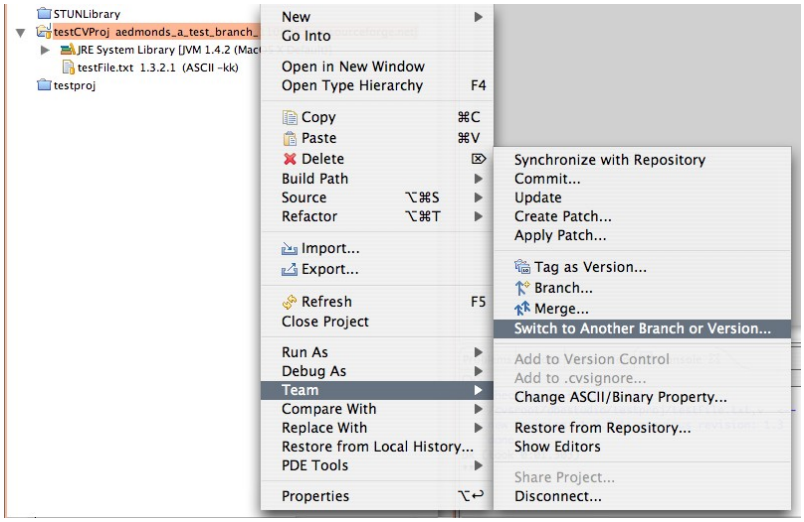

Figure 17: Switching to the Mainline for Merging

Selecting the entry now displays a dialog where you must switch back to your mainline. Select the second radio button and then the "HEAD" entry. Click "`Finish`".

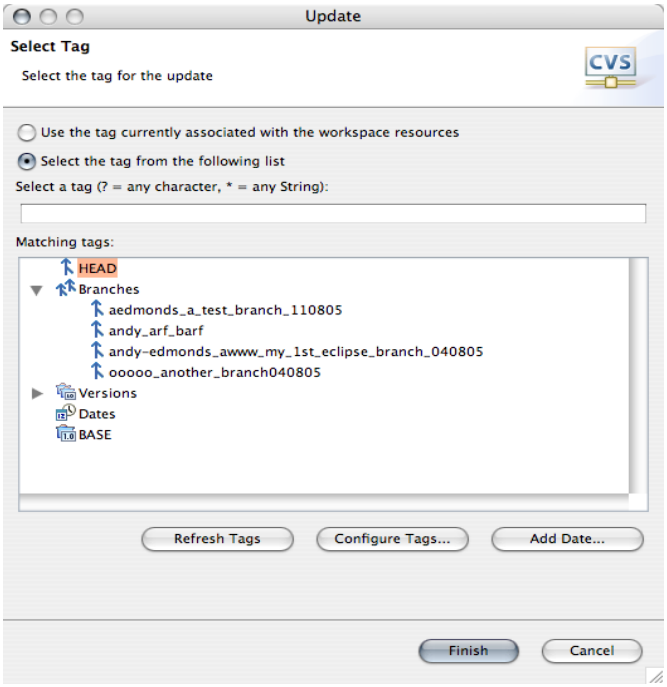

Figure 18: Selecting the Mainline

Now the actual merge is performed. Selecting the "`Merge`" entry in the Team menu does this.

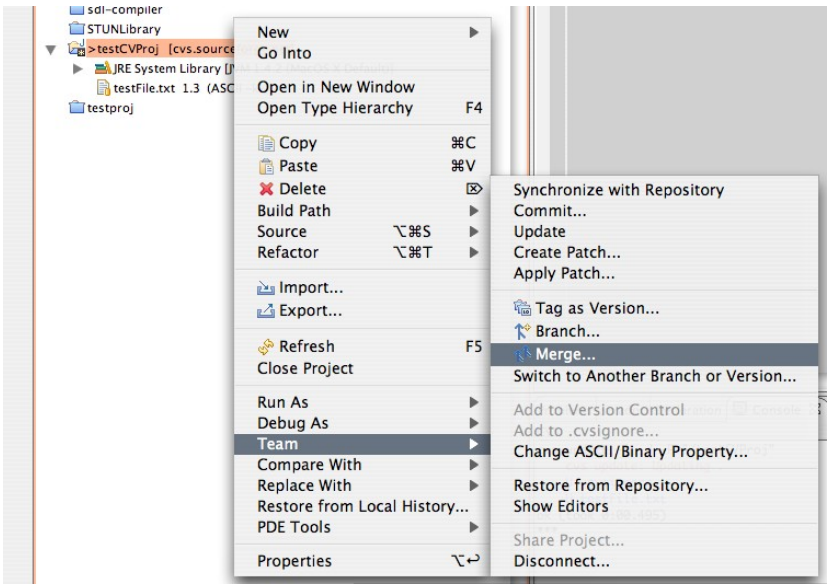

Figure 19: Initiating a Merge

Selecting "`Merge`" will display the following dialog (Figure 19), where you select the branch you want to merge into the mainline.

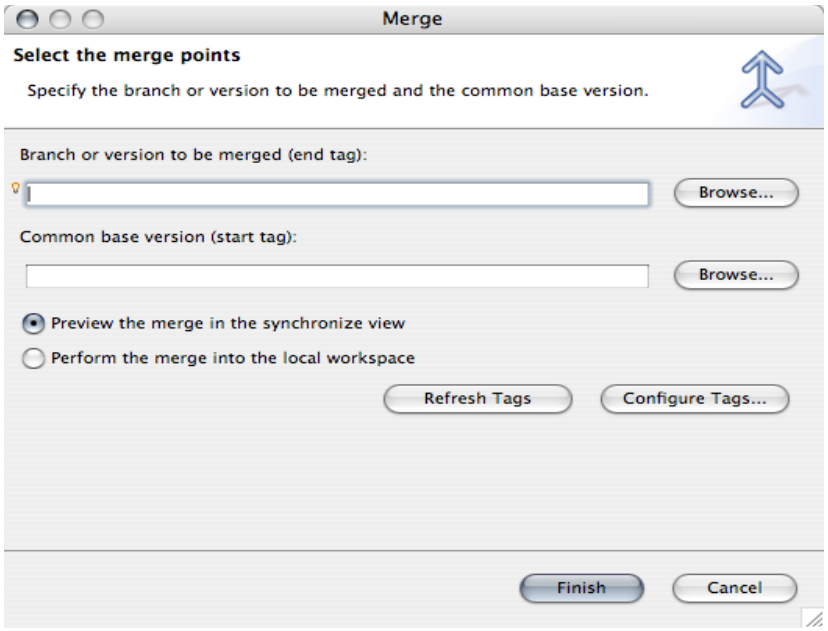

Figure 20: Merging in Eclipse

In the above dialog you need to select the branch you want to merge. To select the correct branch click on the first browse button and this will display all the branches available to merge (Figure 20). With the branch selected, select the radio button labelled "`Perform the merge into the local workspace`". Then to initiate the merge, click

"`Finish`". This will merge the changes made on the branch into your local working copy of the mainline**.** This is important to know as once merged into the local mainline you then have to commit the changes made by the merge into the CVS mainline stored on the CVS server. This is done in a similar fashion as described above (see Figure 21 – *note that when committing the branch tag will not be beside the project root as the merge is being performed on the HEAD*).

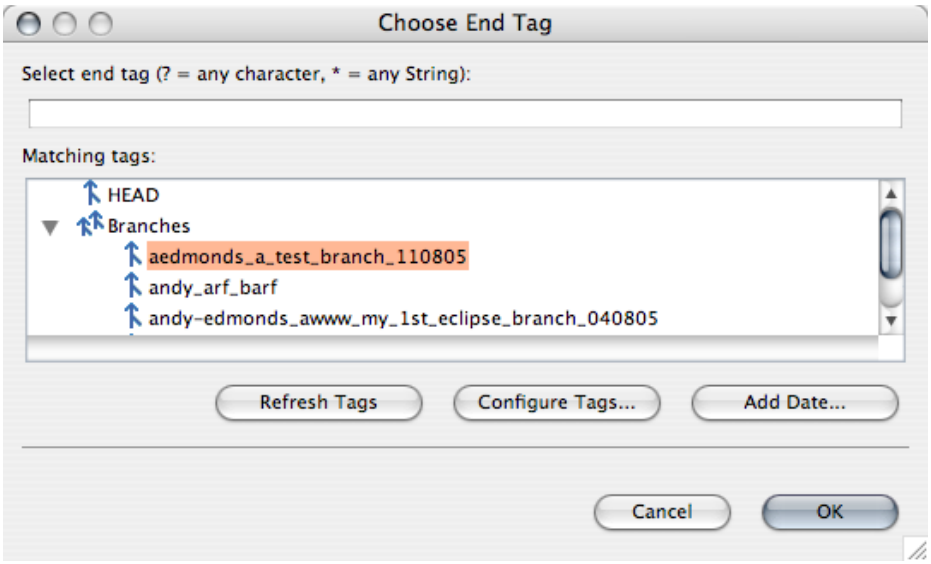

Figure 21: Selecting the Branch to Merge

Command Line Checkouts

If you wish to use the command line to interact with CVS, the following examples demonstrate some of the commands.

Mainline Checkouts

If you are a developer the following command will check out the main line (main trunk) of the DBE Studio

```
export CVS_RSH=ssh

cvs -z3 -d
:ext:$DEVELOPER_NAME@dbestudio.cvs.sourceforge.net:/cvsroot/db
estudio co -P dbestudio
```

Where `$DEVELOPER_NAME` is your sourceforge login name

Anonymous Checkouts

If you only want to check out the source and do not require commit rights then the following command will suffice:

```
cvs -d
:pserver:anonymous@dbestudio.cvs.sourceforge.net:/cvsroot/dbes
tudio login

cvs -z3 -d
:pserver:anonymous@dbestudio.cvs.sourceforge.net:/cvsroot/dbes
tudio co -P dbestudio
```

4.9 Maven and DBE Studio

Maven [3] is the tool that creates build of DBE components and looks after any related build dependency of that component. The usage of maven and how to create a maven compliant project is detailed in [2] but for convenience the most often used maven commands are listed below:

Command	*Description*

maven jar	Creates a jar file of the component classes suitable for non-Eclipse distribution. To create an Eclipse compatible jar plug-in you need to specify the following in your maven `maven.xml` file within your project:
	
	`<goal name="jar">`
	`  <attainGoal name="eclipse-plugin:create-plugin-dist" />`
	`</goal>`

maven eclipse	Creates an Eclipse project so the maven project can be worked with in Eclipse
maven clean	Removes generated files from project
maven site	Executes 'maven jar' and generates the plugin project's maven web site.

Table 2: Maven Commands

4.10 Maven Build Configuration

Maven reads the `build.properties` file that is present in your home directory. This file contains a global configuration for maven. One of the most important configuration variables is `maven.repo.remote`. This is where maven looks to see where it can find libraries to satisfy project dependencies. If this file (`build.properties`) is not present in your home directory, you can get a copy of it from CVS and place it in your home directory. When checked out, the file can be found under:

`<CVSROOT>/dbestudio/master/src/config/build.properties`

To set the maven environment in your command line console you must run the script which can be found in `<CVSROOT>/dbestudio/bin`.

5. Studio Plugin and Naming Conventions

The following guidelines should be adhered to when creating or committing to CVS:

- Plugins should not specify any perspective information - this will be done by a set of perspectives for which the Integration team is responsible.

- The functionality of an editor or tool should be contained within one plugin, although a plugin may have dependencies on other sub-projects. Multiple plugins per editor/tool will just complicate the installation process.

- Source code should have a standard package structure: i.e. '`org.dbe.studio.editors.X`' or '`org.dbe.studio.tools.X`', where `X` is the name of the component a partner is responsible for.

- Plugin IDs should be named after their relevant package structure, e.g. '`org.dbe.studio.editors.sdl`'

- Plugins should follow the common menu and label naming conventions provided by Eclipse human interface guidelines [9].

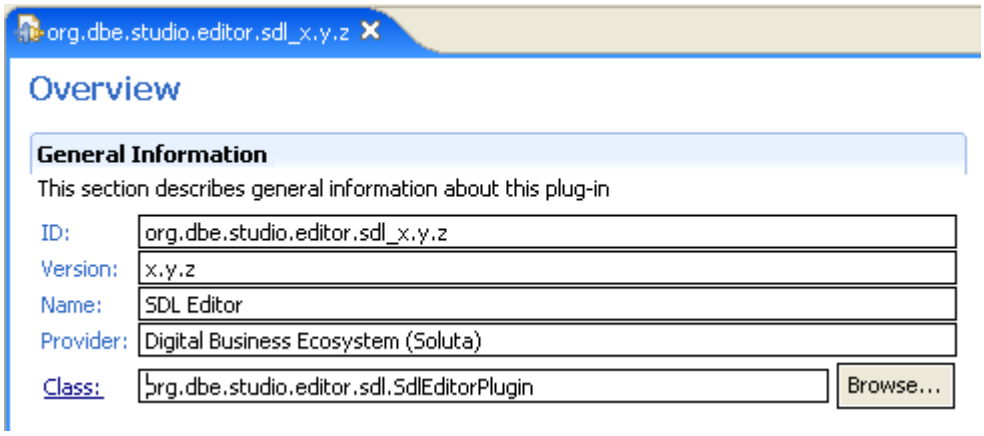

Figure 22: Example generated plugin.xml overview (version and id are taking from the project.xml, see next section 'Plugin Distributions')

6. Building Plugin Projects

6.1 Plugin Distributions

All plugin distributions are automatically generated by Maven. This is accomplished by utilising the Maven Eclipse plugin, which is invoked from the '`maven jar`' command. In order to apply this functionality to your plugin and Maven sub-projects, limited modifications are required from most partners.

The example below outlines a snippet of a plugin configuration file (see code snippet 1), which shows the necessary changes (line `13`, shown in **bold**) to make. The plugin `id` is given a general maven artefact id, which will be replaced by the project id declared in the respective maven `project.xml` file. This occurs when the plugin distribution is generated. Similarly, the `version` number is also taken from the `project.xml` file. This greatly improves the versioning, efficiency and distribution management required for a multi-partner platform distribution.

```
01 <plugin
02    name = "%pluginName"
03    id = "@maven.eclipse.plugin.artifact.id@"
04    version = "@maven.eclipse.plugin.artifact.version@"
05    provider-name = "%providerName"
06    class = "org.dbe.studio.editors.sdl.SdlEditorPlugin">
07
08    <requires>
09          ...
10    </requires>
11
12    <runtime>
13          <library name="@maven.eclipse.plugin.artifact.id@-
            @maven.eclipse.plugin.artifact.version@.jar">
14                <export name="*"/>
15          </library>
16    </runtime>
17
18 .../>
```

Code Snippet 1 Additions to `plugin.xml`

Run-time plugin dependencies can be extracted from the `project.xml` file and automatically added to the `plugin.xml` file during generation. Code snippet 2 shows the necessary additions (lines `05-07`, shown in **bold**), which must be added to a declared dependency within a `project.xml` file. This also helps in the management of dependency version management from related sub-projects.

```
01 <dependency>
02    <groupId>db-ecosystem</groupId>
03    <artifactId>some-project-id</artifactId>
04    <version>x.y.z</version>
05    <properties>
06          <eclipse.plugin.bundle>true</eclipse.plugin.bundle>
07    </properties>
08 </dependency>
```

Code Snippet 2 Additions to `project.xml`

6.2 Project Dependencies

The dependencies of a plugin project are declared in the project.xml file of that project. A dependency declaration consists of a group id, an artifact id and a version number. An example of this is shown below:

```
<dependency>
    <groupId>eclipse</groupId>
    <artifactId>org.eclipse.ui</artifactId>
    <version>${eclipse.version}</version>
</dependency>
```

Please note: the version numbers are now declared in a central properties file found on cvs at dbestudio/master/project.properties. These properties or variables should be used in each project.xml so that version numbers can be globally changed in the build system. The usage of version properties within a project.xml is as follows:

```
<version>${propertyName}</version>
```

An example of some of the version properties are shown below:

```
# Jar Dependencies Versions

# DBE Studio
studio.core.connmgr.version=0.3.0
```

```
studio.core.perspectives.version=0.2.2
studio.core.preferences.version=0.3.0
studio.core.serventclient.version=0.2.2
studio.core.serventtoolkit.version=0.2.2
studio.editors.bml.version=0.2.1
studio.editors.bmldata.version=0.2.1
studio.editors.bpel.version=0.2.3
...
# Eclipse
eclipse.version=3.2.0
# EMF
eclipse.emf.version=2.2.0
# GEF
eclipse.gef.version=3.2.0
# Servent
servent.impl.version=0.2.10
servent.core.version=0.2.10
dbetoolkit.protocoladapter.version=0.5.6
dbetoolkit.proxyframework.version=0.6.10
# KB
kb.pi.version=2.0.8
kb.mdrman.version=2.0
```

7. Versioning

All Maven sub-projects should follow a common versioning approach i.e. `x.y.z` where:

- `x` is a number denoting major milestone releases.
- `y` is a number denoting incremental feature additions to APIs and user interfaces.
- `z` is a number denoting bug fixes and minor implementation adjustments.

All component versions should follow major release and incremental feature versions of the DBE Studio. These version changes will be decided by the project integrators.

8. Code Formatting with Checkstyle

By using the in-built Maven Checkstyle plugin, developers can anaylse their plugin projects for code covention compliance. Checkstyle [6] is a development tool to assist developers in formatting their code to comply with a coding standard. It can automatically check each line of code which makes the task easier for developers. The DBE Studio build system currently uses Checkstyle version 4.1 which is configured to support the SUN Code Conventions [7].[1]

8.1 Reviewing Checkstyle Reports

Developers can review the generated checkstyle report at our public Maven website or generate it themselves using Maven. The the Maven website is only updated after every DBE Studio release, so to get an up to date snapshot of your project's checkstyle report then use Maven to generate it on your local machine, as follows:

1. On a command line, go to the project's root directory, e.g. dbestudio/studio-tools/service-exporter

2. Execute the Maven command, 'maven clean site', to generate your project's Maven site.

3. Open the generated checkstyle report, e.g dbestudio/studio-tools/service-exporter/target/docs/checkstyle/index.html.

8.2 Formatting Code in Eclipse with the Maven CheckStyle Profile

To help with formatting existing plugin projects for the checkstyle code convention, a developer can import a Maven Checkstyle profile into Eclipse. Eclipse allows the automatic formatting of Eclipse projects. The Maven Checkstyle profile xml file can be located on cvs at dbestudio/bin/maven_checkstyle-profile.xml. To import this profile into your Eclipse workbench to go 'Window -> Preferences' from the menu toolbar. Select 'Java -> Code Style -> Formatter'. Click on import and select maven_checkstyle-profile.xml file from the file system. Apply the changes and press OK.

Any project, source folder or Java file can be formatted by selecting the project, source folder or Java file. From the menu toolbar select 'Source -> Format'. Also, the imports can be organised from the menu toolbar or the context menu in the same way via 'Source ->

1 the Maven Checkstyle Plugin has been updated to version 3.0.1, so please update the dbestudio/tools module on your local cvs copy

Organize Imports'. Run the Maven site command and see the improvement in your Checkstyle report.

8.3 Ignoring Generated Code in Checkstyle Reports

If there is any generated code within a plugin project, then this can be ignored by the Maven checkstyle goal (command). To do this simply add a property to the project.properties file within the project. E.g. maven.checkstyle.excludes=**/deployment/**/*.java

9. Developing with the Connection Manager Plugin

This section outlines how a developer should use the Connection Manager plugin for all ExE interactions. The purpose of the Connection Manager plugin is to manage all ExE communications from DBE Studio plugins. Security and Identity plugins can also be integrated into the Connection Manager plugin. This plugin provides a simple interface (ExEConnector) for making all ExE invocations. A factory class is also provided for retrieving an instance of the ExEConnector interface. The sample code shows how each method of the this interface should be invoked. Parts of this code should be applied where existing Servent (Clienthelper) invocations are made.

9.1 Sample Client Code

```
1  import java.rmi.RemoteException;
2  import org.dbe.studio.core.connmgr.ConnectionException;
3  import org.dbe.studio.core.connmgr.ConnectionFactory;
4  import org.dbe.studio.core.connmgr.ExEConnector;
5  import org.dbe.studio.core.connmgr.proxy.ProxySearchListener;
6  import org.dbe.toolkit.pa.ProtocolAdapterException;
7  import org.dbe.toolkit.proxyframework.ServiceProxy;
8  import org.dbe.toolkit.proxyframework.Workspace;
9
10 /**
11  * This is an example client class showing how the ExEConnector
can be used.
12  */
13  public class exampleClient {
14
15   public void exampleMethod() {
16      try {
17           ExEConnector connector =
                 ConnectionFactory.getExEConnector();
18
19    // Optional (connect() is also called in following methods)
20           boolean connected = connector.connect();
   // Connection options are checked          21
22           if (connected) {
23           // start editor/tool
24           }
25
26        // OPTION 1: retrieve workspace object
27          Workspace myWorkspace =
                    connector.retrieveWorkspace("smid");
28          myWorkspace.invoke("operation", new Class[]
                   {String.class},
29          new Object[] {"value"});
30
31
32         // OPTION 2: retrieve dynamic proxy
33          SomeInterface someInterface =
                  (SomeInterface)connector.
34          retrieveDynamicProxy("smid", SomeInterface.class);
35
```

```
36
37      // OPTION 3: retrieve workspace with callback
38          ProxySearchListener listener = new MyListener();
39          connector.
               retrieveWorkspaceWithCallback("smid", listener);
40
41
42    // OPTION 4: retrieve dynamic proxy with callback
43           listener = new MyListener();
44          connector.
               retrieveDynamicProxyWithCallback("smid",
45                  SomeInterface.class,
46                  listener);
47
48
49   // OPTION 5 (not implemented yet): retrieve service proxy
50           ServiceProxy proxy =
              connector.retrieveServiceProxy("smid");
51
52
53     // OPTION 6 (not implemented yet): retrieve service proxy
with callback
54          listener = new MyListener();
55          connector.
               retrieveServiceProxyWithCallback("smid", listener);
56
57           } catch (ConnectionException e) {
58                  // Add Eclipse error logging here
59           } catch (RemoteException e) {
60                  // Add Eclipse error logging here
61           } catch (ProtocolAdapterException e) {
62                  // Add Eclipse error logging here
63           }
64      }
65 }
```

10. Appendix A: References

[1] SourceForge. http://www.sourceforge.net

[2] Trinity College Dublin. Dominik Dahlem. DBE Developer's Guide. Rev 0.4. May 2004.

[3] Maven build system. http://maven.apache.org

[4] Concurrent version system. http://www.gnu.org/software/cvs

[5] Eclipse project. http://www.eclipse.org

[6] Checkstyle Project, Home Page, http://checkstyle.sourceforge.net/

[7] Sun Microsystems, Sun Code Conventions, http://java.sun.com/docs/codeconv/

[8] GNU, Concurrent Versions Systems, http://www.nongnu.org/cvs/

[9] Eclipse User Interaction Guidelines Version 2.1, http://www.eclipse.org/articles/Article-UI-Guidelines/Index.html