Digital Business Ecosystem

Contract n° 507953

WP 24: DBE First Implementation

Del 24.1: DBE First Implementation

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

Contract Number: 507953

Project Acronym: DBE

Title: Digital Business Ecosystem

Deliverable N°: D24.1 (DBE First Implementation)

Due date: 31/04/2005

Delivery Date: 22/06/2005

Short Description:

This document describes research and preliminary implementation work done on the implementation of the first incarnation of the digital business ecosystem (DBE). The main contributions of this deliverable include a development specification, standards and processes for DBE, an open source implementation that encapsulates the design and architecture set forth in WP21, DBE Architecture Requirements. The implementation is functionally split between the Service Factory and Execution Environment. This deliverable is composed of both this document and code.

Partners owning: TCD

Partners contributed: TCD

Made available to: The Project Consortium and the EU

Versioning

Version	*Date*	*Name, organisation*	*Description*
0.1	04/06/05	Andy Edmonds, TCD	Initial Draft Version
0.2	04/14/05	Dominik Dahlem, TCD	Development Methodology
0.3	04/24/05	Andy Edomonds, TCD	Domain Analysis
0.4.1	04/25/05	Dominik Dahlem, TCD	Use Cases
0.4.2	04/25/05	Andy Edmonds, TCD	Restructure the Design & Implementation
0.5	04/26/05	David McKitterick, TCD	Proxy Toolkit, APA
0.6	04/27/05	Dominik Dahlem, TCD	Use Cases: DBE Portal
0.7	28/04/05	Dominik Dahlem, TCD	Design & Implementation: DBE Desktop Improved UML Class Diagrams
0.8.1	04/29/05	Andy Edmonds, TCD	Design & Implementation: DBE Portal
0.8.2	04/29/05	Dominik Dahlem, TCD	Introduction
1	04/29/05	Dominik Dahlem, TCD	Final Version
1.1	06/22/05	Dominik Dahlem, TCD	Reviewed version by Chris Catrie

Quality Check

1st Internal Reviewer: Miguel Vidal, SUN

2nd Internal Reviewer: Pierfranco Ferronato, Soluta.net

Table of Contents

Table of Figures

1 Executive Summary

This document describes the implementation of the first digital business ecosystem (DBE) [1] prototype distributed system as is described in the DBE architecture document [2][3]. This architecture details requirements of the prototype system and also use-cases that it must fulfil. The prototype is guided by this architectural document and paradigms such as the model driven architecture (MDA) and service-oriented architectures (SOA) are used as the basis of its implementation.

Due to WP 24 requiring “a huge effort in terms of software production and research” [4], this document also sets forth sound development processes to enable and achieve realistic software milestone goals. These processes were implemented so to release a stable product within the first eighteen months of the DBE project. Core to these processes are the concepts of test-driven development, continuous integration, and geographically distributed teams.

The goal of this 18-month architecture is to demonstrate how a small to medium enterprise (SME), in the role of a DBE service developer and/or provider, can create, publish and deploy a DBE service within the distributed ecosystem. Also, from this ecosystem, DBE consumers should be able to browse and consume any published and deployed DBE service. Not only were these two goals reached, but the final system delivered is a platform on which the outputs of other work packages could be validated.

2 Introduction

This document describes the implementation done on the DBE Service Factory, DBE Execution Environment, DBE Desktop, and the DBE Portal for the purpose of delivering a first implementation within 18 months of the project's time-frame.

This document is divided into four major chapters. The Chapter 3 details the background and adopted technologies for the DBE. The section on the development process introduces Extreme Programming with an emphasis on Test-Driven Development and Continuous Integration in order to provide a seamless and encouraging development process for the geographically distributed teams in DBE. Further, the Eclipse Plug-in Framework is presented as the basis for all rich client applications, such as the DBE Studio and the DBE Desktop. This framework provides a comprehensive and extensible set of APIs which can be used to implement platform-independent tools in Java to support the interaction of DBE actors with the DBE execution environment and to build and model DBE Services. Also, the Execution Containers are introduced which comprise the DBE Servent. It was important for DBE to use existing open-source projects as much as possible in order to minimise the work involved in building DBE from scratch. Eclipse, Tomcat, and Axis therefore are the main open-source projects used and they provide a solid basis to build DBE upon.

Chapter 4 covers the domain analysis for Service-Oriented Architectures with an emphasis on problems which may be posed to SMEs and their potentially low-cost network setup. The analysis details specific problems SMEs may encounter, such as firewall traversal, network address translation traversal, dynamic host allocation protocol leasing and service deployment, service publishing and availability. From TCD's point of view these problems have to be solved in order to provide a SOA that has minimal requirements on financial resources and that is not dependent on specific infrastructural requirements. Solutions to these problems are partly implemented in the current implementation of the DBE but the peer-to-peer architecture will address any remaining issues posed by these problems in the second half of the project's timeline.

Chapter 5 and 6 present the functional model of the DBE components, their design, and implementation. The requirements and architectural scope are extracted from the two deliverables WP 21.1 and WP 21.2 submitted by Soluta.Net.

This deliverable concludes with the achievements of the DBE 18 months implementation and the direction this implementation is going to lead during the second phase.

3 Background and Adopted Technologies

3.1 DBE Development Process

The first implementation of the DBE required a number of sound development processes in order to provide a suitable and feature complete (with respect to the architecture) DBE product within the time frame allocated to it. Also, to accommodate the possibility of rapid changing requirements and a highly evolving system the practise of Extreme Programming was chosen. Consequently, a number of build configuration and management decision were agreed upon, namely the use of the Maven [5] build system, the continuous integration environment Cruisecontrol [6], and the JUnit [7] testing framework.

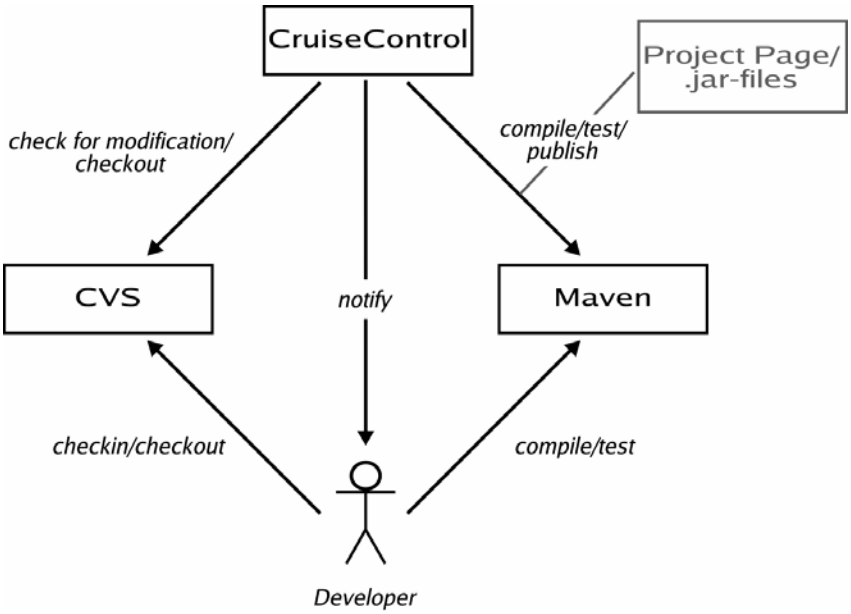


Figure 1 Development Environment

Figure 1 illustrates the development environment of this project from the perspective of a DBE developer. It follows open-source principles and best practices, where the source-code is maintained with CVS. The Maven build system was introduced to manage the whole project with all its individual sub-projects and components. Cruisecontrol is set up to trigger automatic builds every hour and every night. It notifies the developers, who were involved in the last modifications, when a build fails and thereby allows for an early resolution of any problems. The nightly build is responsible for publishing the web-site and .jar-file artifacts generated from each component.

Open-Source Project Development Processes

Often software development is practised in an disorganised fashion where the development cycle is dominated by code first, then fix. Usually in this case, the design of the system is determined by short-term decisions. While this process is working for a small team with a small product, it poses significant problems to both, the team and the product, as it becomes more complex. Therefore, development processes were introduced with the aim to impose a disciplined process on software

development in general and specific parts of it. Development methodologies provide a common understanding for a team on how to plan, implement, and interact to achieve a common goal. Mostly, they identify different roles for team members to optimise the outcome and to match expertise within the team to certain tasks. As a result, development becomes more predictable and thus provides a solid basis for the planning process in order to measure progress and set out the roadmap for future features in a product. Some of the traditional development methodologies put a strong emphasis on planning and are therefore very bureaucratic, document-centric, and require a single leader who guides the progress of the development efforts. In contrast agile development methodologies refer to light-weight processes which are more targeted to companies with flat hierarchies and especially the open-source community which commonly comprises of geographical distributed volunteers who work on these projects in their spare time, such as the Apache projects [8].

DBE adopted such a light-weight development process called Extreme Programming which is introduced in the following section. Also, this section details the open-source tools which have been adopted by DBE in order to support the development in a geographical dispersed team with individual goals and to provide a development platform that tackles the problem of continuously integrating individual development efforts to the general project.

Both, the development process and the development platform have been introduced to the development teams of DBE through a couple of meetings, including the meeting in July in Barcelona and a meeting organised by TCD covering tutorials on the build system, the source-control management system, and how TCD managed the continuous integration problem.

Extreme Programming

Extreme programming (XP) was developed by Kent Beck [9], a well-known figure in the object-oriented community. The emphasis of XP is on customer involvement and satisfaction in order to deliver a product that fulfills customer demands. Also XP promotes team work with an extended development team that includes developers, managers, and customers alike. XP identifies a lightweight development process that was created in response to dynamically changing problem domains and therefore require a methodology which allows for flexible planning and development cycles. In contrast to other existing development methodologies XP does not mandate the adoption of a fixed set of rules, but rather allows for team-specific requirements to be included in the development process as the team becomes more confident with individual best-practices. This makes XP suitable for a geographically dispersed team such as the DBE that consists of both academic and industrial partners. In particular DBE implements novel ideas and approaches which require a methodology that can adapt to changing requirements quickly.

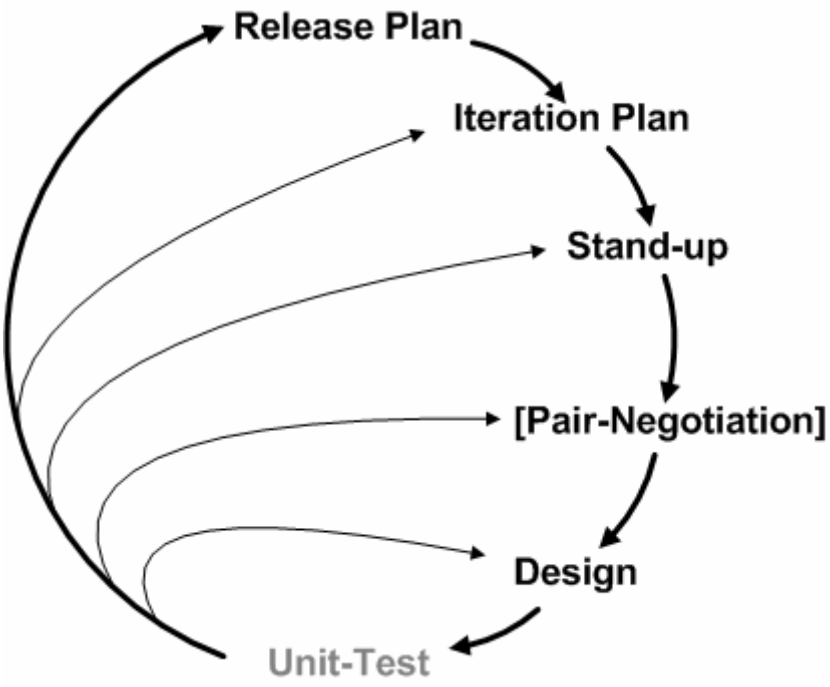


Figure 2 Extreme Programming Planning Loop

Figure 2 illustrates the XP planning cycle. The wish-list of customers and their priorities are set out in a road-map for a product. Each release is divided into iterations (usually two weeks cycles) which includes the user stories written by the team to satisfy the requirements of a particular product release. Daily stand-up meetings give each member of the team the possibility to discuss issues with their tasks and to negotiate pairs for the next task. Pair-programming is especially important for new team members to get to know the code-base and the general development process. Also, pair-programming drives the idea of common code ownership and spreads knowledge among the team members. The pair is responsible for the design of their task and usually adopts test-driven development where tests are written first and functionality is added bit by bit.

XP is based on the values of simplicity, communication, feedback, and courage which are described in greater detail below:

- *Simplicity*: Simplicity in design does not mean it is simple to design simplicity. A feature required by a customer is translated by the development team into “user stories” according to the “divide and conquer” paradigm. Each user story describes a self-contained task which can be implemented within a short time-frame, usually less than 2 weeks. As a result, small tasks are easier to design and implement. Development, driven by user stories, avoids the risk of designing unnecessary complexity or functionality into a product, because the goal of a task is clearly specified. For example optimisations are left until all features are implemented unless service-level agreements identify optimisations as a requirement for the product. By having a simple design code becomes more maintainable and for the developer easier to read and consequentially easier to refactor. Simplicity in extreme programming ultimately aims at minimising the cost of change over time as shown in Figure 3.

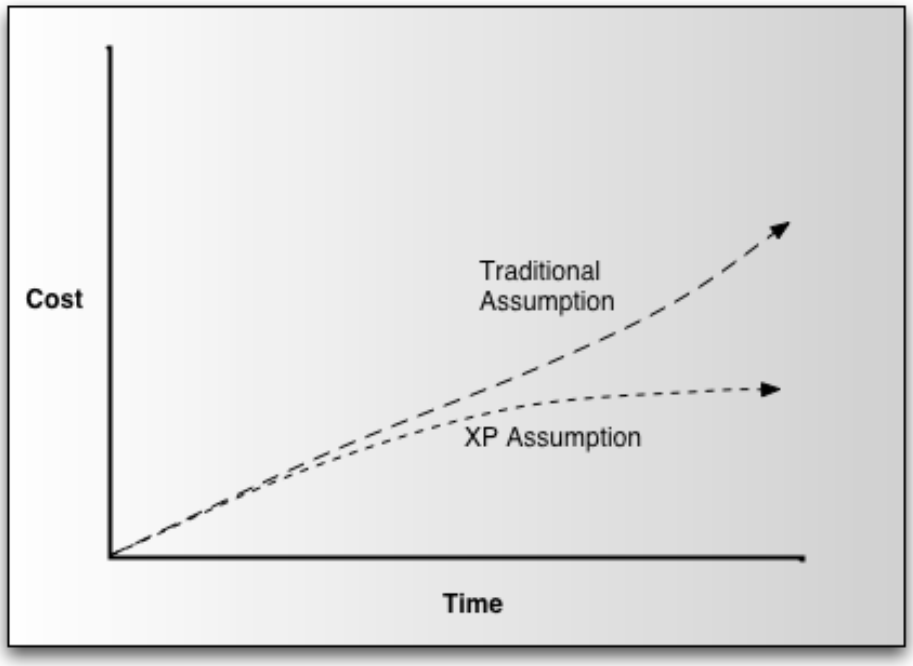


Figure 3 Cost of Change and Extreme Programming

- *Communication*: Constant communication between the customer, the manager, and the development team is important in order to implement a product which meets the demands of the customer(s) and to ensure a clear understanding of the requirements and features of a product. Within a development team communication takes place in “stand-up meetings” on (ideally) a daily basis where each team member explains what they did, intend to do, and where the problems were. Possible problems can be discussed with other team members in order to provide a quick solution. This adds a level of transparency to the team where everyone knows what others are doing. Another aspect of extreme programming is the idea of “pair programming”. With pair programming, two programmers sign up for a task of implementing a particular piece of functionality in a system where the responsibilities can be divided into two aspects. One is responsible for the typing and the tactical approach how to implement the method. The other one is responsible for the strategy of the development and how a method fits into the overall design. Consequently, by changing the pairs often knowledge is spread among the team members which avoids the risk of expertise leaving the team. Also, it promotes collective code ownership where everyone is allowed to make changes everywhere. XP favours flat hierarchies and low bureaucracy where communication can flourish and everyone's opinion plays an important role in the development process. DBE has adopted the approach of regular communication slots via chat sessions to encourage discussions of outstanding issues and progress of work.

- *Feedback*: Feedback cycles are very important for a development team in order to measure progress of the product. Feedback generally is available on different levels in a team. Business-relevant feedback cycles include the measurement of progress within the team, such as project velocity. Project velocity measures the time invested and how much work was accomplished in an iteration. The velocity of one iteration i(t) is an indication of how much work can be done in iteration i(t+1). Managers therefore have a means of adjusting

work units for future iterations. Also, feedback is available from different kind of tests on a technical level. Acceptance tests identify the functional requirements of the customers. Before a release is completed all acceptance tests must pass. Otherwise the product does not meet the demands of the customers. As a result the product might not be a best-seller. Other test categories, such as unit and component tests, provide feedback on a more fine-grained level. Unit tests are the implementation of requirements of a single class whereas component tests implement the required collaboration of two or more classes. The combination of all tests defend the requirements identified by a user story. Once all requirements are met (all tests pass) the implementation of a feature can be considered to be finished. Tests are usually automated and run at certain time-intervals or at a fixed time. Whenever a build is triggered the result is published on an internal web-site and in case of a failure notifications are sent to the team. Additionally, static source-code analysis tools provide feedback on the quality of the product. These tools include reports on the adherence of the code-conventions, code duplications, object-oriented metrics, code-coverage, etc.

- *Courage*: Courageous action embodies the requirement of acting early to dynamically changing problem areas, rather than reacting late. A decision made too late may give competitors an advantage with the potential risk of product failure. The opinion of each team member is important to identify possible problems with the current or emerging directions of a product.

TCD introduced a web-based tool called XPlanner [10] to the development community which allows the management of user stories of each development team's work packages. The workload of TCD's work packages have been divided into several user stories and development tasks which can be implemented within a manageable time-frame. A weekly internal development meeting was set up to discuss individual progress and problems during development and also to plan the inclusion of user stories into the next iterations. XPlanner supports planning decisions by providing project metrics, such as team velocity and individual hours.

In order to drive the common goal of DBE ahead, weekly Jabber meetings were set up to discuss integration points and problems.

The following sections describe two of the major and most important practises used in extreme programming, test-driven development and continuous integration and how it was achieved within DBE.

Test-driven Development

Test-driven development is core to extreme programming. It keeps the promises regarding the quality, cost, and existence of previously installed features of a product.

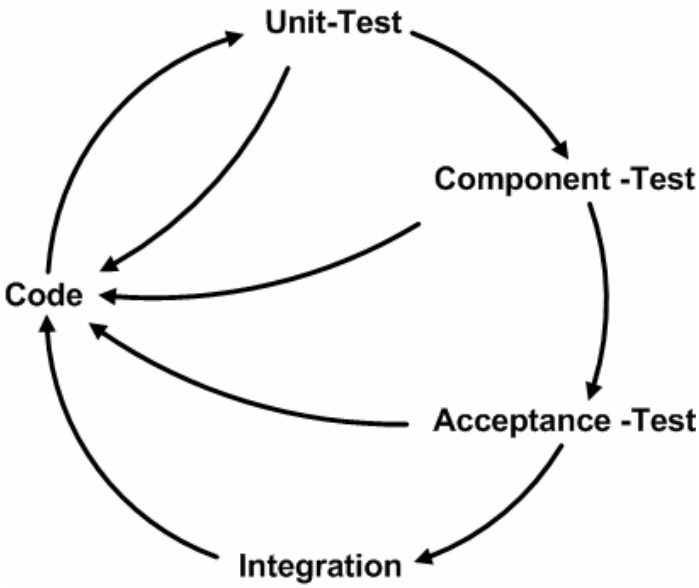


Figure 4 Extreme Programming Development Loop

In test-driven development (see Figure 4), the corner stone are tests which implement specific requirements of a single class, the collaboration of more than two classes, and the functional unit as a whole. A unit test is a test that only covers the functionality of one class and its contained methods. It does not test related classes of the class under test. Therefore the Unit-Test <-> Code cycle refers to the implementation of a single class. A Component-Test <-> Code cycle is aimed at implementing the required collaboration of two or more classes. Finally, the Acceptance-Test <-> Code cycle reports on whether the customer requirement of a certain feature has been met.

Tests provide a certain confidence level that the implemented features works as desired. They also provide a safety net for developers who are working on code that is not originally theirs when making changes. Once the desired feature is implemented and all the tests pass the code has to be integrated into the main development branch of the source-code management system.

The typical steps involved in test driven development are:

1. Write a test that specifies some functionality.
2. Ensure the test fails (the functionality is not yet implemented).
3. Write only the code necessary to make the test pass.
4. Refactor the code, ensuring that it has the simplest design possible for the functionality built to date.

When practicing test-driven development there are a number of rules that need to be followed. The first is that everything that can be broken should be tested. The second is that testing should always come first **followed** by implementation. Finally, before code is returned to a code repository all tests must pass.

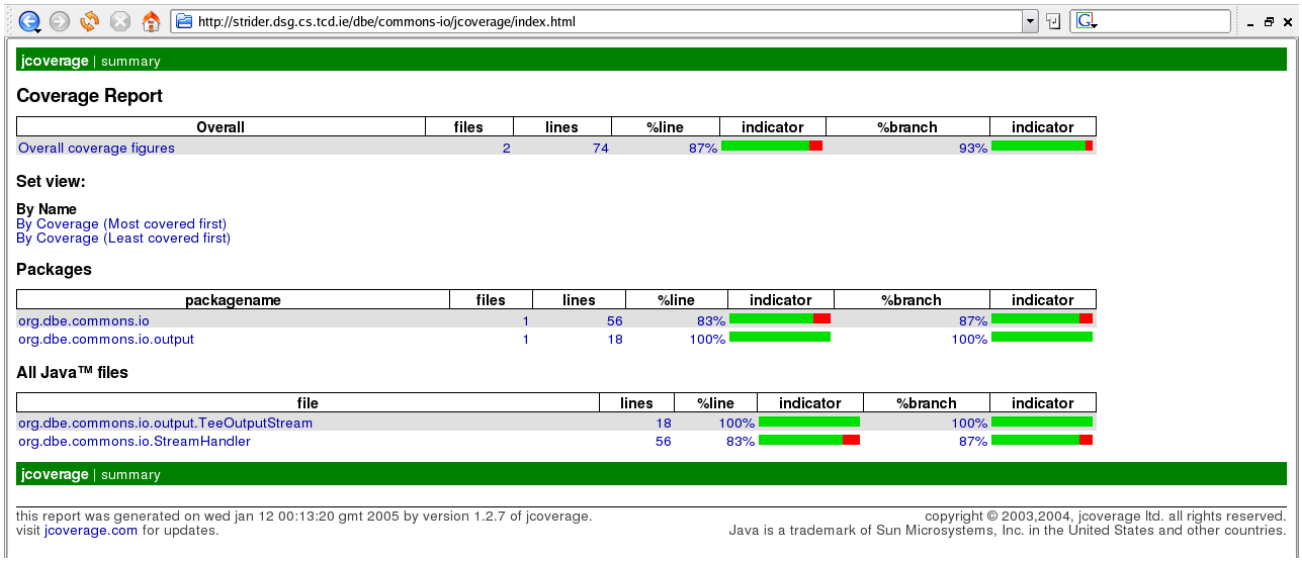


Figure 5 JCoverage Report

Figure 5 details a JCoverage report for the commons-io component of DBE showing the test coverage for two classes in two packages. That means that all tests implemented for this component cover 87 % of the source-code lines in average. This figure demonstrates that confidence in this component can be considered high, because the tests cover the desired behavior at implementation time. A bug in this component can be easily tracked down, because most likely a test is available which covers the scenario involving the bug.

There are major advantages to test-driven development apart from the confidence gained from knowing that a product passes all its tests. Other benefits include:

- Code is written so that classes are testable in isolation. Code written without tests in mind is often tightly coupled. Test-driven development promotes loosely coupled components and simple designs.

- The tests act as system-level documentation. They are the first clients of your classes and thereby illustrate developers the intended use of the class.

- Tests provide immediate feedback on the product. A common feedback cycle is the introduction of automated nightly test runs that publish their result in HTML and/or as email notifications in case of a failure. Consequently, the tests can be run on multiple platforms simultaneously without a developer being involved.

- Development is paced. Small discrete pieces of functionality are specified in tests, then code to fulfil that specification is written, where feedback is available withinseconds to a few minutes of coding. The code base progresses forward at a relatively constant rate in terms of the functionality supported.

Source Control Management for Geographical Distributed Partners

CVS

CVS [11] is a version control system, an important component for source configuration management (SCM). CVS provides the capabilities of recording changes to source files and as such records their histories. Similar examples of version control systems include RCS [12] (CVS is the successor to RCS), Clearcase [13], and Subversion [14]. CVS is a production quality system widely used in both commercial and open-source software projects. CVS does not follow a lock-modify-unlock model and does not provide mechanisms to do so. Instead, CVS approaches file management using a copy-modify-merge model. With this model a developer will download what is known as a *working copy* of the module he/she wants to work upon and directly edit this working copy. When the developer is finished with working on this copy, the developer then *commits* the working copy along with a log message detailing changes made. It is then up to CVS to merge the changes back into the master copy kept on the CVS server, known as the *mainline*. Although CVS stores individual file history in the same format as RCS, it offers the following significant advantages [15] over RCS:

- It can run scripts that you can supply to log CVS operations or enforce site-specific polices.

- It enables developers scattered by geography or slow modems to function as a single team. The version history is stored on a single central server and the client machines have a copy of all the files that the developers are working on. Therefore, the network between the client and the server must be up to perform CVS operations (such as *checkins* or *updates*) but need not be up to edit or manipulate the current versions of the files. Clients can perform all the same operations, which are available locally.

- In cases where several developers or teams want to each maintain their own version of the files, because of geography and/or policy, CVS's vendor branches can import a version from another team (even if they don't use CVS), and then CVS can merge the changes from the vendor branch with the latest files if that is what is desired.

- Unreserved checkouts, allowing more than one developer to work on the same files at the same time.

- CVS provides a flexible modules database that provides a symbolic mapping of names to components of a larger software distribution. It applies names to collections of directories and files. A single command can manipulate the entire collection.

DBE adopted CVS as the source control management system as part of the CollabNet [16] product which also covers project, knowledge, communication management solutions. Each DBE project is associated with a work package and has a dedicated source tree to reduce the risk of development conflicts and therefore allow for maximum parallel software development efforts.

Build Management

Given the extreme programming as the development methodology the build system has to support the execution of certain test cycles to maximise useful feedback for a particular component.

Additionally, the build system has to tackle challenges for DBE project management, documentation, and uniformity in the development process. It would not make sense for the DBE to maintain several different build systems, such as ANT, makefiles, etc., because it would hinder the release of a single open-source project with all its components. Early adopters of DBE who check out the latest version from CVS would find it hard to create a product from a source-tree.

Maven [5] was chosen as the build management tool, because it can provide centralised build functionality for all components and thus supports the installation of a common development platform to support test-driven development and continuous integration in a coherent and comprehensive way.

Maven

Maven is a project management tool, which is based on metadata information. Metadata is captured in a project object model (POM). The advantage of metadata information over actual specific information is flexibility and code-reuse. All goals in Maven operate on the metadata information (e.g. the Java source location and the Java classes destination). As a result, goals are packaged in reusable plug-ins for Maven and are made available to all Maven projects. In contrast, ANT [17] requires the definition of specific targets (init, javac, jar, etc).

ANT is an established tool, which has its place in the development community. However, ANT has some limitations which decrease the maintainability of a build system. A developer is always interested in focusing on main tasks, where maintaining a build system is not a main part thereof. The main limitations of ANT include:

- Re-usability: ANT has no notion of sharing build scripts and targets across projects. Each project requires a full-blown build.xml file, which contains all targets that operate on the current project. Smart developers already came up with more abstract definitions of the most common targets, such as clean, init, compile, and jar, and made them available to all projects. These targets can be referenced in the build.xml file. In the software industry this is not a new paradigm and has been followed successfully on a component-level but not in build systems.

- Extensibility: As a result of the poor re-usability in ANT, ANT is not easily extensible. It provides a useful Java API to implement specific tasks. On a scripting level extensibility is not provided.

- Programming Logic: Traditionally, ANT was never meant to be a tool to provide scripting functionality. It is possible however, but it is awkward.

- Dependency Management: ANT does not force a dependency scheme on to the developer. Hence, it is up to the developer to manage his dependency list and keep them up to date. The lack of such a scheme leads to a specific third library repository in the code base in order to provide a consistent view on these libraries for all developers.

In contrast to the ANT approach to write the build script and run the targets, Maven follows a different pattern. The developer needs to describe the project and configure the plug-ins once and runs the existing plug-ins.

The metadata is kept in a project.xml, which adheres to a XML schema specified by the Maven community. Existing Maven plug-ins are configured in the project.properties file. So, Maven's approach is to recommend developers how to layout the project/component structure where all

goals are shared and the order of execution well known. All generated files are in a well-known directory PROJECT_DIR/target. The project.xml file mainly consists of three parts:

1. The project management section includes general information about the project such as the name, id, organization, version, etc.

2. The project's dependency section describes the dependencies in form of a group id, artifacts, and a version number. Maven's internal mechanism to retrieve these dependencies is based on a global repository. This has the advantage that the developer is not involved in keeping the dependencies at all. At runtime Maven will first attempt to resolve the dependencies in the directory .maven/repository in the developer's home directory. If it can resolve the dependency <groupId>/jars/<artifactId>-<version>.jar it will use this one to set up the class path. Otherwise, a global repository is contacted to download the dependency from there and store it in the aforementioned repository of the developer's home directory

3. The project's build and report section describes where to find the sources and the test files. Also, a developer can register reports such as javadoc, object-oriented metrics, and coverage report to be included in the project's generated web page.

Maven and Ant Equivalents

	MAVEN	***ANT***
Standard build files	project.xml, maven.xml	build.xml
Properties process order	1. ${maven.home}/bin/driver.properties 2. ${project.home}/project.properties 3. ${project.home}/build.properties 4. ${user.home}/build.properties 5. System properties defined by -D command line option The *last* definition wins.	1. System properties defined by -D command line option 2. Properties loaded by the `<property>` task The *first* definition wins.
Build rules	Build rules are more dynamic (similar to a programming language); they are executable XML based on Jelly.	Build rules are more or less static (unless you use the `<script>` task).
Extension language	Plug-ins are written in Jelly (XML).	Plug-ins are written in the Java language.
Build rules extensibility	Build goals are extensible by defining. `<preGoal>` and `<postGoal>` .	

Frequently Used Goals in Maven

Goal	*Description*
clean	deletes the target directory of the project
java:compile	compile the .java files
jar	jar up all .class files and specified resources such as properties.
jar:install	install the jar file into the local repository in order to make it public to other projects/components.
multiproject:goal -Dgoal=java:compile	compile a project which consists of several components. MAVEN makes sure that the components are compiled according to their dependencies. The **-Dgoal** option can take any existing maven goal as a value.
site	The site goal generates the MAVEN-specific web page including all registered reports.

Maven is a feasable project management tool which does not replace ANT, but rather it complements ANT's functionality and provides solutions for ANT's difficiencies. All ant targets are availabe in maven as well and can be encorporated in Maven's build files or in a plugin. The advantages of Maven is code-reuse, flexibility, extensibility, and more advanced scripting mechansisms with the Jelly processing language. Also, Maven is capable of generating a web-page for the complete project and its individual components that contains developer-specific information on the quality of the implementation as well as documentation for a user of the project. User-level documentation can be hooked into the default web-page with the help of XML files.

Once set up, the Maven build system is far easier to maintain than any project based on an ANT build system. Maven encourages thinking in components. Projects can be split up into different components easily while maintaining their relationship. Before developers add functionality to a project, it is very important to find out the best place to put it. Hence Maven's full potential comes to shine, when a well-thought project layout is in place. The proposed project layout for DBE is summarised below in the following sections.

Project Layout

A proper project layout adds a lot of value to a project in terms of maintainability and visibility to developers to find components. Maven's potential is based on a good project layout. This chapter attempts to identify commonalities in all projects and proposes a layout not only for a single component but also for a whole project consisting of sub-projects, and components.

There are other possibilities to structure the project, e.g., to provide separate directories for a particular milestone (18 months demonstrator). This has the disadvantage that source-code is discontinued after a milestone is reached. In contrast, CVS allows to label a project, which can be used to apply a label to a snapshot of a product at a certain time. As a result, code does not need to be moved around.

Top-Level Layout

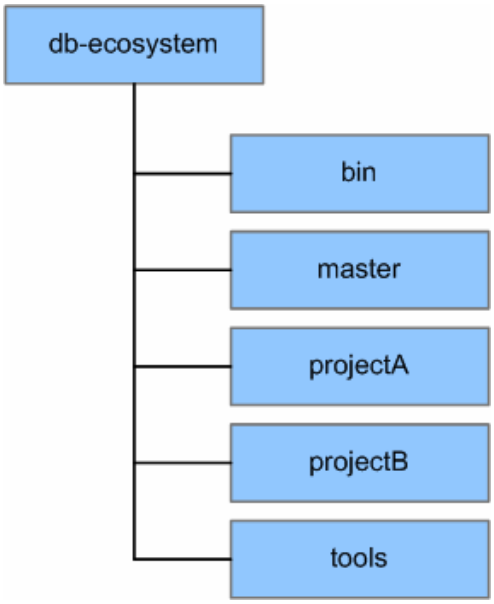


Figure 6 Top-Level Layout

The db-ecosystem is the top-level directory which contains sub-projects, such as the knowledge-base, composer, common, and testutils. These sub-projects may resemble the work-packages in the work plan (see Figure 6). Hence there should be no overlap of responsibilities in the projects. However, some implementations can be shared in the common project, such as configuration and UI frameworks. The bin directory contains helper scripts to set the environment for the developer. This environment script sets variables for Maven and Java. A Maven distribution with customized plug-ins will be added to the CVS repository in the tools section. Consequently, the developer does not need to download additional tools in order to be able to build the DBE projects. This approach ensures that everyone in the team is working with the same environment. The master directory contains global settings for Maven and e.g. log4j.properties, checkstyle.xml, etc. which are common to all projects. It is the driver for the overall build, jar, test, etc.

Sub-Project Layout

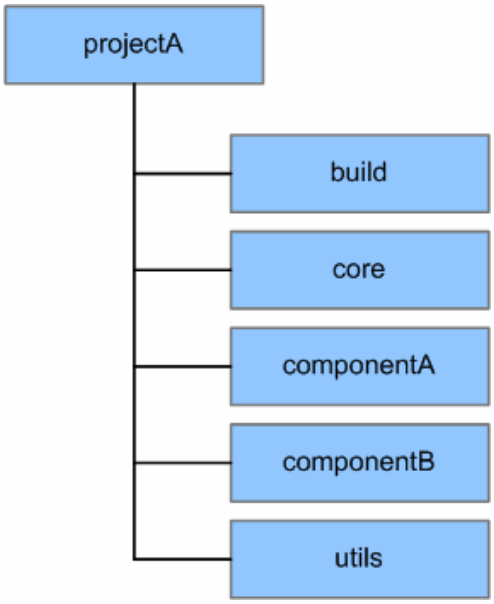


Figure 7 Sub-Project Layout

Figure 7 illustrates the sub-project layout which usually consists of at least three components. The core component contains the core part of the sub-project (e.g. the micro-kernel architecture of the project), x numbers of components (plug-ins for the micro-kernel), a util component which contains helper classes for this project. The goal is to have little components which isolate a certain functionality. In most cases complex components are too big and could be much easier to maintain if they are partitioned into little "independent" components. The build directory contains the configuration for a continuous integration tool (for DBE Cruisecontrol). The main folder contains all classes which implement the `main(String[] args)` method and functional or system tests for the project.

Component Layout

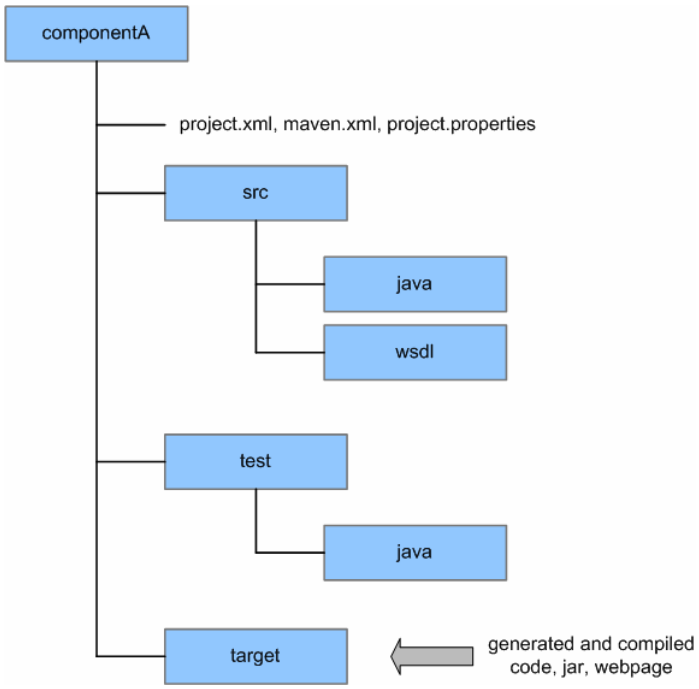


Figure 8 Component Layout

Figure 8 shows the two most important parts of a component, the src and the test folder. The src folder contains all the sources organized in sub-folders depending on the implementing language. E.g. .java files are in the java sub-folder and .wsdl files are in the wsdl sub-folder. The separation of the sources allows specialised Maven plug-ins to operate appropriately on each "language package". For example a java-specific Maven plug-in implements how to compile .java-files and set up a classpath based on the dependencies declared in the POM.

Continuous Integration

Common best practice is to integrate as often as possible whenever a piece of functionality has passed the test cycles. If code is held on to for too long the changes made can lead to integration problems. Continuous integration relies on three factors in order to be useful:

- Branching off the mainline in the source repository. Every development task should have a dedicated development branch in CVS to maintain independence from ongoing parallel developments.
    - Test coverage: It is impossible to integrate a piece of work without the feedback from various test runs.
    - Automation: Getting the latest code, compiling it, and test it must all be automated.

The fully automated build system is the most valuable one that allows a development team to build and test their software many times a day. The implementation of a fully automated build and test system requires the following characteristics:

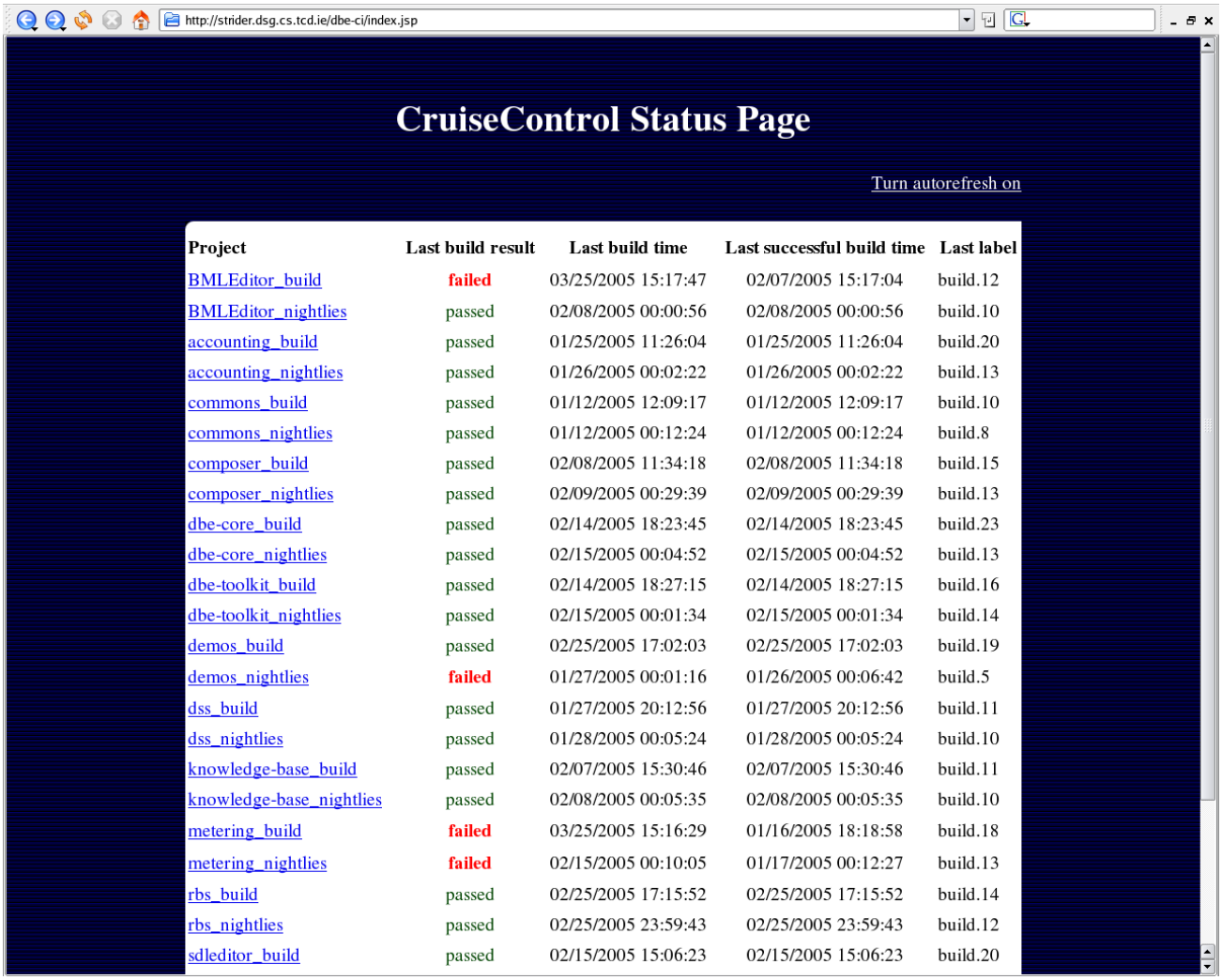
- A central source code repository where any developer can access and obtain the most up to date source code.
- Automate the build process of the code contained in the central source code repository so anyone can issue a single command to build the system from source.
- Automate the testing process so all or a selection of tests can be ran over the system at any time with a single command.
- Where an integration problem arises, the developers involved are notified as soon as is feasible.

The fundamental benefit of continuous integration is that it removes sessions where people spend time searching for bugs where one developer's work has resulted in conflicts in another developer's work without either realising. These bugs are hard to find because the problem is not in one person's area, it is in the interaction between two pieces of development. This problem is further exacerbated by time. Often integration bugs can be inserted weeks or months before they first manifest themselves. As a result they take a lot of finding. With continuous integration, the vast majority of such bugs manifest themselves the same day they were introduced and as such the search scope for the bug is dramatically narrowed. A large caveat of continuous integration is that it is highly dependant testing. Unfortunately testing does not prove the absence of errors but does increase the confidence that what is tested works. Overall, the result of continuous integration is increased productivity by reducing the time spent solving integration issues.

Cruisecontrol

Cruisecontrol is a tool which supports a development team in running continuous builds. Builds not only include compiling the latest sources from scratch, but also running all available tests. It is an open source project hosted at http://cruisecontrol.sourceforge.net. Cruisecontrol supports project management in DBE by producing status reports on the different projects in DBE.

Cruisecontrol is configured to retrieve the latest sources from the DBE source code repository, build each of the components from scratch, and runs any available tests against them. If there are any failures an email notification is sent out to the team members who were involved in the latest modifications to draw their attention to the failing build. Cruisecontrol provides a web-interface (see Figure 9) to display the status of each component. It shows the failures relative to the total number of tests and allows to navigate to the build output and to metrics of the builds for a particular component.



Project	Last build result	Last build time	Last successful build time	Last label
BMLEditor_build	failed	03/25/2005 15:17:47	02/07/2005 15:17:04	build.12
BMLEditor_nightlies	passed	02/08/2005 00:00:56	02/08/2005 00:00:56	build.10
accounting_build	passed	01/25/2005 11:26:04	01/25/2005 11:26:04	build.20
accounting_nightlies	passed	01/26/2005 00:02:22	01/26/2005 00:02:22	build.13
commons_build	passed	01/12/2005 12:09:17	01/12/2005 12:09:17	build.10
commons_nightlies	passed	01/12/2005 00:12:24	01/12/2005 00:12:24	build.8
composer_build	passed	02/08/2005 11:34:18	02/08/2005 11:34:18	build.15
composer_nightlies	passed	02/09/2005 00:29:39	02/09/2005 00:29:39	build.13
dbe-core_build	passed	02/14/2005 18:23:45	02/14/2005 18:23:45	build.23
dbe-core_nightlies	passed	02/15/2005 00:04:52	02/15/2005 00:04:52	build.13
dbe-toolkit_build	passed	02/14/2005 18:27:15	02/14/2005 18:27:15	build.16
dbe-toolkit_nightlies	passed	02/15/2005 00:01:34	02/15/2005 00:01:34	build.14
demos_build	passed	02/25/2005 17:02:03	02/25/2005 17:02:03	build.19
demos_nightlies	failed	01/27/2005 00:01:16	01/26/2005 00:06:42	build.5
dss_build	passed	01/27/2005 20:12:56	01/27/2005 20:12:56	build.11
dss_nightlies	passed	01/28/2005 00:05:24	01/28/2005 00:05:24	build.10
knowledge-base_build	passed	02/07/2005 15:30:46	02/07/2005 15:30:46	build.11
knowledge-base_nightlies	passed	02/08/2005 00:05:35	02/08/2005 00:05:35	build.10
metering_build	failed	03/25/2005 15:16:29	01/16/2005 18:18:58	build.18
metering_nightlies	failed	02/15/2005 00:10:05	01/17/2005 00:12:27	build.13
rbs_build	passed	02/25/2005 17:15:52	02/25/2005 17:15:52	build.14
rbs_nightlies	passed	02/25/2005 23:59:43	02/25/2005 23:59:43	build.12
sdleditor_build	passed	02/15/2005 15:06:23	02/15/2005 15:06:23	build.20

Figure 9 Cruisecontrol Status Page

Another way of using Cruisecontrol is to run regular builds, which are triggered every 30 minutes or 1 hour. It can be configured to only trigger a build process, if there have been modifications on the mainline since the last build. This gives a team immediate feedback on all their integration work. Additionally, a label can be applied automatically to the code base, if the build process completes successfully. However, this only makes sense, if the coverage rate of the tests is high (more than 80%). Otherwise, it wouldn't be possible to determine (automatically) whether a product reached a stable point.

Coding Standards

Most of the time of developers is devoted to maintaining existing applications written by different people. Analogous to speaking the same language in a team meeting it makes sense to have common code conventions for implementations as well in order to provide consistency and a common layout of language constructs. Sun Microsystems already specified code conventions [18] for the JAVA language. Also, Doug Lea provides an extended version of the JAVA Coding Standards [19], which include further recommendations based on the book in [20].

3.2 Eclipse Plug-in Framework

The challenge for DBE is to present the DBE actors, such as consumers, service providers, software developers, and companies and individuals external to the DBE, with a uniform and transparent way of interacting with DBE services. The user interface should support multiple platforms, because assumptions on the DBE users' hardware and software installation cannot be made.

It was decided to use the eclipse framework to provide an integrated environment that leverages the DBE infrastructure in a consistent fashion. Eclipse plug-ins can be used to implement DBE services can be implemented in such a way that no assumptions on the underlying hardware and software environment are made.

Eclipse is an open-source flexible tool platform or tool base. Unlike other open-source projects that do not allow proprietary derivative works, Eclipse can be extended with proprietary plug-ins and repackaged. It is flexible by means of an innovative plug-in architecture allowing unlimited extensions to the base IDE. These plug-ins can do anything from syntax highlighting to interfacing with a source code control system, to building fully featured applications.

The Eclipse platform itself is made up of several major components:

- Platform runtime,
- Rich client platform (RCP),
- Workspace,
- Workbench,
- Standard widget toolkit (SWT),
- Version and configuration management (VCM),
- Help system.

The Eclipse platform runtime is essentially a micro-kernel architecture. All Eclipse functionality is supplied by plug-ins. The Eclipse Platform Run-Time is restricted to handling start-up, discovering plug-ins installed on disk, matching up extensions with extension points, building a global plug-in repository and caching the registry on disk for the next start-up.

Eclipse 3.0 introduces a rich client platform (RCP), which allows developers to use a subset of the Eclipse development tool platform to build applications that are not integrated development environments, such as business applications with a graphical user interface. RCP significantly reduces the amount of code a developer needs to write to build platform-independent user interfaces. Internally, eclipse is implemented as a Model-View-Controller architecture with respect to hiding the platform-specific view behind an extensible API so that any plug-in of eclipse using this API is supported on many available platforms. Therefore, the development of Eclipse-based tools and wizards can be used by various SMEs with different hardware and software requirements. Eclipse has built-in support for Linux, AIX, HP-UX, Solaris 8, Mac OSX, and Windows on different hardware architectures (x86, AMD 64, IA 64, Sparc, PPC and HP9000).

The workspace presents a platform-independent view of the file system. It manages resources such as low-level change tracking and virtual symbolic links (markers). The workspace's abstractions for the file system are exposed to plug-ins, so they can write platform independent file system code.

Eclipse's user interface is known as the workbench and consists of one or more main windows that display a collection of perspectives. Perspectives can be viewed as a composite of workbench parts. Workbench parts can either be a view or an editor. An editor is the component that allows creation and modification of source files (e.g. XML, Java, C++ source files) and a view is a supporting component that provides additional information about the source file that is currently being edited.

SWT [21] is IBM's version of Java's Swing class libraries, however SWT has a closer mapping to the native platform's Windowing toolkit and performs better compared to any Java-based UI. This limits the portability of SWT, but implementations are available for most major platforms (as aforementioned).

Development Tool Support

The Eclipse community provides many tools that can be leveraged by the DBE to support the DBE Service Factory.

Eclipse provides a GUI for CVS. By default, Eclipse's VCM plug-in is implemented for use with the concurrent versioning system (CVS). VCM can be extended to support additional source code control systems. However, since the VCM abstracts the underlying source code control system, the user interface doesn't change — regardless of the source code control system used.

The help component enables plug-ins to supply HTML documentation that can't be presented contextually from the workbench.

Eclipse also provides some useful plug-ins: the debugger, ANT [17] and compare. ANT is an open-source build tool that is part of Apache's Jakarta project. Eclipse provides a graphical front-end to ANT, which allows easy execution of any option defined in the project's ANT build file.

The Eclipse platform also adopts an internationalized implementation, where translations live in the plug-in fragments and can be separately shipped. Translations are available for many European languages.

Plug-in Development in Eclipse

The plug-in development environment is the primary mechanism for providing new functionality to an Eclipse application. It consists of four main parts, see Figure 10:

PDE (Plug-in Development Environment)

- JDT (Java Development Tools)
- Eclipse Platform
- JDK

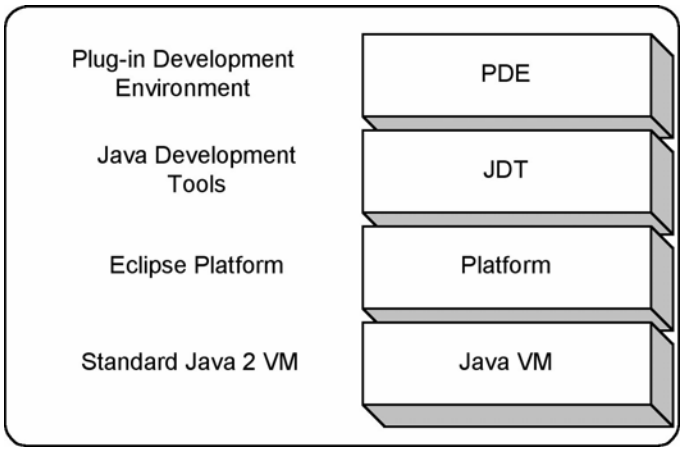


Figure 10 Plug-in Development Environment

Each plug-in contributes to one or more extensions points. Plug-ins can optionally declare new extension points and may depend on a set of other plug-ins. All details, including contribution declarations, can be found in the plug-in manifest.

JDT (Java Development Tools) is aimed to be a state of the art Java development environment. It is built on top of the Eclipse platform and implemented as plug-ins. This environment includes advanced editors, compliers and debugging functionality, and is included in eclipse project releases.

PDE (Plug-in Development Environment) is built on top of the eclipse platform and JDT, providing specialized tools for the development of Eclipse plug-ins. Similarly with the JDT, the PDE is implemented as plug-ins and is included in project releases. It also provides the functionality to generate Ant build scripts.

An example pair of plug-ins can be seen in Figure 11:

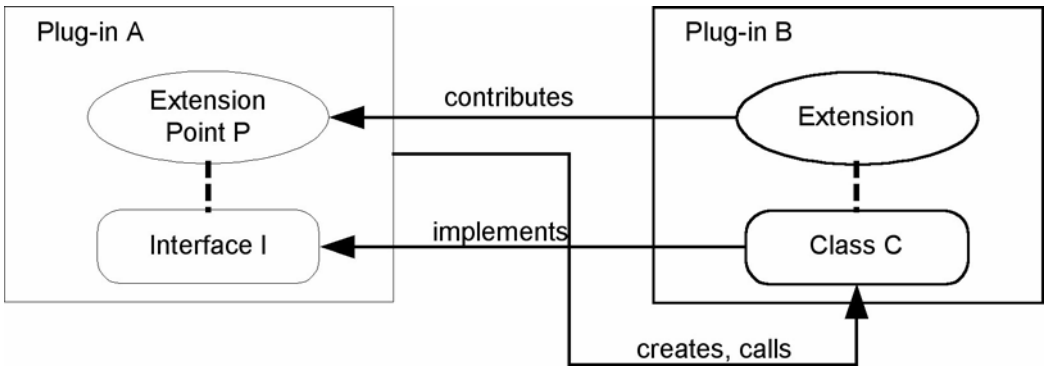


Figure 11 Eclipse Plug-ins

- Plug-in A
    - Declares extension point P
    - Declares interface I to go with P
- Plug-in B
    - Implements interface I with its own class C

    - Contributes class C to extension point P
- Plug-in A instantiates C and calls its I methods

Features: Downloading Plug-ins from a web site

The Eclipse Platform Update Manager allows the downloading of plug-ins from a website. This allows the replacement of plug-ins with newer versions and the installation of new plug-ins.

Features are designed to help in performing the three major tasks:

1. Install new functionality into Eclipse products
2. Update versions of existing plug-ins with newer versions. This task allows you to receive fixes and upgrades to the functionality you already have.
3. Control which plug-ins should run when Eclipse starts. This task allows you to disable an entire group of plug-ins, thus making them invisible to Eclipse run-time.

As plug-ins have plug-in manifests, features have feature manifests. The file name is feature.xml and it can contain the following elements:

- Feature id, version, provider name, image to use in Eclipse
- Short description, URL to long description, license agreement, copyright statement
- References to other features (for building complex feature trees by inclusion)
- Dependencies (features can require presence of other plug-ins or features as a prerequisite for installation)
- List of plug-ins and fragments that belong to the feature
- List of data archives that belong to the feature (normally go together with the custom install handler)

Features can be pre-installed or packaged on a server for updates (ready-to-download features). Features are downloadable from a HTTP server as a JAR file.

3.3 Execution Container

DBE requires an application server to provide a platform-independent execution environment for DBE Services. There are many open-source application servers that could have been leveraged and adopted by DBE to fulfil the role on DBE application server.

Tomcat

Tomcat [http://jakarta.apache.org/tomcat/] is a web server/servlet container that supports servlets and JSPs. Integrated within Tomcat is the Jasper compiler that compiles JSPs into servlets. It implements the Servlet 2.4 specification and the JSP 2.0 specification. Architecturally, it is composed of the following components:

- *Server*: A Server represents the whole container. Tomcat provides a default implementation of the Server interface. This is rarely customised by users.
- Service: A Service is an intermediate component which lives inside a Server and ties one or more Connectors to exactly one Engine.
- *Engine*: An Engine represents request processing pipeline for a specific Service. As a Service may have multiple Connectors, the Engine received and processes all requests from these connectors, handing the response back to the appropriate connector for transmission to the client.
- *Host*: A Host is an association of a network name, e.g. www.host.com, to the Tomcat server. An Engine may contain multiple hosts, and the Host element also supports network aliases such as host.com and abc.host.com.
- *Connector*: A Connector handles communications with the client. There are multiple connectors available with Tomcat, all of which implement the Connector interface. These include the Coyote connector which is used for most HTTP traffic, especially when running Tomcat as a standalone server, and the JK2 connector which implements the AJP protocol used when connecting Tomcat to an Apache HTTPD server.
- *Context*: A Context represents a web application. A Host may contain multiple contexts, each with a unique path. The Context interface may be implemented to create custom Contexts, but this is rarely the case because the StandardContext provides significant additional functionality.

Jetty

Jetty [22] is a Java HTTP server / Servlet container. Jetty is a fully featured web server for static and dynamic content. Jetty's HTTP server and web applications run in the same process so reducing the interconnection overheads and complications. Jetty has a small payload, typically around 350KB. It can scale to thousands of simultaneous connections. Jetty is the Servlet container of choice for the second half of the DBE implementation.

Axis

Axis (**A**pache E**X**tensible **I**nteraction **S**ystem) [23] is a Java SOAP engine that processes SOAP messages. It presents itself to both Tomcat and Jetty as a web application and runs within the respective container. It has extensive support for WSDL, version 1.1, and has emitter tools for generating Java stubs and skeletons from WSDL. Axis is based around configurable "chains" of message "handlers" which implement small pieces of functionality in a flexible and composable manner that operate upon messages.

When the central Axis processing logic runs, a series of handlers are each invoked in order. The particular order is determined by two factors - deployment configuration and whether the engine is a client or a server. Users can develop custom handlers to provide specific functionality and message processing and plug these handlers into axis's chains of handlers. The object which is passed to each Handler invocation is a MessageContext. A MessageContext is a structure which contains several parts:

1. A "request" message
2. A "response" message
3. A bag of properties

There are two basic ways in which Axis is invoked:

1. *As a server,* a Transport Listener will create a MessageContext and invoke the Axis processing framework (see Figure 12).
2. *As a client*, application code will generate a MessageContext and invoke the Axis processing framework (see Figure 13).

The Axis framework's job is simply to pass the resulting MessageContext through the configured set of Handlers, each of which has an opportunity to do whatever it is designed to do with the MessageContext. For example a handler could employ a logging facility in order to visualise the message flow. The client and server side message paths are shown in the following diagrams. The small cylinders represent Handlers and the larger, enclosing cylinders represent chains (ordered collections of Handlers).

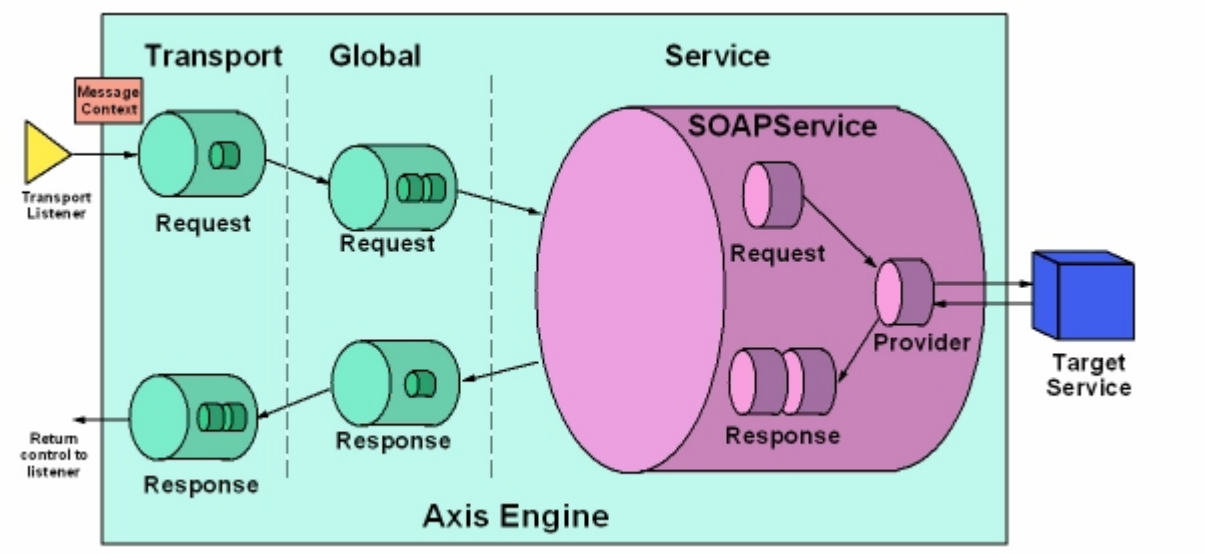


Figure 12 Server-side Message Path

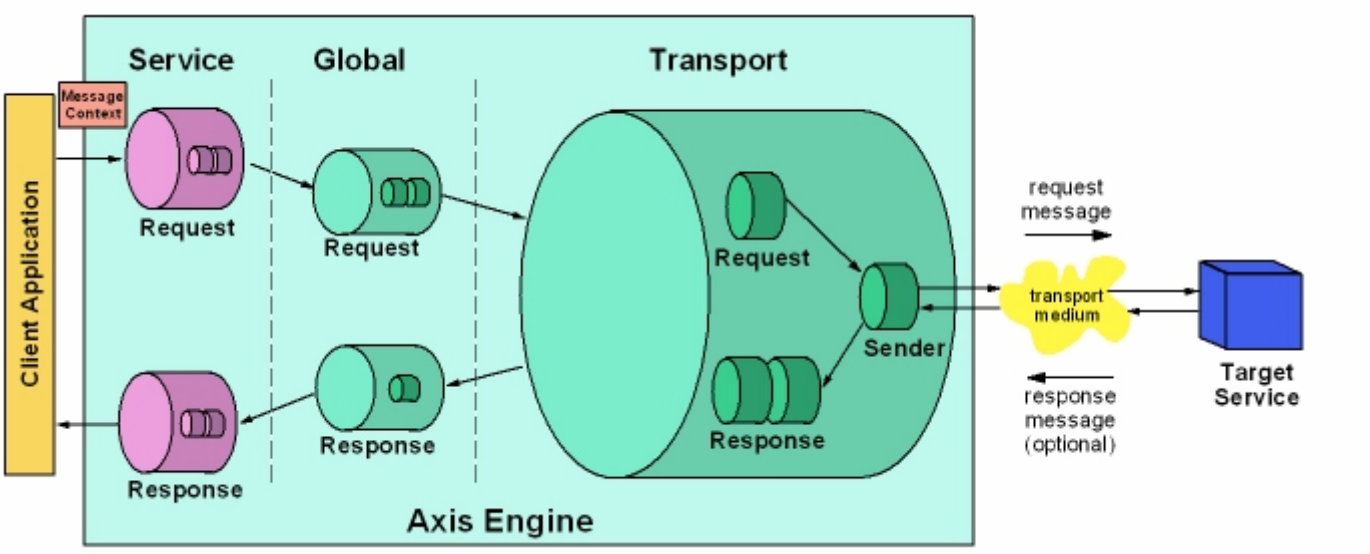


Figure 13 Client-side Message Path

3.4 Conclusions

This chapter presented the background and adopted technologies of the DBE.

Extreme Programming as the development process adopted in DBE was introduced to base the planning and development cycles on a flexible and light-weight process most suitable for geographically dispersed. TCD introduced a common build system to support test-driven development and continuous integration of development efforts.

The eclipse plug-in framework was covered to address the challenges of heterogeneous software and hardware systems of SME. Eclipse provides a platform-independent and extensible set of APIs to allow the implementation of DBE plug-ins that facilitate the DBE infrastructure. The plug-ins cover editors and views for the Service Factory in order to model and develop a DBE Service. Also, the plug-ins include tools to interact with the DBE Execution Environment to browse existing services and deploy new ones.

The section on the Execution Container presented the main servent components that are used for the DBE Execution Environment. These components include Tomcat as the servlet container and Axis to process Web Services. It is anticipated that the servent is going to be reimplemented by SUN Microsystems to provide a more feature rich environment.

4 Domain Analysis

For the 18-month implementation of the DBE, TCD has developed a service-oriented architecture (SOA) that is adapted to typical deployment environments found in small-to-medium enterprises (SME) [3]. This architecture will allow SMEs to deploy their services rapidly, cost effectively and with minimal resources, both technical and financial, when combined with the model-driven architecture (MDA) [24] as mandated by the Architectural Scope Document (WP21) [2].

4.1 Service Oriented Architecture

"A Service-Oriented Architecture (SOA) is a component model that inter-relates the different functional units of an application, called services, through well-defined interfaces and contracts between these services." [25]. The following diagram, Figure 14, summarises the SOA approach in a generic fashion. Services are:

1. Published by service providers in some type of service registry.
2. Discovered in service by service consumers.
3. Used by service consumers.

Along with the above definition of SOA a description of a service is also needed. When referring to a service, we define it to be "a collection of protocols and standards used for exchanging data between applications." From this definition we can draw out a definition of many concrete service. For example, a CORBA [26] service would be a service with IIOP over a connection-oriented transport as its protocol and the standards used are those set forth by the object management group (OMG) [27]. Similarly and with relevance to the 18-month implementation of the DBE a web service is a service with SOAP [28] as its protocol and the standards used are those provided by OASIS [29], W3C [30], and WS-I [31] standards bodies. It should be noted that SOA is an *architecture* and not a particular implementation of some concrete technology. As such it is feasible and very possible to imagine a SOA-styled system allowing the communication between technologies such as DCOM [32], CORBA [26], and web services. Also, it is viable to imagine a service registry that is not centralised but distributed among SMEs in a peer-to-peer (P2P) network within the DBE. These variations invalidate the common misnomer that SOAs are implemented with only web service technologies, such as WSDL, SOAP, HTTP and UDDI.

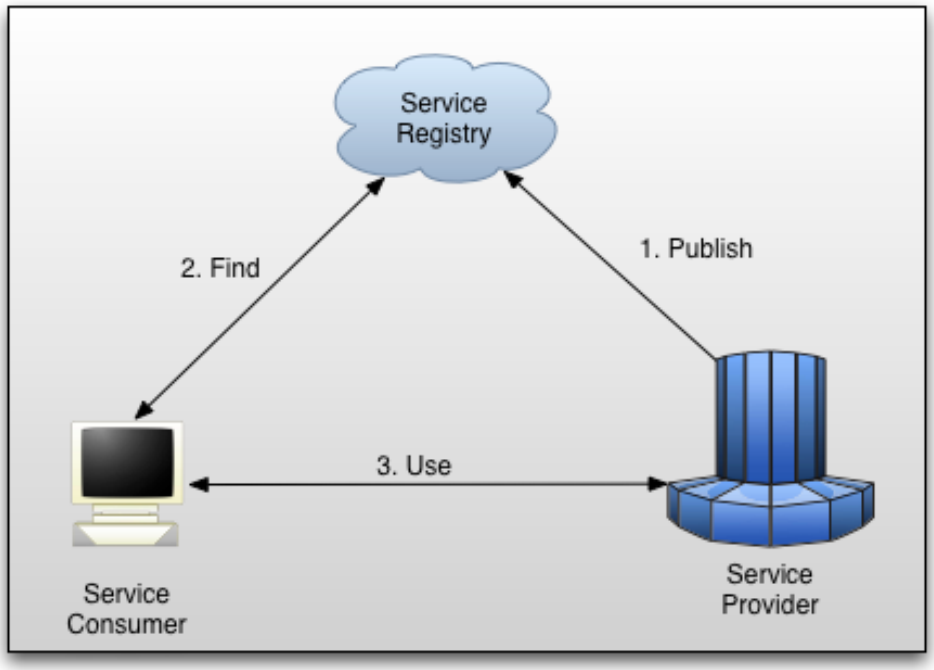


Figure 14 Generic SOA Model

For a system's architecture to be classed as a SOA the major attribute that they should exhibit is loose coupling of service consumers and service providers.

Loose coupling is achieved by designing the interfaces to the services as independent of the hardware platform, the operating system and the programming language that the service is implemented in. As such SOA interfaces need to be implementation neutral. This allows services from heterogeneous systems interact with each other easily and uniformly. Loose coupling gives a service and/or system an elasticity that enables it to survive evolutionary changes in the structure and implementation of the service. It also enables a high-level contractual agreement between services and minimises platform-level dependencies between those components.

Another desirable feature of middleware and an often-noted attribute of SOA is the enabling of flexible configuration. A SOA is best configured late and flexibly. This allows late binding between services, which enables consumers change the configuration of the target service dynamically. This is illustrated in Figure 14 where the service consumer performs a lookup for the required service before invoking the service. This allows changes to be made to the service without the notification of dependent consumers of the service about those changes.

Finally, for successful SOAs to be realised, services within a SOA should be designed with coarse granularity in mind. What is important about coarse granularity is that it "reduces dependencies between services and reduces communications to fewer messages of greater significance" [33].

Although SOAs are not necessarily implemented with web service technologies, this has been the most common and visible architecture implementation to date. For a typical web service model of a SOA is based on the following technologies:

- *UDDI*: Universal Description, Discovery, and Integration server is a platform neutral, XML based registry for web service providers to advertise and list their services. The UDDI specification is standardised by the OASIS group [29]. UDDI enables service consumers to query its registry using SOAP requests, which for the common case will return the web service description language, web service description language (WSDL) [34], interface of the requested service.

- *WSDL*: The web service description language is used to specify the interfaces to web services. It is a platform and language independent language, specified by W3C. Mappings exist for languages such as Java, C++, python etc. Of particular note, it is within a WSDL description, is binding information related to the defined service.

- *SOAP*: The Simple Object Access Protocol (SOAP) is a [28] protocol for exchanging XML based messages and provides the basic framework for web services to operate. The SOAP specification is issued by the W3C. There are a number of messaging patterns that SOAP can accommodate but the most commonly used one is the remote procedure call (RPC) pattern. SOAP operates over Internet protocols such as simple mail transfer protocol (SMTP) [35], the file transfer protocol (FTP) [36], and most notably HTTP [37].

Taking our generic SOA diagram, shown in Figure 14, we can illustrate the typical web service use case, Figure 23, using the SOA approach.

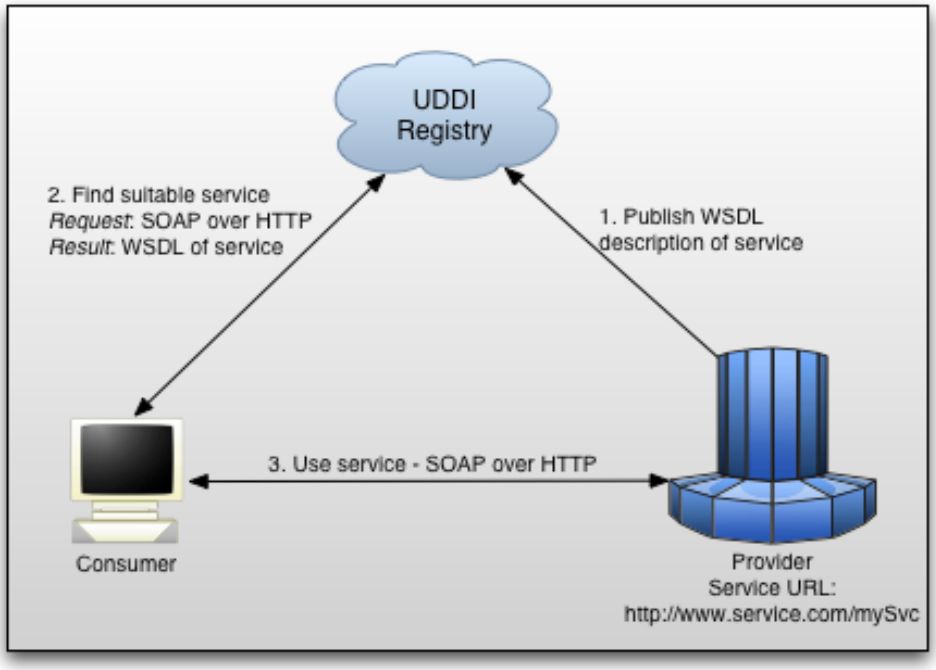


Figure 15 Web Services and SOA

In Figure 15, for the consumer to interact with the service mySvc hosted by the provider at the well-known address of http://www.service.com, the consumer must first retrieve the WSDL interface of the service by first issuing a SOAP request to the UDDI registry.

Based on the WSDL interface the consumer has full knowledge of the service including the well-known endpoint of the service, descriptions of the types the service understands and what methods the service exports. Using the WSDL interface the consumer can either dynamically invoke the service, by parsing the WSDL and generating an appropriate SOAP message or statically by generating stubs using a language specific WSDL compiler.

The following list outlines several reasons of the rapid adoption of Web Services in SOA:

- A major advantage of web services is by using HTTP, they can work through many common firewall security measures without requiring changes to their filtering rules. This has been the downfall of many distributed architectures for use over wide area networks. e.g. DCOM and CORBA. This becomes a great advantage and strength when discussing the needs of a SME, as discussed later in section.
- Due to SOAs property of loose coupling, web services allow the reuse of services and components within an infrastructure with minimal dependencies.
- Web services allow services from different companies and locations to be composed easily to provide an integrated service, such as the business process and execution language (BPEL) [38].
- As web services leverage open standards and protocols. This provides interoperability between various software applications running on various platforms and architectures. Also as many of the protocols and data formats are text based, it makes it easy for developers to understand and debug request/response flows between services.

4.2 Assumptions of Web Service-based SOAs

An assumption of web services is the availability of static end points to deploy services. When a web service is implemented, in its WSDL definition, the developer of the service needs to enter the well-known URL of where the web service will be made available:

```
<wsdl:definitions
  xmlns=http://schemas.xmlsoap.org/wsdl/
  xmlns:wsdl=http://schemas.xmlsoap.org/wsdl/
  xmlns:wsdlsoap=http://schemas.xmlsoap.org/wsdl/soap/
  xmlns:dbe="urn:soapimpl.emailservice.demos.dbe.org"
  xmlns:xsd=http://www.w3.org/2001/XMLSchema
  targetNamespace="urn:soapimpl.emailservice.demos.dbe.org">
...
  <wsdl:service name="EmailServiceService">
    <wsdl:port name="EmailService" binding="dbe:EmailSoapBinding">
      <wsdlsoap:address location="http://localhost/EmaiService"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

While Web Services tackle interoperability problems prevalent in existing distributed object models, such as CORBA and DCOM, it poses a performance penalty, because each message has to be marshalled into an object model and vice-versa. Additionally, one problem arises with the static mapping of a published WSDL interface definition to the associated deployed service instance. The mapping binds the endpoint of the service instance to the actual WSDL interface definition

published in the UDDI registry. Therefore, a consumer requesting a Web Service interface definition from the UDDI registry can bind to the actual service endpoint using dynamic or static invocation mechanisms.

If the endpoint address of the Web Service changes often the UDDI entry representing that service cannot guarantee an up-to-date WSDL interface definition. Consequently, this will affect service consumers who try to access the web service as well as potential service providers who do not have any means of providing a static and well-known URL for their services. Also, the web services SOA model assumes the availability of a centralised UDDI registry. However, in the DBE peer-to-peer architecture, there will be no UDDI registry and service registration and lookup will be performed using the DBE services semantic registry and FADA. This can be envisioned as a potential problem for SMEs trying to bridge the “digital divide”.

4.3 SOA and Small-to-Medium Enterprises

SOAs and their associated technologies, while removing many of the problems with established middleware systems such as CORBA and DCOM, are still to heavyweight for the average SME unattainable due to the high resource requirements needed to implement them. One of the major goals of the DBE is to reduce these resource requirements to a bare minimum in order to enable SMEs to make their services available in the DBE using minimal resources. In order to minimise these resource requirements, the design of the first DBE implementation needs to make assumptions about the resources available to the average SME. With these assumptions in mind we can model the SMEs' environment, discover potential problems related specifically to SME environments and provide solutions for them. The assumptions made are:

1. *Financial Resources*: The SME initially has a minimal budget for financial support of the design, implementation and maintenance of services.
2. *Technical Resources*: We assume that a SME shall have basic experience with computers. One could expect a proficiency that an ECDL [39] type qualification would provide. As such, the expected minimum skills of the SME would not exceed the basic usage of word processors, spreadsheets, email applications and web browsers.
3. *Internal Infrastructural Resources*: The SME is assumed to have a minimum of:
    1. At least one host with the possibility of more connected using a private internal network. The local host/network is administered by the SME
    2. SMEs often have hosts with DHCP-allocated IP addresses.
    3. A dial-up or more commonly a broadband Internet connection e.g. DSL with 512/128 kbps.
    4. A firewall (deployed by SME and/or ISP) to secure the Internet-facing hosts.
    5. Local resources should be used to host DBE services.
4. External Resources: All DBE infrastructural services should be implemented as P2P services to reduce costs for SMEs.

With these assumptions in mind we now examine problems that are posed by designing a SOA for the average SME environment.

4.4 Problems of SOA in the SME Environment

Although SOAs provide many solutions to publishing and consuming services there are a number of problems that are major barriers to basing the DBE SOA on a web services model. In this section we discuss the problems that are specific to the SME consumer/provider and problems that our SOA for DBE needs to address. The service discovery and usage problems we have identified that result from the assumptions about the typical SME environments are:

- Firewall traversal
- Network Address Translators
- Dynamic Host Allocation Protocol Leasing and Service Deployment
- Service Publishing and Availability

Firewall Traversal

It is now commonplace and good practise for Internet facing computers to be protected by one or more firewalls. Firewalls are also used to protect the networks of most Internet service providers (ISP) to provide additional protection for customer computers. Firewalls operate on the principle of Internet protocol (IP) packet inspection. When an IP packet arrives at the interface of a firewall, the firewall examines the destination address and port in the packet's header and then based on a set of rules, makes a decision whether or not the packet should be allowed pass. In the case on a host where a user deploys a service on a host with a port number blocked by a firewall for incoming connections, any potential service consumer trying to connect and use the advertised service will fail due to the firewall dropping the consumer's packets.

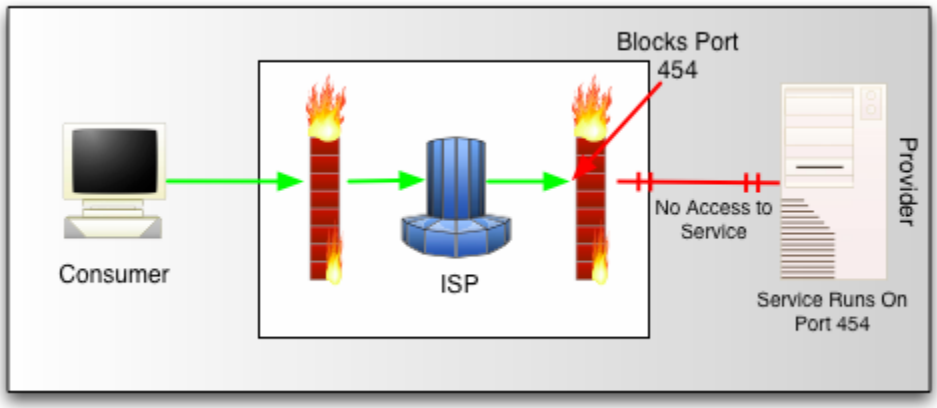


Figure 16 Firewall Traversal Problem

This problem situation is illustrated in Figure 16. Because SMEs usually do not have access to ISP's firewall configuration, the design of a SOA for DBE should allow for the traversal of firewalls without configuring any specific firewall rules. As aforementioned, Web Services meet this requirement, because it is based on the HTTP transport protocol.

Network Address Translation Traversal

Due to the limited address space in IP, version 4, Internet service providers (ISP) have adopted the widespread use of network address translators (NAT) to increase the number of hosts they can connect to the Internet to over-come this problem. NAT is a technique in which the source and/or destination addresses, including port numbers, of IP packet headers are rewritten as they are relayed through NAT gateway. NAT does not provide end-to-end connectivity with communications. NAT is most commonly used to enable multiple hosts on a private network to access the Internet using a single public IP address. Therefore, the design of a SOA for DBE should allow for the traversal of routers and firewalls using NAT in a transparent fashion and not propagate this issue to the DBE consumer. However, the existing web services model does not provide a way for NAT gateway traversal.

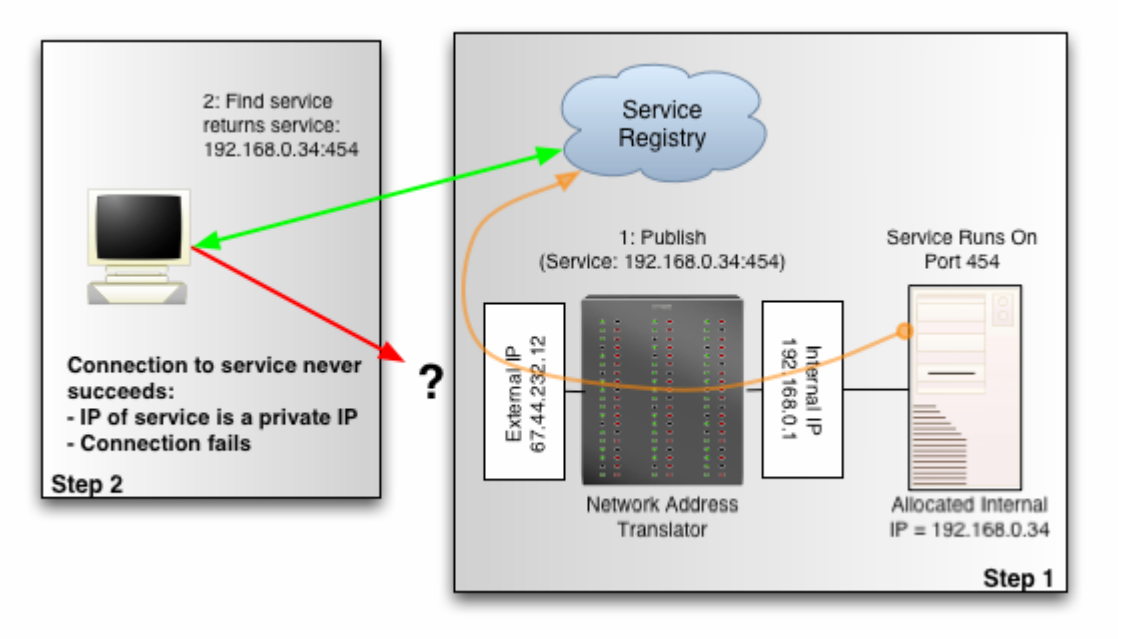


Figure 17 Network Translator Traversal Problem

Figure 17 illustrates the problem of NATs deployed in the same environment as the deployed DBE service. In this situation the service provider has created and deployed a service. The next step for the provider is to register the service with a service registry e.g. a UDDI registry. The provider then registers the interface of the service along with the binding information detailing the service's location. If the binding information is a location behind a router/firewall running NAT then the default IP address contained in the binding information will be a private IP address and will not be accessible by anyone outside of the NAT's network. To provide a solution for the problem of traversing, a NAT gateway, a peer-to-peer architecture will be implemented in the next 18 months that uses relay servers to traverse NAT gateways.

Dynamic Host Allocation Protocol Leasing and Service Deployment

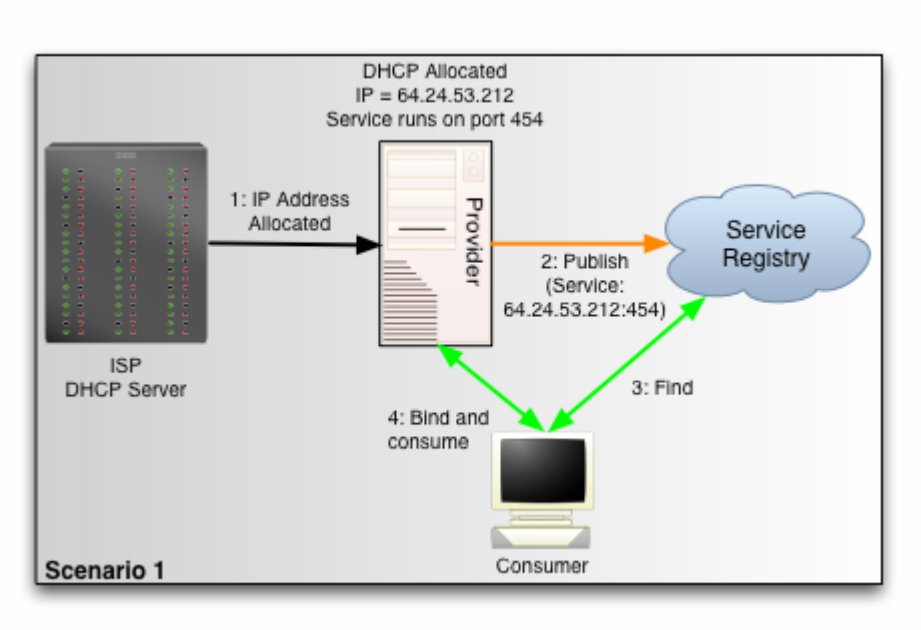


Figure 18 DHCP Address Allocation: Initial Scenario

In order for ISPs to efficiently manage and configure multiple hosts' IP configurations the dynamic host configuration protocol (DHCP) [40] is used to perform such a task. The process of a SME computer acquiring a DHCP allocated IP address is illustrated in Figure 18.

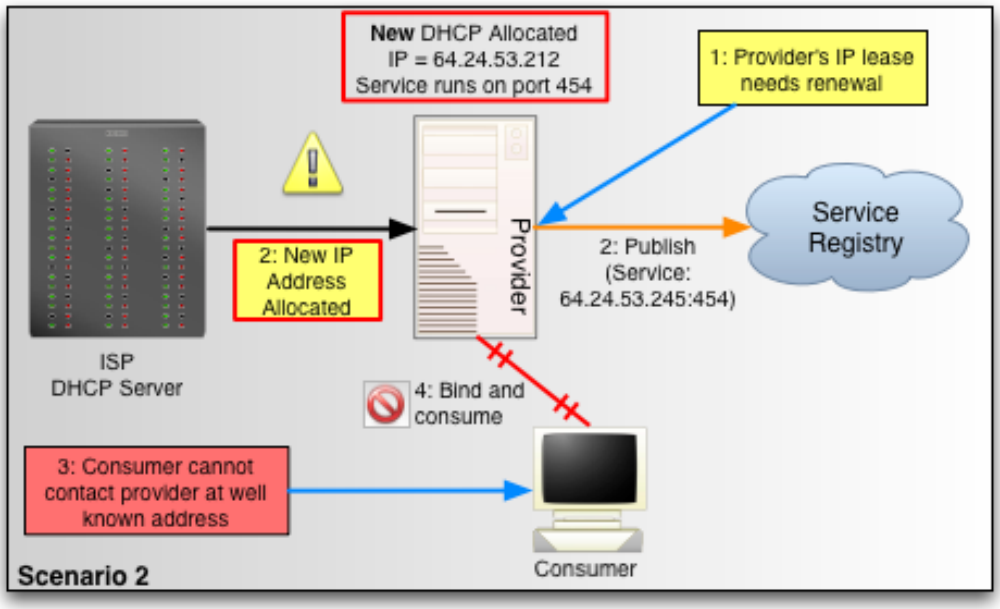


Figure 19 DHCP Address Allocation: Renewal Problem

When the provider's computer is first connected to the network, it acquires an IP address, using the DHCP protocol, which is given out by the ISP's DHCP server. The provider is then able to provide a service to consumers. Every DHCP allocated IP address is leased for a set period. When this lease expires the host has to release and renew its IP address lease, a lease that can be from seconds to days. Often the new IP address that is returned on renewal is not the same as the one the provider was providing services on. In the web service model, the service registry is not updated with new binding information for the provider's service. Any consumers that attempt to use the provider's service using the old endpoint information contained in the service registry, will not be able to use the service, as illustrated in Figure 19. To address the problems posed to the SME by DHCP addresses, a persistent name service using the semantic registry and leasing mechanism using FADA has been implemented.

Service Publishing and Availability

For many SMEs having a 100% service uptime is not feasible, due to financial, technical and infrastructural constraints. As such the SOA for DBE should take service availability into account. The distinction between a SME's service being "out-of-business" or just out of order for the day needs to be made. This distinction is necessary to avoid consumers of a service ceasing the use of it when the consumer perceives the service to be no longer available when in fact the service has stopped running for a period of time and will resume operation after that time e.g. at the weekends.

This is where static endpoints such as those used in WSDL interface definitions fail. The service discovery process does not distinguish between "no longer available" and "temporarily unavailable". No information is available to the consumer of the service. Service discovery is now a 2-stage process involving a persistent name service, the semantic registry, to discover service identifiers and a service leasing system, FADA, to acquire a lease to a deployed and available service. To address the problems of service publishing and availability, the same technique to circumvent the DHCP problem is used.

4.5 Conclusions

This chapter presented the the domain analysis needed to identify potential problems with providing a SOA for SMEs. SOAs and their application to the SME domain have been described and discussed. This discussion firstly listed the assumptions made about the typical SME and problems of applying SOA with those assumptions to SMEs. With problems revealed, solutions to these were then presented. The domain analysis has provided prerequisites for the following chapter where requirement analysis is performed and the functional model is described.

5. Functional Model

The following use cases introduce the functional models for the DBE Service Factory (DBE Studio), the DBE Desktop, the DBE Portal, and the DBE Servent with respect to the requirements analysis [dbeArch] and the design document [dbeArchScope] from Soluta to comply to the first implementation of the DBE. It is anticipated that in the second half of the project, when much of the "intelligence" of the DBE is managed by the peer-to-peer network, some of these use cases will be simplified, e.g., the service manifest creation will not require the service developer to specify a service deployment location.

5.1 DBE Service Factory

The DBE Service Factory is an Integrated Development Environment (IDE) in order to design and implement DBE Services. It provides all tools of existing IDEs as well as those specifically designed for DBE in order to accomplish this task.

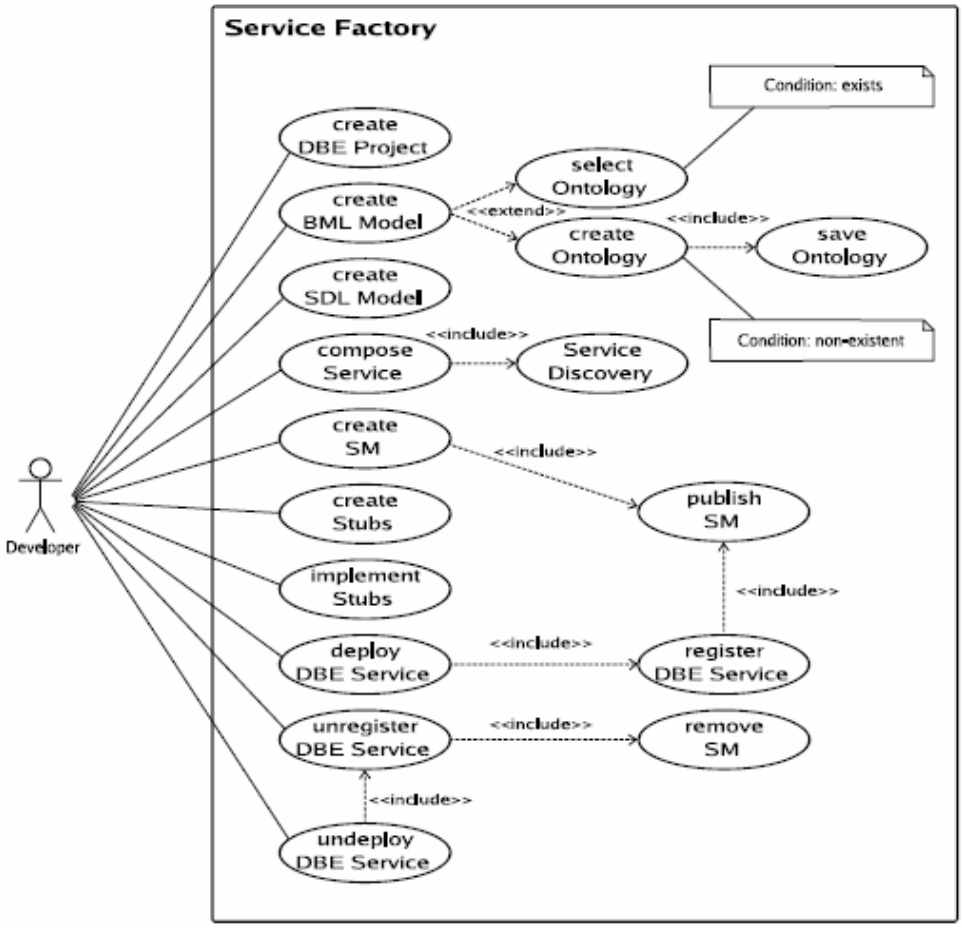


Figure 20 Service Factory Use Cases

Figure 20 illustrates the use cases for the DBE Service Factory. The following tables detail each of the use cases and puts them into the context of the development of a DBE Service.

Use Case	***Create DBE Project***
Area:	Service Factory
Objective:	Create/load a DBE Project
Actors:	Developer
Pre-Conditions:	
Post-Conditions:	
Description:	The developer will create a new or load an existing DBE project into the workspace of the Service Factory. The project adheres to a certain project layout whereby each folder represents a specific functional implementation of the DBE Service, such as the BML model, the SDL model, the service composition, the service implementation, and the 3rd party libraries of the service.
Dependencies:	
Responsible Partner:	TCD

Use Case	***Create BML Model***
Area:	Service Factory
Objective:	Create/Load a BML Model
Actors:	Developer
Pre-Conditions:	• Active DBE Project in the DBE Service Factory's workspace
Post-Conditions:	• If the developer created a new or modified an existing ontology, it will be available to every participant in DBE. Ontologies are not owned by the developer.
Description:	The developer will create the business model representation of a new DBE Service or load and modify an existing one based on a business ontology. The developer will need to select an existing ontology from the knowledge base which identifies the business domain of the service. Otherwise the developer will create a new ontology and save it into the global knowledge base which suits his/her needs to represent the service appropriately.
Dependencies:	Select Ontology, Create Ontology, Save Ontology
Responsible Partner:	TUC

Use Case	***Create SDL Model***
Area:	Service Factory
Objective:	Create/load a SDL Model
Actors:	Developer
Pre-Conditions:	Active DBE Project in the DBE Service Factory's workspace
Post-Conditions:	
Description:	The developer will create the technical service definition of the DBE Service or load an existing one into the workspace. The technical service definition identifies methods with their in- and output parameters.
Dependencies:	
Responsible Partner:	Soluta

Use Case	***Compose Service***
Area:	Service Factory
Objective:	Compose Service
Actors:	Developer
Pre-Conditions:	• Active DBE Project in the DBE Service Factory's workspace • DBE Services must be available
Post-Conditions:	• If service instances of the composition become unavailable, then the workflow of the composition is incomplete unless the design is “fault-tolerant”.
Description:	The developer may compose his/her DBE Service of existing services. The composition represents the flow of events of an incoming service request and the internal interaction between the services including the handling of exceptions. The developer will find and compare existing services in order to compose the most suitable one with his/her service.
Dependencies:	Service Discovery
Responsible Partner:	TCD

Use Case	***Create Service Manifest***
Area:	Service Factory
Objective:	Create/Modify the Service Manifest of a DBE Service
Actors:	Developer
Pre-Conditions:	• Active DBE Project in the DBE Service Factory's workspace • BML model must be created • SDL model must be created
Post-Conditions:	• If the Service Manifest is published, it becomes available to DBE participants. But the corresponding service can not be executed. • If the Service Manifest is published, the developer is owner of this Service Manifest.
Description:	The developer may create the Service Manifest once the BML and the SDL model have been created. The Service Manifest can be stored in the semantic registry at this point.
Dependencies:	Publish Service Manifest
Responsible Partner:	TUC

Use Case	***Create Stubs***
Area:	Service Factory
Objective:	Create/Modify the Service Stubs of the DBE Service
Actors:	Developer
Pre-Conditions:	• Active DBE Project in the DBE Service Factory's workspace • SDL model must be created
Post-Conditions:	
Description:	The developer will create the stubs of a service based on the technical description (SDL Model). For the first 18 months the stubs will be created for the Java programming language. If the SDL model has been changed the stubs have to be re-created to reflect those changes.
Dependencies:	
Responsible Partner:	TCD

Use Case	***Implement Stubs***
Area:	Service Factory
Objective:	Implement/Refactor the Stubs of the DBE Service
Actors:	Developer
Pre-Conditions:	• Active DBE Project in the DBE Service Factory's workspace • SDL model must be created • The service stubs must be created
Post-Conditions:	
Description:	The developer will implement the service stubs or refactor the existing implementation if the stubs and/or the back-end have been changed. The stub implementation will delegate to the business logic of the service which can be implemented as a legacy back-end service and hosted on a remote computer.
Dependencies:	
Responsible Partner:	

Use Case	***Deploy Service***
Area:	Service Factory
Objective:	Deploy the complete DBE Service
Actors:	Developer
Pre-Conditions:	• Active DBE Project in the DBE Service Factory's workspace • BML model must be created • SDL model must be created • The service stubs must be created • The service must be implemented • Optionally, the service may be composed of existing DBE Services
Post-Conditions:	• The service is owned by the developer and can not be changed by anyone else in the DBE. • The service is available for execution and its corresponding models can be browsed in the Semantic Registry.
Description:	The developer will deploy a complete DBE Service. The deployment includes the registration of the DBE Service with FADA and the publication of the Service Manifest in the Semantic Registry. As a result, the service is visible to DBE consumers and therefore can be utilised.
Dependencies:	Register DBE Service, Publish Service Manifest
Responsible Partner:	TCD

Use Case	***Unregister Service***
Area:	Service Factory
Objective:	Unregister the DBE Service instance
Actors:	Developer
Pre-Conditions:	• Active DBE Project in the DBE Service Factory's workspace • The DBE Service must have been deployed • The developer must be the owner of the service to be unregistered
Post-Conditions:	• The service is not available for execution
Description:	The developer will unregister a DBE Service in order to remove a service instance from the FADA service registry.
Dependencies:	(Remove Service Manifest)
Responsible Partner:	TCD

Use Case	***Undeploy Service***
Area:	Service Factory
Objective:	Undeploy the complete DBE Service
Actors:	Developer
Pre-Conditions:	• Active DBE Project in the DBE Service Factory's workspace • The DBE Service must have been deployed • The developer must be the owner of the service to be undeployed
Post-Conditions:	• The service is not available for execution • In case the Service Manifest is removed from the Semantic Registry, the service models will not be available • The developer must be the owner of the service to be undeployed.
Description:	The developer will undeploy a DBE Service in order to remove a complete service from the DBE infrastructure. Optionally, the service models described in the Service Manifest will be removed from the Semantic Registry.
Dependencies:	Unregister Service, Remove Service Manifest
Responsible Partner:	TCD

5.2 DBE Servent

The DBE Servent is a server component which provides the runtime environment for DBE Services and access points for consumers to execute deployed services.

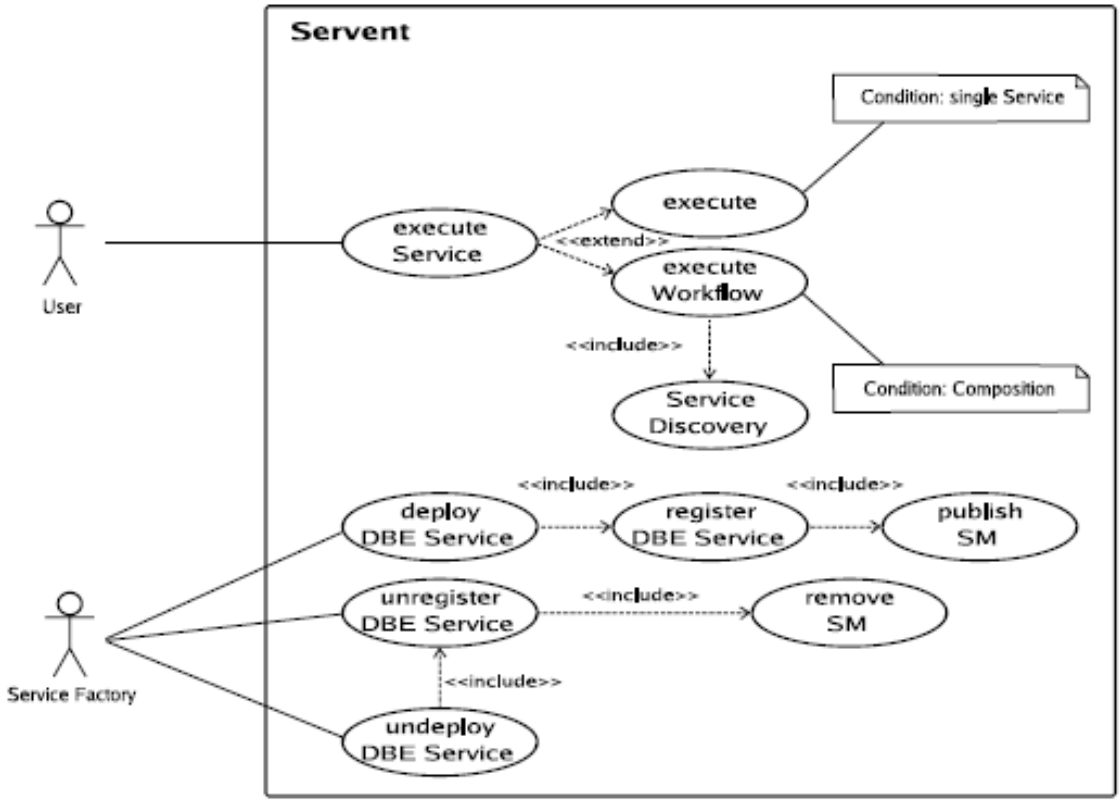


Figure 21 DBE Servent Use Cases

Figure 21 illustrates the use cases applicable for the DBE Servent. Each use case is further described in the tables below.

Use Case	***Execute Service***
Area:	Servent
Objective:	Execute a deployed DBE Service
Actors:	User
Pre-Conditions:	• DBE Services must be available
Post-Conditions:	
Description:	The user will execute a deployed DBE Service. If the service is a service composition the workflow associated with this composition is executed and all participating services are discovered. Otherwise the single service implementation is executed.
Dependencies:	Execute, Execute Workflow, Service Discovery
Responsible Partner:	TCD

Use Case	***Deploy Service***
Area:	Servent
Objective:	Deploy the complete DBE Service
Actors:	Service Factory
Pre-Conditions:	• The complete service must be packaged in a self-contained .jar-file.
Post-Conditions:	• The service is available for execution and its corresponding models can be browsed in the Semantic Registry.
Description:	The Service Factory will deploy a self-contained DBE Service which comprises of the models and the implementation. The deployment includes the registration of the DBE Service with FADA and the publication of the Service Manifest in the Semantic Registry. As a result, the service is visible to DBE consumers and therefore can be utilised.
Dependencies:	Register DBE Service, Publish Service Manifest
Responsible Partner:	TCD

Use Case	***Unregister Service***
Area:	Servent
Objective:	Unregister the DBE Service instance
Actors:	Service Factory
Pre-Conditions:	• The DBE Service must have been deployed
Post-Conditions:	• The service is not available for execution
Description:	The Service Factory will unregister a DBE Service in order to remove a service instance from the FADA service registry.
Dependencies:	(Remove Service Manifest)
Responsible Partner:	TCD

Use Case	***Undeploy Service***
Area:	Servent
Objective:	Undeploy the complete DBE Service
Actors:	Service Factory
Pre-Conditions:	• The DBE Service must have been deployed
Post-Conditions:	• The service is not available for execution • In case the Service Manifest is removed from the Semantic Registry, the service models will not be available
Description:	The Service Factory will undeploy a DBE Service in order to remove a complete service from the DBE infrastructure. Optionally, the service models described in the Service Manifest will be removed from the Semantic Registry.
Dependencies:	Unregister Service, Remove Service Manifest
Responsible Partner:	TCD

5.3 DBE Portal

The DBE Portal is responsible for providing the DBE applications. It is a platform to download the DBE Desktop, the DBE Studio, and DBE Eclipse plug-ins. Additionally, services can be searched and executed on the portal to allow users to use a Web Browser rather than a rich client application.

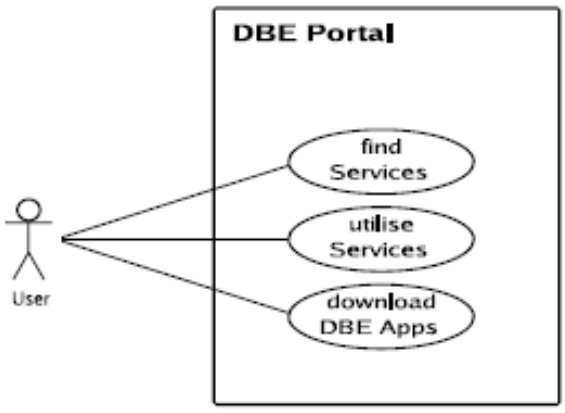


Figure 22 DBE Portal Use Cases

Use Case	***Find Services***
Area:	DBE Portal
Objective:	Find existing DBE Service through a Web Browser
Actors:	User
Pre-Conditions:	• DBE Services must be available
Post-Conditions:	
Description:	The user will issue search queries in a Web Browser in order to find services he/she is interested in.
Dependencies:	User Interface (web-based)
Responsible Partner:	TCD, SUN, Intel

Use Case	***Utilise Services***
Area:	DBE Portal
Objective:	Utilise an existing DBE Service through a Web Browser
Actors:	User
Pre-Conditions:	• DBE Services must be available • The search query must return at least two DBE Services
Post-Conditions:	
Description:	The user will utilise an existing DBE Service through a Web Browser. The service will display a HTML user interface with input fields to fill in by the user.
Dependencies:	Web-based GUI Development Tool
Responsible Partner:	Intel, SUN

Use Case	***Download DBE Applications***
Area:	DBE Portal
Objective:	Download DBE Applications from the portal
Actors:	User
Pre-Conditions:	• DBE Applications must be available
Post-Conditions:	
Description:	The user will download DBE rich client applications, such as the DBE Desktop and the DBE Studio, from the portal. Also, the portal would host all eclipse plug-ins to extend and update the features of both DBE applications.
Dependencies:	
Responsible Partner:	TCD

5.4 DBE Desktop

The DBE Desktop is a light-weight rich client application for DBE consumers to assist him/her in searching for existing DBE Services and utilise them.

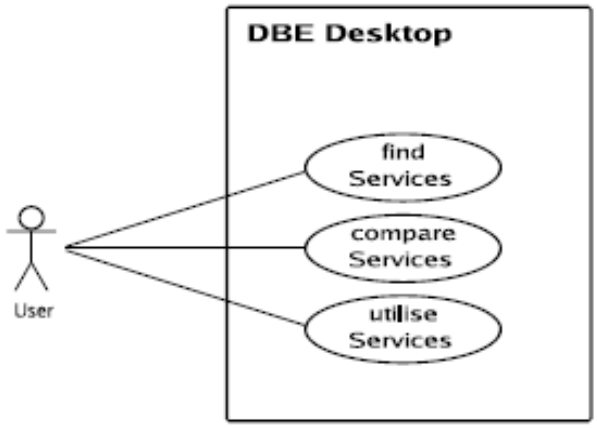


Figure 23 DBE Desktop Use Cases

The Figure 23 delineates the use cases for the DBE Desktop which are further explained in the following tables.

Use Case	***Find Services***
Area:	DBE Desktop
Objective:	Find existing DBE Service
Actors:	User
Pre-Conditions:	• DBE Services must be available
Post-Conditions:	
Description:	The user will issue search queries in order to find services he/she is interested in.
Dependencies:	
Responsible Partner:	TUC

Use Case	***Compare Services***
Area:	DBE Desktop
Objective:	Compare existing DBE Service
Actors:	User
Pre-Conditions:	• DBE Services must be available • The search query must return at least two DBE Services
Post-Conditions:	
Description:	The user will compare existing services based on business and/or technical related properties.
Dependencies:	
Responsible Partner:	

Use Case	***Utilise Services***
Area:	DBE Desktop
Objective:	Utilise an existing DBE Service
Actors:	User
Pre-Conditions:	• DBE Services must be available • The search query must return at least two DBE Services
Post-Conditions:	
Description:	The user will utilise an existing DBE Service. The service will display a user interface with input fields to fill in by the user.
Dependencies:	
Responsible Partner:	

5.5 Conclusions

This chapter covered the use cases of the DBE Service Factory, the DBE Desktop, the DBE Servent, and the DBE Portal with respect to the requirements analysis [3] and the design document [2] from Soluta.Net.

These use cases may be simplified from a DBE actor's perspective in the second half of the project and moved to the “intelligence” middleware of DBE to provide a simpler and more transparent way of interacting with DBE services.

6. Design and Implementation

6.1 DBE Service Factory

Model-Driven Architecture

One of the main goals in the DBE is to reduce the “Digital Divide”. To aid in reducing this division the MDA approach was chosen to minimise the amount of code that an SME may have to produce. Using the MDA methodology the service factory can be mapped into three main activities. These are:

- Business Specification: This maps to the Computational Independent Model (CIM) activity.
- Interface Specification: This maps to the Platform Independent Model (PIM) activity. Currently it is possible to generate code from this level.
- Service Implementation: This maps to the Platform Specific Model (PSM) activity. Currently at this level it is necessary for the SME to implement the business logic in a platform-specific language which is in the DBE’s case, Java.

The above activities have corresponding mappings to the DBE studio and they are integrated into the DBE studio as visual software tools. See Figure 24 for the component diagram. These tools are encapsulated as eclipse perspectives and can be summarised as follows:

- CIM Perspective: Within this perspective all modelling that takes place is done in the integrated CIM editor. The integrated CIM editor represents the BML editor in the DBE Studio workbench. In the CIM perspective a user models the service with respect to business-related ontologies.

- PIM Perspective: All modelling in this perspective is performed by the integrated PIM editor. This editor represents the SDL editor and its related tools and wizards. In the PIM perspective a user models the platform-independent technical aspects of a service hiding the underlying software technology.
- PSM Perspective: In this perspective all modelling and authoring is done by the integrated PSM editor. This editor is encapsulated by the PSM perspective within the DBE Studio workbench, In the PSM the user implements the service in a fashion that is specific to the target software on which the service will be deployed upon.

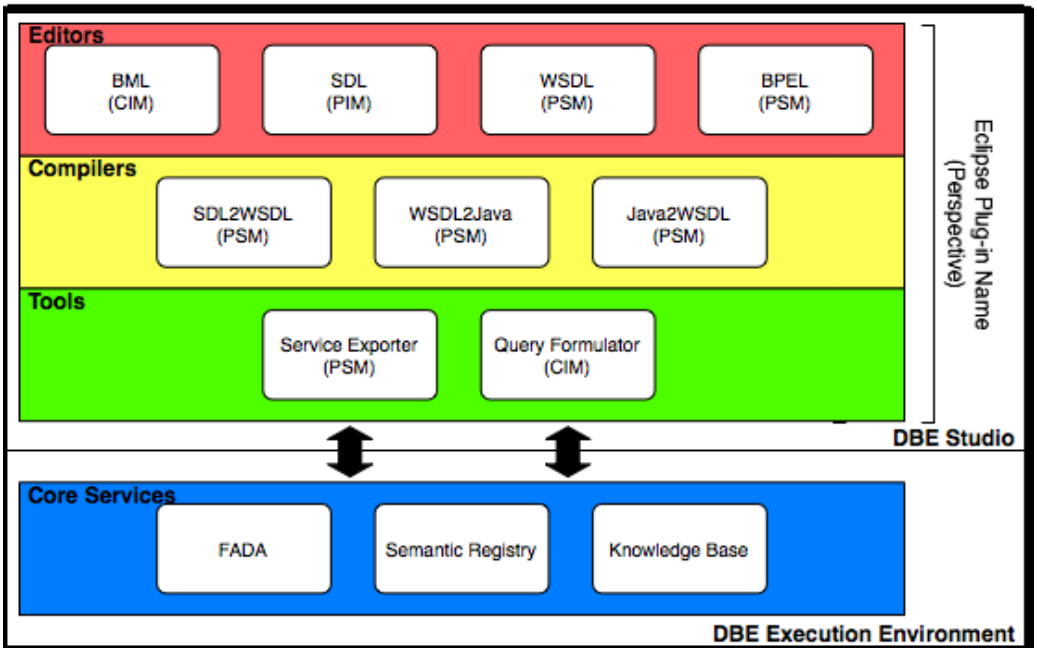


Figure 24 DBE Studio Plug-in Components

The above plug-ins and their related perspectives can be shown in the following work flow diagram.

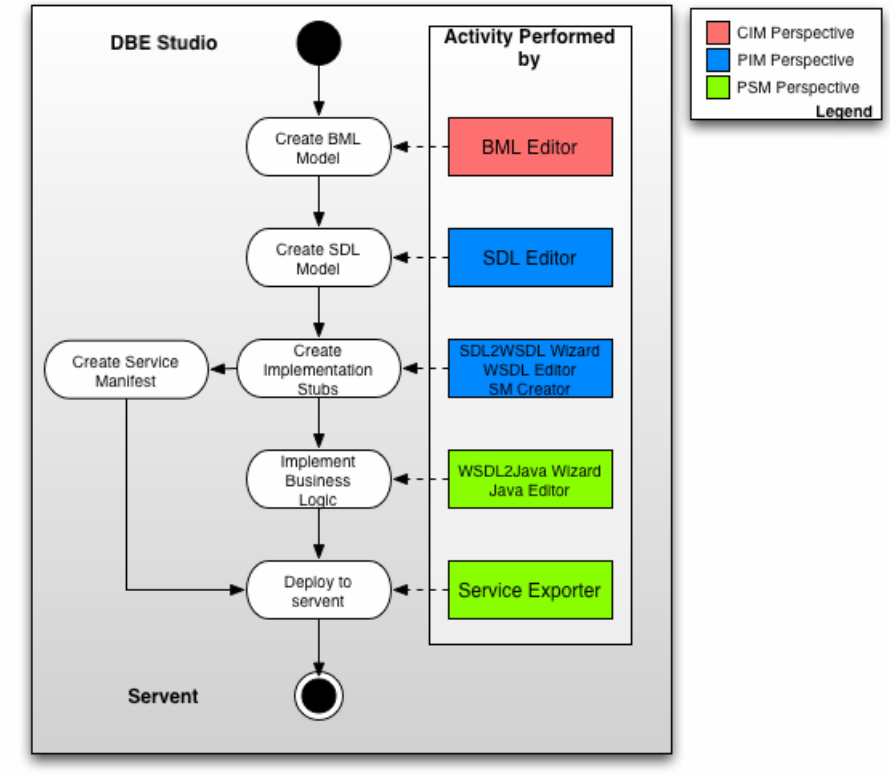


Figure 25 DBE Studio Workflow and Related Components

Integrated DBE Platform

Eclipse offers the potential for building an integrated IDE (as an Eclipse Workbench) that can be adapted to all the different actors in the DBE without changing the plug-in implementations. The DBE functional architecture components have been implemented as plug-ins to an eclipse application as it allows (see Figure 26):

- An integrated DBE application (DBE Studio) for Service Usage and Development
- Single platform that can be used by SME Service Providers, Service Implementers and Service Users
- DBE Service Factory Tools as Plug-ins
  - BML Editor, SDL Editor, SDL Compiler plug-in, WSDL Editor, WSDL compiler plug-in, etc.
- DBE Run-time Services and Tools as Plug-ins
  - Service Discovery, Recommender Service, Fada lookup Service, etc.
- Downloadable collections of plug-ins as packaged “features” with necessary legal and security mechanisms
  - Update mechanism for versions of DBE Tools and Services

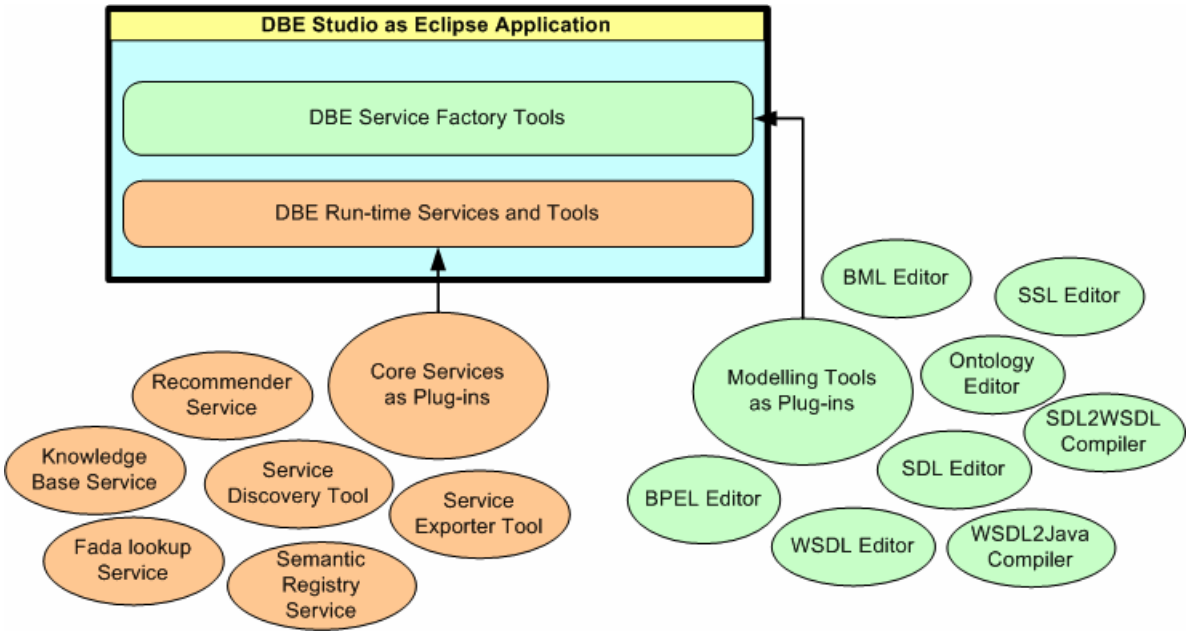


Figure 26 Proposed DBE Application User Interface in Eclipse

DBE Studio as an Eclipse Application

The DBE Studio, as it is initially presented to the user, loads with an introductory splash screen, which then presents the DBE Studio start page. The start page allows the selection of a perspective the user would like to start using. Each of these perspectives in the DBE Studio consists of an editor (in fact in some cases more than one) and their supporting views. Both of these are implemented using the eclipse plug-in framework. In Figure 27, an architectural overview of the DBE Studio is given and its components' relationship to the eclipse framework is presented.

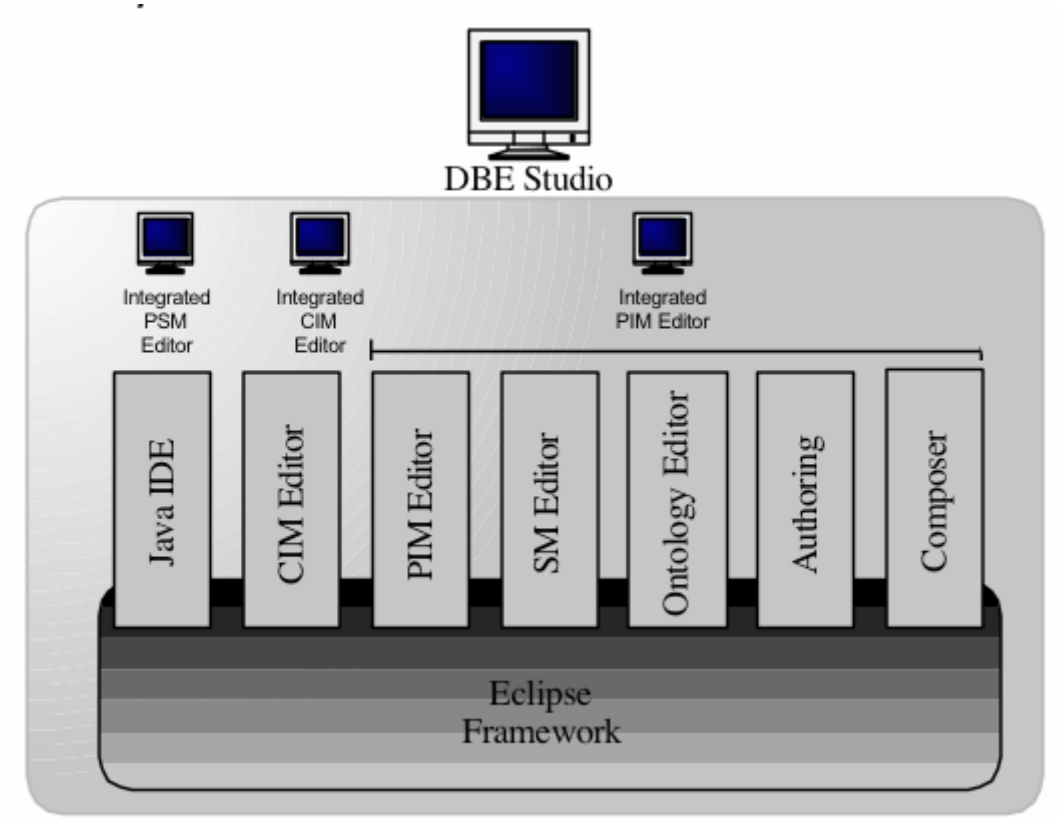


Figure 27 DBE Studio Architecture

DBE Studio

The DBE studio is where the SMEs with adequate technical skills and knowledge can model and create services.

In order to create a service that can be offered with the DBE, the DBE studio needs to be expressive and rich enough so that multiple users (actors) can describe the many aspects of the process in creating a DBE service or product. These aspects range from the business motivation of the service to the description of types passed through the interface of the service to be created. The DBE studio provides a number of visual tools to model and create services and aims to provide seamless support from service inception and description right through to deployment and updating of that realised service. The process of realising a DBE service contains a number of phases. Each transition from one phase to another is supported by semi-automated model and code generation. Within the DBE Studio there are five distinct and sequential activities, as shown in Figure 25:

1. *Business Modelling*: This is the activity where the business model of the service is created. The BML editor is responsible for this task. It has been developed by the Technical University of Crete (TUC) and has been implemented as an eclipse plug-in that uses the Graphical Editor Framework (GEF) to provide a graphical user interface so that a user can accurately model his/her business and services. The BML editor is part of the BML perspective within the workbench. The major features it supports are the creation and import of BML models, which can be stored and retrieved from the Semantic Registry and the easy creation of BML models using a familiar UI metaphor of box and line drawing. Further description of this plug-in is

outside the scope of this document. TCD's part in the BML editor was the tasks of integrating the BML editor into the DBE Studio.

2. *Service Interface definition*: To model the SDL of a service, Soluta created a SDL editor. The SDL editor is part of the PIM perspective within the workbench. The SDL editor presents a hierarchical, tree-based view of the SDL model of the technical interface description of a DBE Service. For any developer who has any experience with XML, using such an editor is intuitive and productive and results in SDL being quickly and easily modelled.
3. *Stub and Service Manifest Creation*: To generate stubs and skeletons from SDL resulting from the service interface definition stage, Soluta created the SDL2WSDL eclipse plug-in. This plug-in converts SDL to WSDL. Once the SDL2WSDL plug-in has performed its task of converting the PIM SDL into PSM WSDL it is then necessary to generate the stubs and skeletons for the service implementation. To do this TCD developed an eclipse WSDL2Java plug-in, that allows the developer of the service to create the required stubs and skeletons. This plug-in provides a graphical user interface to the Apache axis WSDL2Java compiler and provides all the same options for stub and skeleton generation that the command line version does. For a developer to use it, an entry in the context menu of the WSDL file is provided that performs this task. The WSDL2Java wizard is presented and the developer can then specify the options and the output location for stub and skeleton generation. Should developers need to edit the WSDL file TCD created a WSDL editor. The editor presents a hierarchical, tree-based view of the WSDL model that is being created. The difference between this editor and the SDL editor is that the WSDL editor includes platform specific elements such as the nodes **binding** and **service**.

   With the service stubs and skeletons generated, the service manifest needs to be generated. This is accomplished by an eclipse plug-in implemented by TUC named the Service Manifest Creator. This plug-in retrieves the service's BML and SDL from the semantic registry and from them, creates the service manifest.

4. *Business Logic Implementation*: This is the activity where the business logic of the modelled service is implemented. The business logic of the service is edited and implemented in the eclipse in-built Java Editor.
5. *Service Deployment*: For a service to be deployed to a servent, it is necessary to package all the necessary files that are required to run a service successfully on the servent. To this end, TCD have initially defined what should be in a deployment unit within the DBE. The deployment unit has been named as a DBE archive, or DAR for short. The DBE archive is a compressed file (it is in fact an appropriately named Jar file) that can contain the implementation of the service, the implementation of the service's UI, service workflow information (if applicable), the service manifest, and also a configuration file that informs the servent how the service should be configured at runtime. There are a number of different deployment options that can take place from the DBE Studio. There are as follows:

   - Deploy service without UI
   - Deploy service with UI
   - Deploy service workflow
   - Deploy service manifest only

   To create a DAR file the developer of the service needs to create a compressed file containing the required files for the service's needs. Currently the contents of the DAR file have to be named in a specific fashion. The naming scheme for files to be contained with a DAR is as follows:

- Service implementation = serviceImpl.jar
- Service composition = serviceImpl.bpr
- Service manifest = service.manifest
- Configuration file = dbeservice.xml

When deploying the actual service implementation, which can contain the service UI implementation, the developer needs to package the implementation classes as a Jar file. Once the developer has completed the service, he/she can then proceed to use the service exporter to deploy the service. To do this, the developer right-clicks on the service's project and selects "Export". A dialog is then presented where the developer selects "Jar file". The developer is then presented with the Jar Export Wizard. From this the developer selects the required files for his DAR file, supplies a location and file name (ending in the extension .dar) and then clicks the finish button to create the DAR file. In order to actually deploy a DAR file to the servent, an eclipse plug-in, the Service Exporter, was developed. This plug-in, is a contextual menu plug-in that presents itself when a user in the PSM perspective right clicks on a WSDD file. A wizard is then presented where the user specifies the DAR file that is to be deployed to the servent. The user also needs to specify the URL of the deployment web service that is hosted by the servent. If the user has not supplied a configuration file in the DAR file, a wizard page will guide the user through the creation of one and automatically insert the configuration file into the DAR file. There is also an optional advanced wizard page, which the user can call up. On this page the user can specify an interceptor [Interceptors in TCD WP Del.] configuration for the service so additional value added services can be "attached" to the user's service, like accounting [WIT]. Once the user has supplied all the information to the wizard, all that is required is the finish button pressed and the service exporter will send the DAR file across to the servent via web service.

6.2 DBE Servent and Execution Environment

The DBE Execution Environment (ExE) provides an architecture where services can be deployed, discovered and ultimately consumed. It is this dynamic environment that is referred to as being the run-time of the DBE. A goal of the ExE is to hide all the complexity inherent to distributed systems from the "regular programmer", that is, a programmer with minimal, perhaps no, experience in distributed systems. The ExE is not supposed to replace the back-end systems of SMEs, but rather to allow SMEs to exposed their back-end systems as services from which they and their clients can interact, evolve and integrate. A robust and effective architecture is necessary to provide an environment where SMEs can rely on a consistent quality of service from the tools and structural services using the ExE. This is even more important for the provision of the SME's own commercial services, where a SME's reputation and success depends on the stable performance of this complex service-oriented environment.

The ExE also provides transparent integration with the Evolutionary Environment [41]. For the Evolutionary Environment to be fully implemented, it is essential that the ExE provides a well defined structure which is maintainable but adaptive to the evolution mechanisms within the DBE. The ExE is required to allow non-restrictive access to its environment from the evolutionary components, by defining various interfacing points to services and data which are actively present throughout the environment.

The following diagram relates the execution environment with the previously discussed service factory.

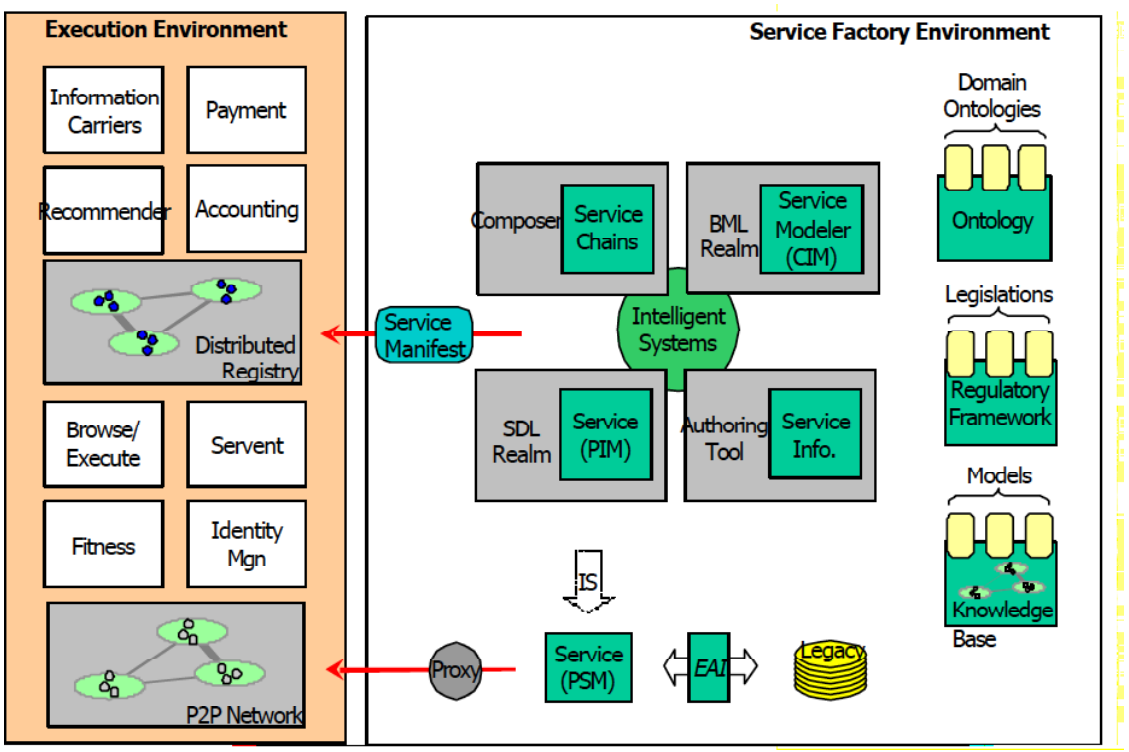


Figure 28 ExE's Relationship to SF

Within the execution environment there are a number of use cases for which the design of ExE should respect. These have been detailed already in chapter 5. Briefly these use cases are as follows:

- *Service Registration and Deployment*: Once a service has been defined and implemented in the DBE Studio, it is then registered in the DBE nervous system (the Fada peer-to-peer network) accompanied by its service manifest.
- *Service Search and Retrieval*: To execute a particular service, the SME must first find the correct service to execute. The SME accomplishes this by issuing a set of search criteria (including business semantics and service interface attributes) to the discovery or recommender tool. The result of this query will return back a list, listed as service manifests, of all the possible suitable services. By selecting one of these service implementations, the service proxy is downloaded to the client side servent and can be utilised there.
- *Service Execution*: Once the service proxy is in the Servent of the client SME, the client may execute it. The service proxy makes use of the proxy framework that is installed along with the DBE servent. On invocation, the service proxy reveals its service UI to the user, who can interact with it and make requests. On sending a request, the service proxy invokes the back-end service via the servent. The servent also provides an interceptor framework so other services such as the accounting service can be called from the Servent transparently. The result of the invocation is then returned back to the client and displayed in the UI.

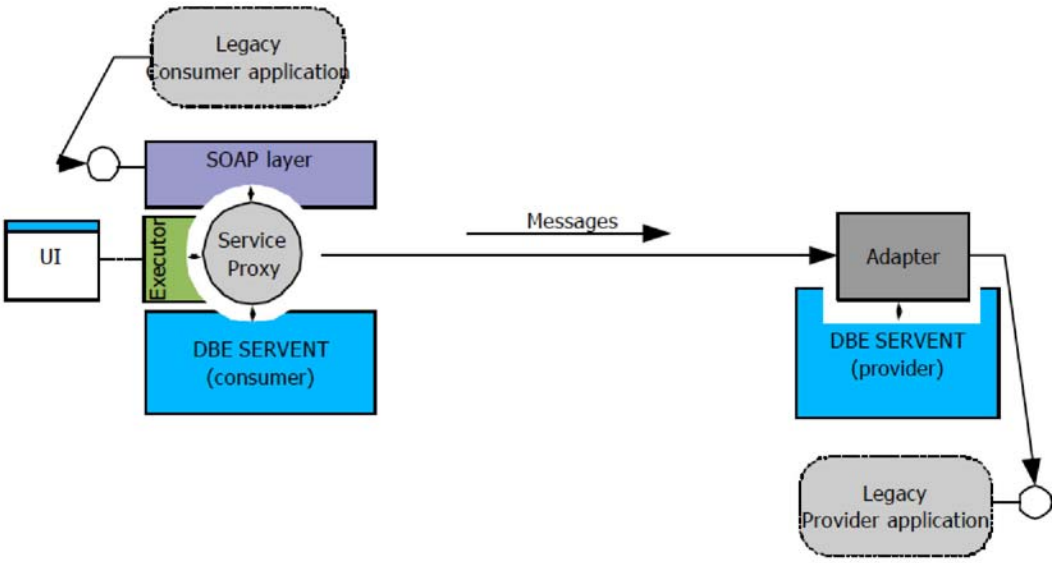


Figure 29 Execution in the ExE

The execution environment has been implemented as a set of tools and services. This collection is known as the servent.

- *Core Services*: The services that will be initially offer in the ExE and by servents will be: Knowledge Base Service, Semantic Registry Service, FADA, DSS and Metering.

- *Environment*: The environment that a servent runs under is execution container independent. To date the servent runs under both Apache Tomcat and Jetty. The servent is also protocol agnostic, a characteristic that is provided by the Abstract Protocol Adapter (APA), which will be explained in detail in the next section. Currently the servent can communicate using the SOAP protocol and also object serialisation over HTTP.

The name “Servent” is in fact an amalgamation of the two words **serv**er and cli**ent**. This term, Servent or as it is sometimes called Servant, is used in a similar context on existing P2P architectures, such as Gnutella. As discussed and explained in other documents [refs], the ExE is part of a P2P network and correspondingly so too is the servent. Each servent contributes to the P2P network by sharing its resources e.g. services or disk space, in the case of the DSS [intel].

A goal for the servent [2] is that all services appearing in the ExE should appear as if they were deployed locally, even though they are not. This will simplify the programming of DBE services as there will not be any need for developers to deal with communications in DBE. The P2P network that is created between servents will be transparent to users, by means of the servent architecture.

Service Deployment in the Servent

The service deployment service is a component of the servent and runs within the execution container (Tomcat/Jetty). It is implemented as a web service and takes as its input a DAR file that contains a service that is to be deployed on the servent. The sequence of operations that take place to deploy a service are illustrated in figure 30.

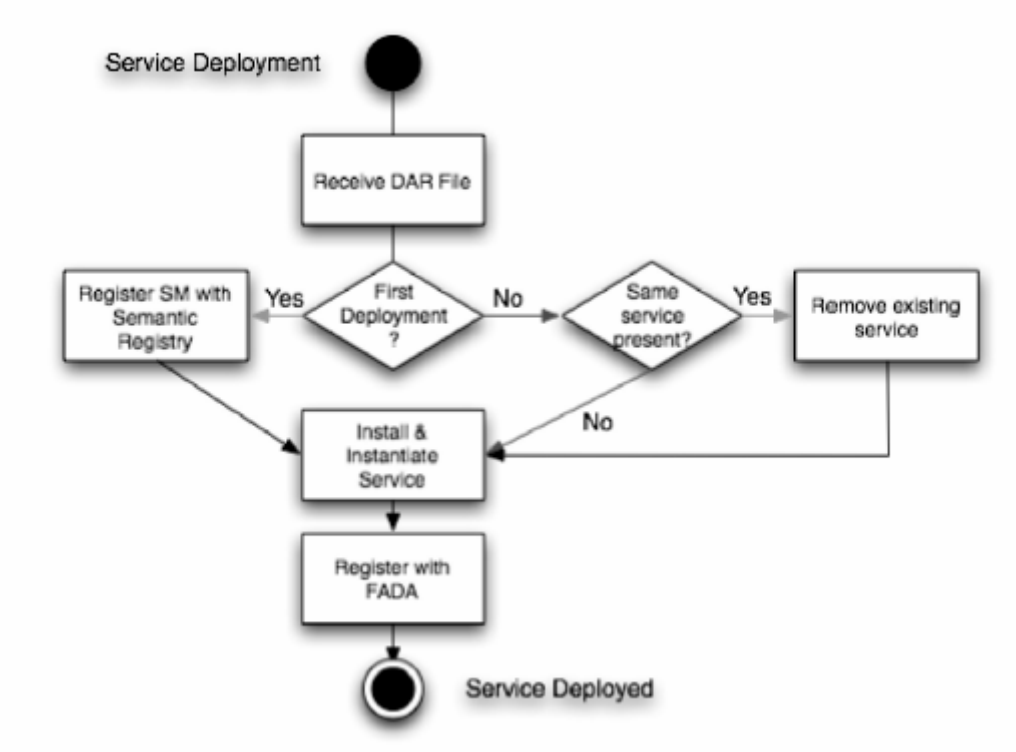


Figure 30 Server-side Service Deployment

When a DAR file is received the deployment service checks that there is a service manifest contained within the DAR file. A SM is mandatory for the first deployment of a service. If the deployment is a first time one, the SM is extracted from the DAR file and is saved to the Semantic Registry, which returns a SM ID referencing the SM. Once the SM ID is received, the service implementation is extracted and installed within the execution container. Once installed and instantiated, the service is then registered in FADA using the SM ID as a unique entry for lookup within FADA.

If the deployment is not the first one of the same service, that is, a SM ID already exists for the service, then the existing service is stopped and removed from the execution container. The new updated service is then installed and instantiated within the execution container and using the existing SM ID, it is registered to FADA.

DBE Toolkit

The loosely coupled architecture of SOA implementations provides a flexible platform for client-side developers to design their applications against remote services. Although, there is still room for greater flexibility between client applications and services. If a service is implemented to support a single communication protocol, e.g. SOAP, then the client application will need a separate adapter or proxy to create the necessary SOAP calls. This static binding to services can be restrictive, therefore a more abstracted dynamic approach would be more favourable to the adaptive nature of client-service relationships within the DBE.

The DBE Toolkit was designed to provide a set of development frameworks and tools to assist DBE developers in the implementation of both server-side and client-side applications within the Execution Environment. The initial version of this toolkit has two frameworks, the Proxy Framework and the Abstract Protocol Adapter framework. These frameworks provide a simple format for the deployment, discovery and invocation of services. It is intended that DBE developers will be able to easily implement client side applications, which perhaps are integrated with back-end legacy systems, that are capable of dynamically binding to and invoking remote DBE services.

Proxy Framework

The Proxy Framework provides a set of interfaces and classes for registering, discovering and invoking services using FADA. FADA allows the usage of varied typed objects for the proxies of registered services, therefore the Proxy Framework was designed to use a structured type definition for service registration and proxy representation. The Service Proxy class is there to provide a structured way to set service information as properties in a serialised object stored in FADA. Using the Map class provided by the java.util package, this object can be initialised by the Service Provider before registration with FADA, where all the necessary properties can be set. Such properties can be the choice of PA (Protocol Adapter), the binding information for the service (i.e. endpoint URL, port number, service name), and any interceptor information for a specific PA (i.e. which interceptors to invoke on the client-side accounting, security, user identity). The PA is part of the Abstract Protocol Adapter framework and will be discussed more later in this section.

On the Client side, the property information will be delegated to the Workspace class which will possibly initiate a user session, and therefore delegating the dynamic invocation of the remote service to the Abstract Protocol Adapter (see later in this section for more details on how services are dynamically invoked). Using the information provided by the properties, the required PA will be invoked. Here the required interceptor chain will be created as specified in the properties for that service instance. The service will be invoked dynamically without downloading any client stubs from a code base. Additionally, if a UI is provided for the service instance, then the classes for the UI factory and UI implementations will be needed to be downloaded from a code base.

The Proxy Framework is integrated with FADA to register and retrieve Service Proxy objects. The framework provides a helper class for performing Fada lookups and discovery registered proxies. The ServiceProxyLocator class wraps around the functionality provided by the Fada Toolkit API and provides a simpler interface to the Fada lookup mechanism. Also, this removes any need for the client application developer to have knowledge of the Fada Toolkit API.

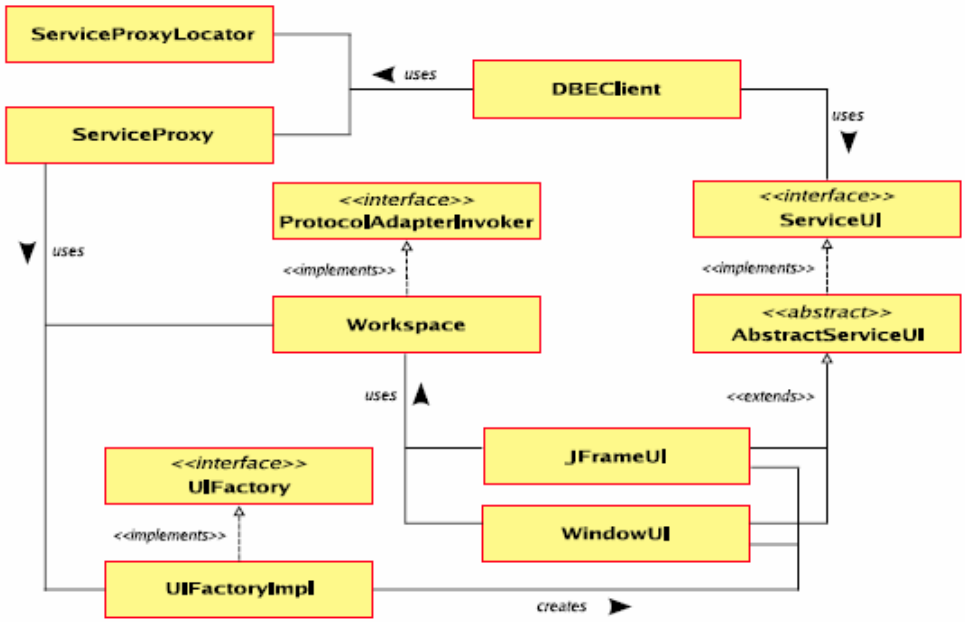


Figure 31 Static Class Diagram of the Proxy Framework

The design of the Service Proxy does provide the functionality for different types of client side UIs to enable a human user to interact with the service. This involves the creation of a UI factory, at runtime on the client side, once the service proxy has been retrieved. The client can then use this UI factory to instantiate different UIs to consume the selected service. In addition, to providing functionality for a UI factory, it may be necessary to provide access to the remote DBE service from a client which has no human user, but some sort of IT infrastructure or legacy system. This creates the need for Service Proxy to provide a programmatic interface, as well as a user interface. This programmatic interface is provided by using of the previously suggested workspace concept. The Workspace class is integrated with the APA framework to enable developer to use the its generic operations to dynamically invoke a service. The Service Proxy is a deploy-time vehicle that carries a factory of runtime objects, i.e. a UI and workspace. At runtime these objects provide the passage for message flows from the client user to the UI to the workspace to the APA and then finally to the remote DBE service.

In the client scenario, where the human user interacts with the service via a UI, the entire functionality of the remote service is accessible by the workspace object created by the service proxy object at runtime. The UI code is encapsulated in a separate object, which will interact with the workspace object to consume the service. An advantage to this architecture, in which UI and service functionality are separated, is that multiple UIs can be associated with one service. This will enable the service provider to tailor different UIs for different clients and for different purposes, such as a general user UI or an administrator UI. The workspace object, which enables access to all the functionality of the remote service and is created at runtime by the service proxy, could also provide some sort of a persistent client session for a user of the service.

In another client scenario, where there is no human user and the service is consumed by a back-end IT infrastructure or legacy system. This direct-use client interacts with the remote service functionality directly by invoking the generic methods on the workspace object. This scenario demonstrates another advantage for separating the UI from the service functionality, by enabling client applications to access a service without human intervention or supervision.

The Proxy Framework is a standardized way of interacting with remote service implementations. The Proxy Framework provides session capability for service instances. Sessions and caches can be implemented in the workspace, which does not contain any logic at the moment. It delegates the RPC invocation to the APA component. If all services implement the Proxy Framework it can be abstracted further in order to ease and reduce the development effort for service providers. Integrating with such a generic Proxy Framework is not complex. No generation of classes or download of those on the client-side is necessary. The Proxy Framework would provide a dynamic invocation mechanism, which is loosely coupled to the service implementation. Actually, it does not contain any information of the remote service implementation. Only UI implementations would be service-specific.

Abstract Protocol Adapter

The Abstract Protocol Adapter (APA) is a framework, which provides an abstract extensible layer for generic client-side dynamic invocations. The APA was initially developed so we could integrate SUN's implementation of the Servent at an early stage in the project, e.g. the first implementation of the Servent is a combination of Tomcat and axis, SUN's is a stand alone Java application using combination of Jetty that talks object serialisation over HTTP. The APA adds to the achieving the goal of PIM in the context of the MDA approach by enabling protocol independence during the specification of service models.

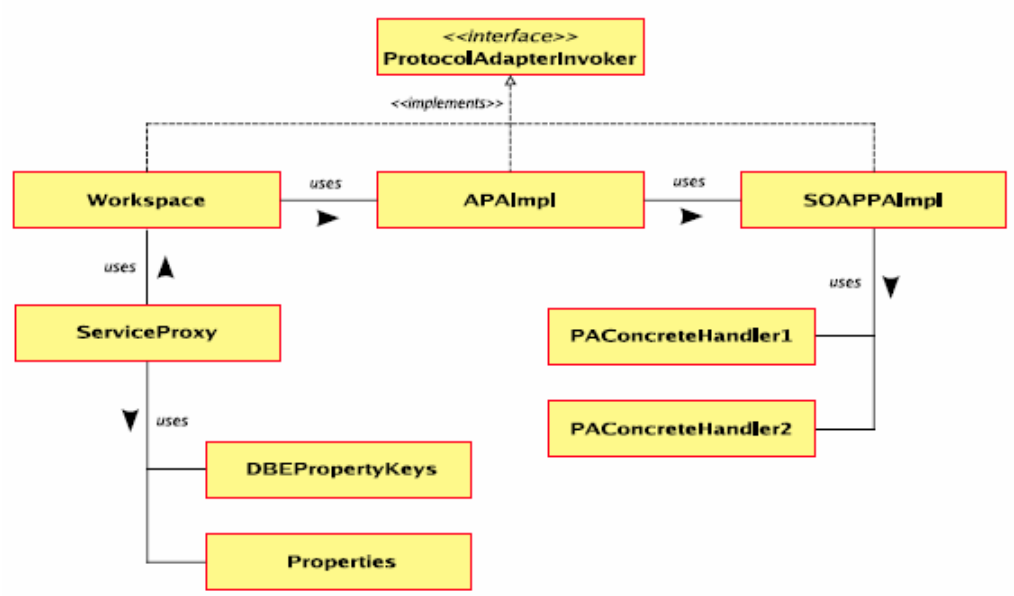


Figure 32 Static Class Diagram of the APA

The APA framework can be integrated into any component, which requires a protocol independent communication mechanism. In such a scenario the protocol is declared at runtime, rather than at compile time. APA itself does not know anything about protocols. It delegates to a concrete protocol adapter at runtime, which is selected by the encapsulated properties within the Service Proxy object. As a result it is a matter of configuration. APA promotes lose coupling between clients and protocols.

APA easily supports Intra-enterprise servents , rather than having a local servent. In this use case we would have the following flow of information:

client -> APA(1) -> Intra-enterprise servent -> APA(2) -> remote servent -> concrete protocol adapter -> service implementation

The first APA redirects the request to the Intra-enterprise servent based on local configuration. The second APA actually invokes the remote service. Both APA may use different communication protocols.

Any number of PAs can be implemented for various protocols. For the first 18th month implementation three PAs have been developed, which one is compatible with SOAP, kSOAP and object serialisation over HTTP. The SOAP PA supports the invocation of remote web services which expose SOAP endpoints. The kSOAP PA supports web service communication from mobile clients to kSOAP compliant services.

6.3 DBE Portal

At the heart of the DBE portal is a content management system. A content management system (CMS) is a system used to organize and facilitate collaborative content creation. It provides a simple way to create, maintain and update a portal. The DBE portal exposes similar functionality that is present in the DBE desktop and even the DBE Studio to browse and execute services. Not only does it expose these functionalities but the portal is also a distribution point for the DBE applications including the Desktop, the Studio (including plug-ins) and also any frameworks that a developer may be interested in using e.g. the APA. Further, it is anticipated that the portal is also the main entry point for an SME wishing to join the DBE and it will be here where the SME will register with the DBE in order to provide their services.

The method of interfacing with the DBE portal is by using a web browser (Mozilla, Internet Explorer, Opera, etc.). The portal does not require any specialised client side plug-ins such as ActiveX controls, Java run-time environments etc. and as such the web browser is a thin client. All computation is performed on the server (portal) side with the results of computation displayed on the client (web browser). The computation on the server side is performed by Java Servlets interfacing with the DBE via the toolkits and frameworks provided by the Java development kit (JDK) and those already provided by TCD, the DBE Toolkit and the APA. Taking this approach allows any user of the DBE without a full installation of DBE applications to access DBE with just the bare minimum of a web browser. The portal is broken down into three main areas of functionality. These are:

- Informational Functionality
  - View and download DBE documentation.
  - Interact with and get help from other DBE users in forums.
- Application/Service Functionality
  - Search for existing DBE Service
  - Execute an existing DBE Service
  - UDDI style service browsing
  - White directory: based on address and contact information
  - Yellow directory: based on service categorised by industry/business
  - Green directory: based on technical information about services
- Infrastructural Functionality
  - Registration with the DBE via distributed identity service
  - Download DBE Applications from the portal
  - Listing of DBE boot nodes (FADA, SRs, KBs)
  - DBE Statistics, e.g., number of KB nodes, network topology and it size.

6.4 DBE Desktop

The DBE Desktop is based on the Eclipse Rich Client Platform which had been introduced to Eclipse in version 3.0 and consists of a minimal set of Eclipse plug-ins to build a rich client application. These plug-ins are managed by a container model that is the basis of all Eclipse applications.

The plug-in manifest, the plugin.xml file, ties together the resources, such as the images and .html help files, of the DBE Desktop with the main classes. The main class and the perspectives are declared in their appropriate extension points, `org.eclipse.core.runtime.applications` and `org.eclipse.ui.perspectives` respectively. When the DBE Desktop starts up the plug-in manifest advices the eclipse runtime which main application to invoke and which perspective to load in the initial workbench. The following Figure 33 delineates the class diagram of the DBE Desktop.

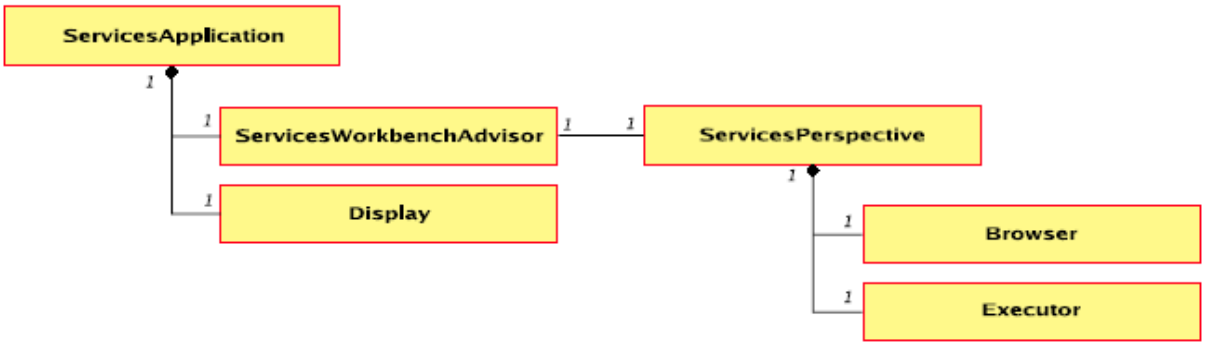


Figure 33 DBE Desktop Class Diagram

The ServicesApplication is the controller of the underlying MVC pattern. It is responsible to create the Display and the ServicesWorkbenchAdvisor and to execute the Workbench. The ServicesWorkbenchAdvisor establishes the menu entry and advices on which perspective is loaded up initially. The DBE Desktop consists of only one perspective, the ServicesPerspective, which is a composition of two views, the Browser and the Executor views. To date the Browser browses the FADA entries, whereas the Executor executes one DBE Service registered in the FADA registry and selected by the Browser. It is anticipated that the Service Discovery tool from TUC will supersede both existing views. The Service Discovery tool interacts with the semantic registry to find a DBE Service and the FADA registry to execute a particular DBE Service instance.

7. Conclusions

7.1 Achievements

As stated in the Technical Annex, the “Expected results: Proof of concept prototype of the DBE service description, deployment and usage”.

TCD has developed a Development Process that is potentially suitable for other large-scale FP6 software projects. TCD also drove forward the usage of Eclipse as the development platform for the rich client applications, the DBE Studio and the DBE Desktop, in order to gain on Eclipse's abstraction of the native Windowing Toolkits to overcome challenges from heterogeneous environments from SMEs.

The execution environment has been updated to introduce an abstract protocol adapter (APA) component, developed and implemented by TCD, that abstracts RPC-style remote method invocations to work over different underlying transport protocols, e.g. SOAP. The APA consists of two basic parts, the APAImpl and a concrete protocol adapter. The APAImpl delegates method invocations to a concrete protocol adapter which implements some middleware specific protocol e.g. SOAP, RMI. The preference for the type of method invocation is dynamic (either via dynamic stub generation or dynamic proxies). The proxy framework enables the invocation of DBE services and internally uses the APA. A ServiceProxy is instantiated with all the information necessary to invoke the service implementation that is deployed via FADA. TCD have also been working on the service deployer an eclipse plug-in that deploys DBE services on the servent. Its unit of deployment is a DAR (DBE archive) file, which contains the service implementation (either as a standard web service for a composite web service), service configuration, and service manifest.

It will in future support the configuring of interceptors (handlers), which can be optionally deployed with the service. TCD has also been working on an updated FADA Browser, an eclipse plug-in, which allows to graphically browse the contents of a FADA cloud and invoke those services it discovers.

7.2 Second Phase Implementation

The implementation tasks for the second phase of the project will be overseen by SUN. The second implementation of the Servent will be provided by SUN and will be integrated to the DBE Studio, DBE Toolkit and the other tools and services within the Execution Environment. TCD's contribution to the implementation under this work-package will come from the Distributed Identity System (task C18) and the P2P Architecture and Service-Oriented Routing task.

8. Bibliography

1: Digital Business Ecosystem Homepage, http://www.digital-ecosystem.net

2: Del 21.2: Architecture Scope Document,

3: https://dbe.digital-ecosystem.net/files/documents/8/298/Del_21.1_DBE_Architecture_Requirements_2.pdf

4: DBE Technical Annex 1, https://dbe.digital-ecosystem.net/files/documents/8/6/file_6.dat?filename=DBE%20Annex%20I%5ffinal%2edoc%2egz

5: Maven Homepage, http://maven.apache.org

6: CruiseControl Homepage, http://cruisecontrol.sf.net

7: JUnit Homepage, http://www.junit.org

8: The Apache Software Foundation, http://www.apache.org/

9: Kent Beck, Extreme Programming Explained: Embracing Change, 2004

10: XPlanner, http://www.xplanner.org/

11: Concurrent Versions System Homepage, http://www.cvshome.net

12: GNU RCS Homepage, http://www.gnu.org/software/rcs/rcs.html

13: ClearCase Product Homepage, http://www-306.ibm.com/software/awdtools/clearcase/

14: Subversion Homepage, http://subversion.tigris.org/

15: CVS - Concurrent Versions System, http://www.gnu.org/software/cvs/

16: CollabNet Homepage, http://www.collabnet.net

17: Apache ANT Homepage, http://ant.apache.org/

18: Code Conventions for the Java Programming Language, http://java.sun.com/docs/codeconv/

19: Draft Java Coding Standard, http://g.oswego.edu/dl/html/javaCodingStd.html

20: Watts S. Humphrey, A Discipline for Software Engineering, 1995

21: Eclipse Platform, http://www.eclipse.org/platform/

22: Jetty Homepage, http://jetty.mortbay.org

23: Apache <webservices/> project - Axis, http://ws.apache.org/axis/index.html

24: Model Driven Architecture, http://www.omg.org/mda

25: IBM, New to Web Services, http://www-128.ibm.com/developerworks/webservices/newto/

26: OMG CORBA FAQ, http://www.omg.org/gettingstarted/corbafaq.htm

27: Object Management Group, http://www.omg.org

28: SOAP Specifications, http://www.w3.org/TR/soap/

29: OASIS, www.oasis-open.org

30: W3C, http://www.w3.org

31: WS-I, http://www.ws-i.org

32: DCOM, http://www.microsoft.com/com/

33: Michael Huhns, Munidar P. Singh, Service-Oriented Computing: Key Concepts and Principles, 2005

34: WSDL Specifications, http://www.w3.org/TR/wsdl

35: Simple Mail Transfer Protocol, http://www.ietf.org/rfc/rfc821.txt

36: File Transfer Protocol, http://www.ietf.org/rfc/rfc959.txt

37: Hypertext Transfer Protocol -- HTTP/1.1, http://www.ietf.org/rfc/rfc2616.txt

38: Business Process Execution Language, http://www-128.ibm.com/developerworks/library/specification/ws-bpel/

39: European Computing Driving License Foundation, http://www.ecdl.com/main/index.php

40: Dynamic Host Configuration Protocol, http://www.ietf.org/rfc/rfc2131.txt

41: Del 23.1: DBE Fitness Landscape, https://dbe.digital-ecosystem.net/files/documents/8/451/Del_23.1_Final_DBE_Fitness_Landscape.pdf