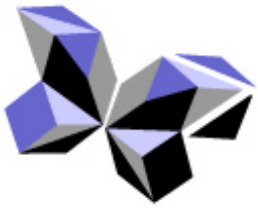

Digital Business Ecosystem

Contract n° 507953

Workpackage 21: DBE Architecture Requirements

Deliverables D21.4: Detailed Technical Architectural Models

Information Society Technologies

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

Contract Number: 507953

Project Acronym: DBE

Title: Digital Business Ecosystem

Deliverable N°: D21.4

Due date: 12/2006

Delivery Date: 01/2007

Short Description:

The document introduces and describes the DBE components UML Interaction Reference Model, it depicts components, software services and their dependencies as part of the DBE service factory and the DBE execution environment.

Partners owning: Soluta.Net

Partners contributed: Soluta.Net

Made available to: All DBE partners and the EC

Version	***Date***	***Author***	***Description***
v00.01.0	June 2006	P. Ferronato	draft
v00.02.00	November 2006	P. Ferronato	Draft for first internal review
v01.00.00	January 2007	P. Ferronato	Final for second internal review

Quality check

1st Internal Reviewer: John Kennedy (Intel Ireland Ltd.)

2nd Internal Reviewer: Jason Finnegan (Telecommunications Software and Systems Group - Waterford Institute of Technology)

Table of Contents

1 Abstract....5
2 Executive Summary....6
3 Interaction Logical View....8
3.1 Service Factory....8
3.2 Execution Environment....9
3.3 Overall Interaction Model....10
3.3.1 UML Interoperability Reference Model Description....11
3.4 Other Components and Future Implementations....14
4 "Plug-in to plug-in" Interaction Types in Eclipse....15
4.1 Mouse double-click....15
4.2 Drag&Drop....15
4.3 Adding JFace viewer drag support....16
4.4 Adding view drop support....17
4.5 Plug-in drop handling....17
4.6 Transfer....19
4.7 Transfer types supported by the standard views....20
4.7.1 Cut, Copy and Paste....20
4.7.2 Launching a plug-in from another plug-in....21
5 Service discovery and execution....23
6 DBE Components....25
7 Glossary....28
8 References....31

Figure Index

Figure 1: DBE Components UML Interoperability Reference Model.....10
Figure 2: Ontology Viewer to SDL Editor Communication ("plug-in to plug-in").....15
Figure 3: Interaction between a generic component and a DBE service.....23

1 Abstract

This deliverable introduces and describes the DBE components UML Interaction Reference Model. This UML model describes the operations that are provided and required by the software components included in the DBE project, together with the specification of the dependencies among the components[1]. A software component corresponds to a reusable piece of software with a precise functional responsibility, that can be integrated with other components in the DBE environment. The model is depicted as a UML Component Diagram.

This document is a descriptive compendium around the UML Interaction Reference Model, as part of Task C36. It depicts components, software services and their dependencies as part of the DBE Service Factory and the DBE Execution Environment.

The goals of the UML model described in this document is to help understand the rather complex DBE architecture and the overall component architecture and topology.

In the document details about technical interactions are not considered: specific communication protocol, interfaces, and activation sequences for example are not in scope and won't be described. Instead, every component included in the model will be categorized as a specific service type, and every interaction will be mapped to a set of well defined "interaction types".

Other documents necessary to understand DBE architecture are [WP21DEL21.1], [WP21DEL21.2], [WP21DEL21.3].

1 A component is said to be dependent on another component if the former requires and uses at least one of the operations provided by the latter

2 Executive Summary

The main functionality of DBE is explained in [WP21DEL21.1] . This document highlights the role of the main DBE Components in relation to the functional requirements of the project.

From a very high level viewpoint, the DBE utilization scenario assumes that a Small to Medium Enterprise (SME) specifies its business model (using the Business Modelling Language, BML), provides specific information related to it, defines the model of the computational interface (using the Service Definition Language, SDL) and eventually implements some code and deploys the service.

The DBE architecture consists of three distinct environments: the Service Factory (SF), the Execution Environment (ExE) and the Evolutionary Environment (EvE).

The SF is constituted of a set of tools that allow SMEs to describe and publish their services. The SF environment offers modelling editors, regular Java based development environment and a repository where service models and data can be published to and retrieved from.

The ExE is a decentralized environment that allows services to be published, searched and consumed over the Internet. Furthermore the ExE provides a set of structural features like identity, metering and logging.

The EvE is designed to be a background environment where new services are automatically composed, the new services are obtained by chaining the basic services of the ExE environment. EvE observes ExE at run time, and identifies repetitive sequences of services and the most requested services, EvE focuses on the needs of the DBE system. In response to this system behaviour the EvE produces new services as a composition of the basic ones, aiming at improving the response of the DBE to users needs. In the long term, the EvE will promote the evolution of the business ecosystem, the most useful and requested services will spread over the DBE network, also as part of new composed services, becoming more and more connected and used[2].

Some key usage scenarios of the Digital Business Ecosystems are extensively described in [WP21DEL21.1], in chapter 7 and following. The simplest case is of a single user who searches for a service, he or she finds it and uses it. Starting from this plain "search and consume" process a more complicated "search-negotiate-agree-consume" scenario could be reached, in which SMEs offer and consume services integrating them in their Information Technology (IT) systems, composing them to satisfy complex needs, and carrying on negotiations to establish long term business partnerships. To support these wide range of utilization scenarios the DBE offers a rich set of tools in each of its operating environments. In the SF, for example, the Studio comprehends Business Model Language Editor, Service Definition Language Editor, Service Manifest (SM) Editor, BML Data Editor, and other structural services such as the Model Repository. The execution environment primarily consists of the Servent, the Semantic Registry and the Recommender; however, other components will also be described in this deliverable.

2 For a more complete description of these environments the reader can refer to [WP21DEL21.2].

In [WP21DEL21.2] the main architectural viewpoints of the DBE are described: Structural, Technical and Functional Viewpoints. In [WP21DEL21.1] every viewpoint is introduced with its characteristics and goals. This document is aimed at modelling the interaction between the DBE environments' components, and might be regarded as a specialization of the Structural Viewpoint, in which every component has been identified and every interaction traced. The content of this deliverable is not intended to be a substitute for other documents related to the structural architecture of the DBE, rather it should be seen rather as complementary information that adds an overall integrated perspective focusing on the collaboration between components.

The approach used in modelling divides the DBE components into Eclipse plug-ins and DBE Services[3], and every interaction type in the model will be characterized by the source and the destination component type. For example, one of these interaction categories will be the "plug-in to DBE Service communication", which includes the interaction between the BML Editor and the Model Repository of the Service Factory Environment.

3 The components types will be defined in the next chapters of the document.

3 Interaction Logical View

The goal of the Digital Business Ecosystem is to become a virtual environment where consumers can meet producers and do business with them. Therefore, the fundamental tools that DBE must provide, are the ones necessary to define and publish services, and those useful for locating and consuming offered services. To simplify the drafting of the interaction model the whole DBE project space will be partitioned into three viewpoints inherited from the architectural specification ([WP21DEL21.1]): the Service Factory Environment, the Execution Environment and the Evolutionary Environment.

Sections 3.1-Service Factory and 3.2-Execution Environment introduce the model diagram with a description of the two main functional areas involved. The model depicted as a UML Component Diagram is commented in paragraph 3.3.1-UML Interoperability Reference Model Description.

3.1 Service Factory

The Service Factory could be modelled taking into account two main structural and functional areas, the DBE Studio and the Model Repository.

The **DBE Studio** is an Integrated Development Environment (IDE) for the Digital Business Ecosystem. It is built on top of Eclipse [ECLIPSE]. This platform provides an extensible pluggable architecture which supports an evolving developing environment. For a comprehensive documentation about DBE Studio and its environment refer to [WP26DEL26.3]. Because of the nature of this pluggable software components, a dedicated stereotype has been introduced in the UML interaction reference model of the DBE: <<EP Component>> (Eclipse plug-in Component). Therefore in the model, every tool constructed and deployed as an Eclipse plug-in will be labelled with the <<EP Component>> stereotype, and in the documentation such a component will be generically referred as an Eclipse plug-in component. Referring to Figure 1: DBE Components UML Interoperability Reference Model, every component marked with the stereotype <<EP Component>> is to be considered as part of DBE Studio; the Service Factory Environment is represented by an UML package entity which contains both the DBE Studio and the Model Repository service.

The other main structural area of the Service Factory is the **Model Repository** (MR), which aims at providing a common and consistent repository for all the models. In [WP14-DBEKB] the Model Repository is described as a decentralized distributed Knowledge implemented that makes use of peer 2 peer protocols and technologies. It is a pivotal DBE Service belonging to the Service Factory Environment: many models managed by DBE Studio (BML, SDL, SSL, ...)[4] are persisted into it.

With regards to the interaction modelling, another stereotype has been introduced: <<DBE Service>>. Every software module or software component with a defined responsibility and a mechanism to accept and serve requests will be referred to as a DBE

4 See the glossary for the description of the acronyms

Service, while in the model specification it is formally tagged with the <<DBE Service>> stereotype.

3.2 Execution Environment

The DBE Service Execution Environment is where actual services are registered, deployed, searched, retrieved and consumed. As documented in [WP21DEL21.2], the ExE is the actual DBE Network, and to avoid the presence of single points of failure or control, it is implemented as a P2P network, where many instances of single ExE nodes contribute to the whole DBE P2P system, it is ultimately perceived as a single unique environment. The DBE ExE is not supposed to replace the back-end systems of the SMEs, but rather to act as an Internet-based service bus which enable consumers and publishers of the services to interact with each other. Independently from the architectural and technical characteristics of the ExE, the focus of the current document is to model the ExE environment from the "components and interactions" point of view. In any event, some specific behaviours related to the P2P topology of the DBE Network will be taken into account, including, for example, the Nervous System interactions because of their role in the UML interaction model. Every detail about P2P architecture will be found in [WP21DEL21.2]. Referring to Figure 1: DBE Components UML Interoperability Reference Model the Execution Environment is depicted as a UML package which contains pertinent DBE Services.

3.3 Overall Interaction Model

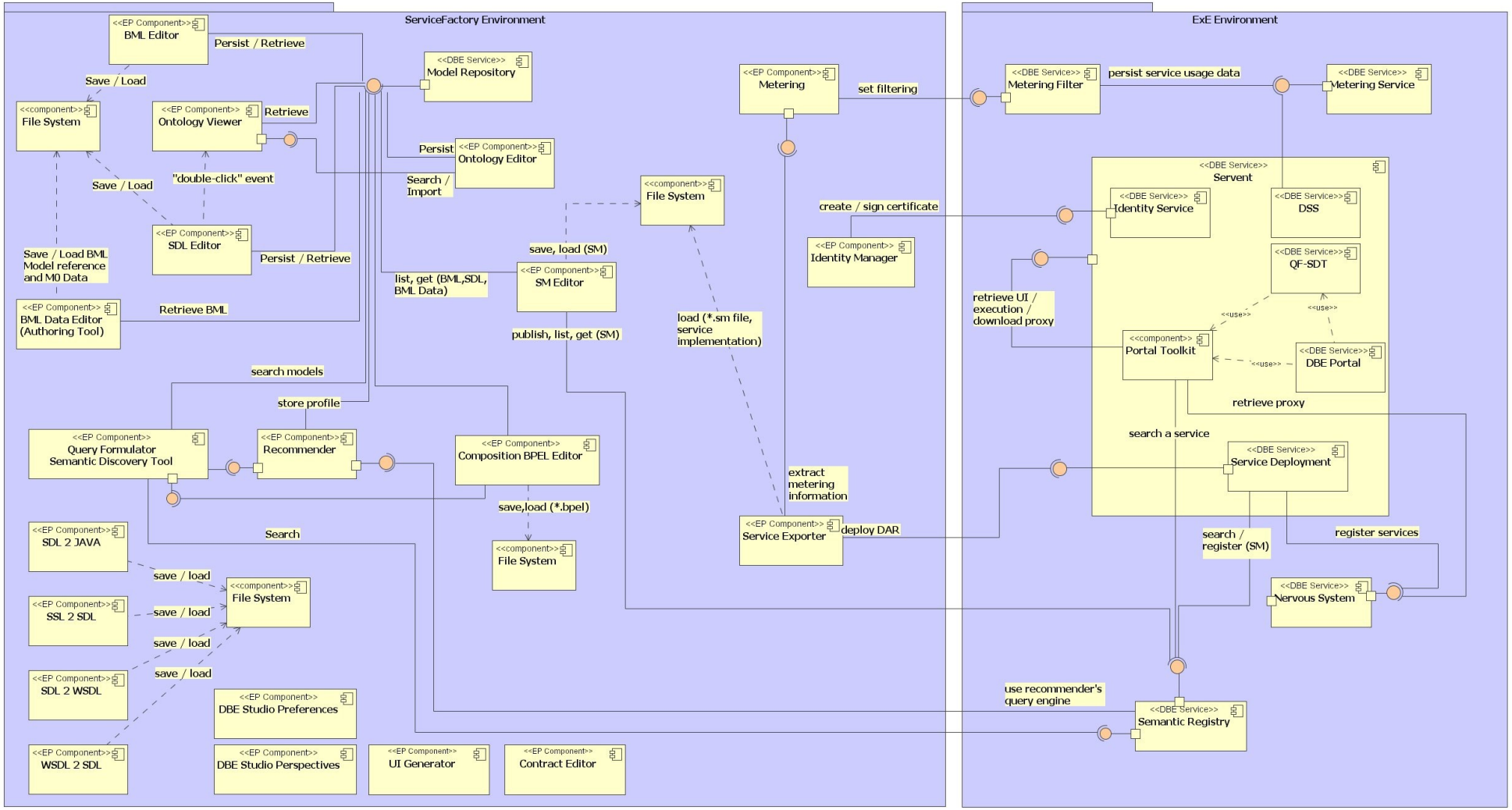

Figure 1: DBE Components UML Interoperability Reference Model

3.3.1 UML Interoperability Reference Model Description

This section describes the UML model of Figure 1: DBE Components UML Interoperability Reference Model. It aims to support reading of the diagram, recalling some of the activities with the DBE to better understand interactions between components. However, it is not intended as a substitute for the UML model which itself remains the actual reference.

In the service modelling process, the first components involved in sequence are the BML Editor and the SDL Editor.

The **BML Editor** provides a set of interfaces for the creation of a BML Model. The model can be written from scratch or retrieved from the Model Repository (MR) and then modified. If required, a transient model can be persisted into the MR while temporary models can be saved and read in the file system. From the viewpoint of the business analyst[5], the modelling activity is supported by the Ontology Viewer Eclipse plug-in component which retrieves the ontology concepts from the MR and associates them to model elements. The BML Editor can also create Semantic Service Language (SSL) models that are also persisted into the MR. Using the BML Editor it is also possible to export and import model files to and from the file system.

The **SDL Editor** provides the functionality to create the SDL that has to be associated with a business model. The SDL model can be created from scratch or generated from the SSL model. It can also be defined as an extension or a modification of a pre-existing model; in each case the interaction is between the SDL Editor and the MR. As it happens for the BML Editor, the file system can be used by the SDL Editor to store and reload temporary or archived model examples. For enriching the model it is possible to associate an ontology to the SDL; this functionality is provided through the interaction between the SDL Editor and the Ontology Viewer.

The BML describes the main characteristics of SMEs and of the business specification they support [WP15-DEL15.3]. As a refinement of the current BML, the already formalised version 2.0 of BML might be introduced in a future version of the DBE project. BML 2.0 enables DBE and BML models to the emerging SBVR - OMG's standard - for business modelling (Semantics of Business Vocabulary and Business Rules). SBVR Editor might be an Eclipse plug-in Component in a future release of DBE, that will allow service designers to model a business service via a natural language enriched model. The SBVR Editor, similarly to the other modelling tools, will interact with the Model Repository to persist and retrieve service models.

In the definition process of a service instance, there is the need to provide specific data for a given BML model [WP21-DEL21.1]. This functionality is provided by the Eclipse plug-in Component **BML Data Editor**. This tool is used to parse a given BML model and to provide a data entry system to allow the user to specify service and business related information. In accordance with the MOF levels of abstraction given in chapter 5 of [WP21DEL21.2] the BML Model fits in level M1 of the MDA abstraction layer, while the BML

5 Model designer is the DBE user that develop BML models.

Data are at level M0. To accomplish its task the BML Data Editor interacts with the MR to retrieve BML models. The component can also save and reload the M0 data related to a BML Model to and from the file system.

The **Ontology Viewer**'s GUI (Graphical User Interface), looks for semantics and scans tags to the invoking SDL editor as described previously. The Ontology Viewer Eclipse plug-in Component raises a "double click over an item" event to the DBE Studio environment, and the SDL Editor catches the event, receiving the unique ID of the double clicked item from the event data. This interaction enables the SDL Editor to associate an ontology item to the SDL model, and to perform a specific query against the ontology repository, in order to fetch extensive ontology items descriptions.

The **Ontology Editor** is an Eclipse plug-in Component that provides a GUI to manage the Ontologies that are developed and used in the DBE environment. The Ontology Editor connects to the Model Repository to save and update the ontologies, it also uses the Ontology Viewer to search a model which can be then imported and modified in the editor context.

The BML model combined with the the SDL model and instance data (such as location, name, costs, ...) builds up the "service manifest" that is the self contained descriptor of the service. The **Service Manifest (SM) Editor** creates manifests starting from the models retrieved from the MR or file system, and so there is a direct interaction between the editor and the MR. The SM Creator also interacts with the Semantic Registry[6] (of the DBE Execution Environment) to publish service manifests.

The **Metering** is an eclipse plug-in, part of the DBE Studio, that provides a "wizard-type" interface that enables a DBE Studio user to configure the metering of a service. The Metering Wizard is designed primarily for the purposes of accounting and charging; this plug-in filters/intercepts DBE Service requests and responses, to extract usage data, and to send it to a DBE Metering Service for persistence. In addition, Metering provided operations are also used by the DBE Service Exporter Wizard.

The **Service Exporter** Eclipse plug-in enables a user to export a DBE project and deploy it to the Servent. Using wizards the user can add/edit the service deployment information. The tool creates a DBE Archive file (DAR) which contains a particular structure for deployment within a Servent. This plug-in also uses the Metering Wizard interface to allow users to add metering information to the service at deployment time.

The **Composition BPEL Editor** (Manual Composer Tool) provides the functionality to chain business services together to obtain a service-composition that can readily be published and executed as an atomic service [WP21-DEL21.3] [Gioldatis et al,2004]. Composition BPEL Editor interacts with the Query Formulator-Semantic Discovery Tool in order to discover candidate constituent services and to publish the newly created (composite) services. It also interacts with the Model Repository Service in order to store/retrieve/update Orchestration Specifications and interacts with File System in order to store/retrieve/update Composite Services (*.bpel files).

6 See the paragraph corresponding to the DBE Execution Environment in this document.

The **UI Generator** is an Eclipse plug-in which uses an SDL model to create a UI. It uses the SDL model that is present in the user's project to generate Open Laszlo[7] code, which contains the "boiler plate" code to perform remote invocations on DBE services via the Portal. Internally, the UI Generator performs its activity using XSLT and supports various user interface pattern implementations, some of which are distributed with the DBE Portal. The generated UIs are intended to test the UIs and also to support developers who can further adjust his or her own presentation needs.

The **Contract Editor** tool (and other strictly related tools) aims at creating and manipulating a standard format for legal contracts in a Digital Business Ecosystem.

The DBE Studio enables the user to search for models through the **Query Formulator-Semantic Discovery Tool**. Using this plug-in the user is able to define queries for existing model instances by posing constraints on the attributes of primitives provided by SSL and BML's metamodel.

The **Recommender** is a tool that assists the SMEs by providing partnership recommendations and suggestion. The Recommender Service acts as an autonomous process that manages SME preferences (either business preferences or service preferences) and matches theses preferences with available business descriptions and service descriptions. The Recommender exploits the Model Repository to store preferences; it uses a query language (the Query Definition Metamodel) to express the matching criteria.

SDL 2 Java Eclipse plug-in, **SSL 2 SDL** Eclipse plug-in, **SDL 2 WSDL** Eclipse plug-in and **WSDL 2 SDL** Eclipse plug-in are DBE Studio components aimed at model transformation. They all use the file system to read and store models.

The Eclipse **Perspectives** declares an arrangement of a set of related views and editors which are combined together to perform specific tasks. The DBE Studio Perspectives component don't interact directly with other DBE components.

The **Preferences** Eclipse plug-in Component provides an entry in the Eclipse's workspace preferences menu for global DBE Studio preferences. The DBE Studio Preferences component doesn't interact directly with other DBE components.

A DBE Service must be published in the **Semantic Registry** before it can be found in the ExE. A Service Manifest, produced by the Service Factory, completely describes a service, and it is published via the Semantic Registry so that consumers can find and use it.

Once a service is coded and available to be used, a proxy for the service must be created. The proxy is the means by which a client could actually invoke the service, despite where the back-end IT system hosting it resides. The middleware in the DBE that is in charge of storing and distributing the service proxies and the endpoints of the services is called the **Nervous System** (currently implemented with the FADA framework, see [WP21DEL21.2]). In the Interoperability Reference Model, the Nervous System can be

7 A programming language that produces applications runnable in Internet browsers enhanced with the Macromedia Flash plug-in (www.openlaszlo.org)

seen as a DBE Service, in fact it exposes a black box interface that supports proxies registration and searching.

The **Servent** (SERVer & cliENT, [WP21DEL21.2]) is the software component which contains and executes services, both infrastructural and custom made. For example the proxies and the adapters[8] of the published services are executed in the Servent.

The **DBE Portal** is a DBE Service that is deployed and executed on top of a Servent. The DBE Portal is an entry point to the DBE that provides the means to search, browse and execute DBE Services over the DBE P2P network using a web browser [WP26DEL26.6]. Nevertheless the DBE Portal is not a central access point or single point of failure in the architecture. Each DBE node running on a Servent can and shall host a Portal. The underlying design and implementation allow users to access any Portal and gather information in a consistent way. Any DBE Portal can be used to leverage the DBE, there is not a root Portal or a preference node, the are all equipotential and enforced to be stateless.

The DBE provides the functionality to configure the metering of a service before it is deployed in the DBE. The generic metering functionality is provided as part of the DBE Execution environment as a **Metering Filter** and a **Metering Service**. The purpose of the Metering Filter is to filter/intercept DBE Service requests and responses, to extract usage data, and to send it to the DBE Metering Service for persistence. The Metering Wizard plug-in allows the user to specify the Service Method and Method Parameters to be metered and persisted as usage data. Method and Parameter details are extracted from the SDL description of the service contained in the Service Manifest of the service.

Many other components are part of ExE Environment and their characteristics and behaviour are documented in the deliverables reported in the paragraph 8-References. Any other interaction detail and topology, not mentioned in this paragraph, is specified in the UML diagram reported in Figure 1: DBE Components UML Interoperability Reference Model.

3.4 Other Components and Future Implementations

The **Examples** plug-in, which is delivered separately to the DBE Studio distribution, provides DBE projects containing example services; they don't interact directly with other DBE components.

The **User Profiling** Eclipse plug-in gathers data from the user's implicit and explicit activities[WP7-DEL7.1]. Via the user profiling information it shall be possible to support personalised recommendation in order to meet the customer's needs more effectively and efficiently making interactions faster and easier.

8 See the glossary at the “Servent” entry

4 “Plug-in to plug-in” Interaction Types in Eclipse

DBE Studio pulls together a number of plug-ins [WP26DEL26.3], as described in the Service Factory Environment interaction model of Figure 1: DBE Components UML Interoperability Reference Model. Every plug-in is natively designed to be integrated in the Eclipse IDE environment: there are specific APIs for communicating between the plug-in and the Eclipse container. In this chapter the plug-in to plug-in communication techniques developed are described.

This mechanism is not present in the Eclipse IDE and it represents an original work of the DBE project.

4.1 Mouse double-click

An Eclipse plug-in to plug-in interaction mechanism in DBE Studio has been designed and developed. It is employed between the SDL Editor and the Ontology Viewer, where the former of the two Eclipse plug-in components catches a “double click” event raised by the Ontology Viewer.

The “raise and catch event” mechanism is implemented through Java classes: the Ontology Viewer package defines an interface Java object that raises the event through a method call, the SDL Editor package defines a class that implements the exposed interface. In such integration schema, every event raised from Ontology Viewer is translated into a method call to the SDL Editor package. In this way, the SDL Editor receives the unique ID of the “double clicked item” of the ontology, and the proper handle action is taken (see Figure 2: Ontology Viewer to SDL Editor Communication (“plug-in to plug-in”), it should be noted that the names of the classes may vary in the future).

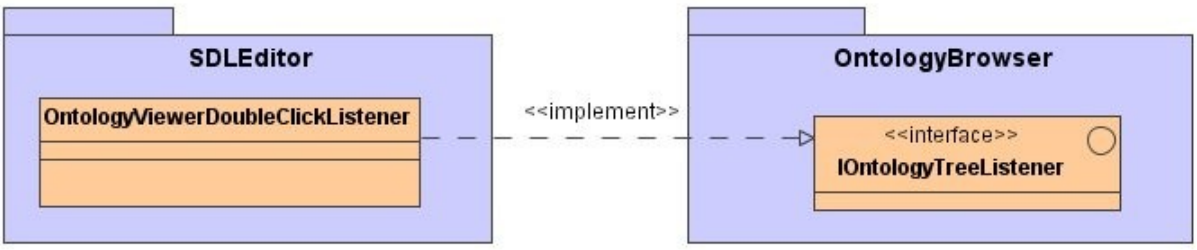

Figure 2: Ontology Viewer to SDL Editor Communication (“plug-in to plug-in”)

Refer to [WP26DEL26.1], Table 1, chapter 2, for a comprehensive list of all Eclipse plug-ins and related development teams.

4.2 Drag&Drop

Note: For sections 4.2 to 4.7, please refer to [ECLIPSE – Articles]

Another way to implement a plug-in to plug-in communication is through a Drag&Drop mechanism. This communication should be understood as a way of transferring information from one plug-in to another by the means of the Eclipse platform views and

editors. For example you can drag an object from a view of one plug-in and drop it into a view or editor of another plug-in. To make this possible you should "enable" the source plug-in view to allow dragging and the destination plug-in to allow dropping.

In keeping with the general philosophy of Jface[9], the Drag&Drop adds a layer of functionality on top of the SWT[10] Drag&Drop support. This layer allows the developer to deal directly with domain objects (such as resources, tasks, etc.), without having to worry about the underlying window controls. Rather than concealing or replacing the Drag&Drop support in SWT, the JFace Drag&Drop support works as an extension to the same concepts found in SWT Drag&Drop.

In a Drag&Drop collaboration scheme two parts are involved: the drag source from where you drag the object and the drop target where you drop it.

A drag source is the provider of data in a Drag&Drop data transfer as well as the originator of the Drag&Drop operation. The data provided by the drag source may be transferred to another location in the same widget, to a different widget within the same application (e.g. to a view/editor of another plug-in), or to a different application altogether. For example, you can drag an object from your application and drop it in a file, or you could drag an item from a tree and drop it below a different node in the same tree.

4.3 Adding JFace viewer drag support

Adding drag support to a viewer means that it enables the user to select any item in the viewer with the mouse, and drag it into another viewer or another application. Drag support can be added to any subclass of org.eclipse.jface.viewers.StructuredViewer using the addDragSupport(int, Transfer[], DragSourceListener) method. Here is the code for associating a drag listener with a tree viewer:

```
TreeViewer yourViewer = new TreeViewer(...);
int ops = DND.DROP_COPY | DND.DROP_MOVE;
Transfer[] transfers = new Transfer[] { YourTransfer.getInstance()};
viewer.addDragSupport(ops, transfers, new YourDragListener(viewer));
```

The addDragSupport is a method of the StructuredView class, which creates a DragSource object and adds the drag listener. Here it is the code of this method:

```
Control yourControl = getControl();
final DragSource dragSource = new DragSource(yourControl, operations);
dragSource.setTransfer(transferTypes);
dragSource.addDragListener(listener);
```

YourDragListener is an implementation of the SWT interface org.eclipse.swt.dnd.DragSourceListener.

YourTransfer can handle any type of transfer as we'll see a little later.

9 http://wiki.eclipse.org/index.php/JFace
10 http://www.eclipse.org/swt/

4.4 Adding view drop support

Drop support can be added to viewers using the StructuredViewer.addDropSupport(int, Transfer[], DropTargetListener) method. This allows your viewer to be the target of a drop operation. The code for adding drop support is almost the same as for adding drag support:

```
TreeViewer yourViewer = new TreeViewer(...);
int ops = DND.DROP_COPY | DND.DROP_MOVE;
Transfer[] transfers = new Transfer[] { YourTransfer.getInstance()};
viewer.addDropSupport(ops, transfers, new YourViewerDropAdapter(viewer));
```

Again, the addDropSupport creates the target object and adds to it the drop listener. JFace provides a standard implementation of DropTargetListener called org.eclipse.jface.viewers.ViewerDropAdapter. This adapter makes it easy to add drop support for simple cases. If you have more complex requirements, you can always override the SWT DropTargetListener interface directly for ultimate flexibility. When sub-classing ViewerDropAdapter, simply implement its two abstract methods: validateDrop(Object target, int operation, TransferData transferType), and performDrop(Object data).

validateDrop is called whenever the user moves over a new item in your viewer, or changes the drop type with one of the modifier keys. The method provides the current drop target, operation, and transfer type. The return value of this method indicates whether a drop at the current location is valid or not. A return value of false will change the drag icon to indicate to the user that it is illegal to drop what they are dragging at the current location. If you have more complex validation requirements based on the target object, you can do that here. For example, in a file navigator, you may want to allow dropping on top of directories, but not on top of files.

performDrop is called when the user lets go of the mouse button, indicating that they want the drop to occur. Your implementation should accordingly perform the expected behaviour for that drop. Context for the drop is provided by the methods getCurrentTarget, getCurrentOperation, and getCurrentLocation on ViewerDropAdapter. Most importantly, at the time when performDrop is called, getCurrentTarget will provide the object in your viewer that is currently under the mouse.

4.5 Plug-in drop handling

Due to the UI layering imposed by the plug-in mechanism, viewers are often not aware of the content and nature of other viewers. This can make Drag&Drop operations between plug-ins difficult. For example, our plug-in may want to allow the user to drop objects into the Navigator view. Since the Navigator view doesn't know anything about our objects (the Navigator only displays org.eclipse.core.resources.IResource objects), it would not be able to support this. To address this problem, a plug-in drop support mechanism is provided by the workbench. This mechanism essentially delegates the drop behaviour back to the originator of the drag operation. Here are the steps required to add Drag&Drop behaviour using this mechanism:

Step 1) In your plug-in.xml, define an extension on the "org.eclipse.ui.dropActions" extension point. Here is an example XML declaration:

```
	<extension
		id="SdlDrop"
		name="SDL Editor Drop"
		point="org.eclipse.ui.dropActions">
			<action
				class="org.dbe.sdl.presentation.SdlEditorplug-
inDropAdapter"
				id="org.dbe.sdl.presentation.sdlEditorDrop">
			</action>
	</extension>
```

Step 2) Implement the code that will perform the drop. This work is done by the class defined in the extension markup above, which must implement org.eclipse.ui.part.IDropActionDelegate. This interface defines a single run() method that gets called when the drop occurs. The run method is supplied with the object being dragged, as well as the object under the cursor when the drop occurs. Here is an example implementation of the drop delegate:

```
public boolean run(Object source, Object target) {
	if (source instanceof byte[] && target instanceof IFile) {
		IFile file = (IFile) target;

		if (file.getFileExtension().equals("txt")) {
			try {
				file.appendContents(new ByteArrayInputStream((byte[])
source), false, true, null);
			}
			catch (CoreException e) {
				System.out.println("Exception in SdlEditor drop dapter"
+ e.getStatus().getMessage());

				return false;
			}

			return true;
		}
	}

	return false;
}
```

Step 3) In the viewer that will be the source of the Drag&Drop, add drag support using the StructuredViewer.addDragSupport() method described earlier. In the array of supported transfer types, include the singleton instance of org.eclipse.ui.part.plug-inTransfer. In your implementation of DragSourceListener, the dragSetData method must set the data to be an instance of org.eclipse.ui.part.plug-inTransferData. This object consists of the id of your drop action, along with the data being transferred. Here is the code in the drag listener from the gadget example:

```
public void dragSetData(DragSourceEvent event) {
		IStructuredSelection selection =
(IStructuredSelection)viewer.getSelection();

		if(plugin Transfer.getInstance().isSupportedType(event.dataType)) {
			byte [] data =
```

```
YourTransfer.getInstance().toByteArray(selection);
                    event.data = new plugin
TransferData("org.dbe.sdl.presentation.sdlEditorDrop", data);
              }
      }
```

Step 4) In the viewer that will receive the drop, the drop listener for that viewer must subclass org.eclipse.ui.part.plug-inDropAdapter, which in turn subclasses ViewerDropAdapter as described earlier. Be sure to invoke the super methods for validateDrop and performDrop in cases where your adapter does not understand the transfer type. Following from our earlier example, the viewer's declaration would look like this:

```
TreeViewer yourViewer = new TreeViewer(...);
      int ops = DND.DROP_COPY | DND.DROP_MOVE;
      Transfer[] transfers = new Transfer[] { YourTransfer.getInstance(), plug-
inTransfer.getInstance()};
      viewer.addDropSupport(ops, transfers, new YourDropAdapter(viewer));
```

The basic Workbench views such as the Navigator view already have this support added. It is recommended that anyone defining their own views should add the plug-in support, in anticipation of future third party plug-ins wanting to drop content to their views. For more information about this advanced Drag&Drop mechanism, refer to the documentation for the org.eclipse.ui.dropActions extension point.

4.6 Transfer

Transfer is an abstract class that provides a mechanism for converting between a Java representation of data and a platform specific representation of data and vice versa. The Java representation of the data is what the application uses. For example, text is represented by a String object. The platform specific representation is what the operating system uses and is represented in SWT by the TransferData object.

Table 1 shows the subclasses of Transfer provided in org.eclipse.swt.dnd.

The TransferData class contains public fields that are platform-specific. Because the fields in TransferData vary from platform to platform, applications should not access them. The purpose of making the fields public is to allow developers to extend the Transfer class and provide additional platform specific types for data transfer (e.g., bitmap images or wave files).

The notion of transfer types is central to the Drag&Drop support in Eclipse-based UIs. Transfer types allow drag sources to specify what kinds of object they allow to be dragged out of their widget, and they allow drop targets to specify what kinds of objects they are willing to receive. For each transfer type, there is a subclass of org.eclipse.swt.dnd.Transfer. These subclasses implement the marshalling behaviour that converts between objects and bytes, allowing Drag&Drop transfers between applications. The following table summarizes the transfer types provided by the basic Eclipse platform, along with the object they are capable of transferring:

Transfer class	***Object it transfers***
org.eclipse.swt.dnd.FileTransfer	java.lang.String [] (list of absolute paths)
org.eclipse.swt.dnd.RTFTransfer	java.lang.String (may contain RFT formatting characters)
org.eclipse.swt.dnd.TextTransfer	java.lang.String
org.eclipse.swt.dnd.MarkerTransfer	org.eclipse.core.resources.IMarker []
org.eclipse.swt.dnd.ResourceTransfer	org.eclipse.core.resources.IResource []
org.eclipse.swt.dnd.EditorInputTransfer	org.eclipse.ui.part.EditorInputTransfer.EditorInputData []
org.eclipse.swt.dnd.plug-inTransfer	org.eclipse.ui.part.plug-inTransferData

Table 1: Transfer subclassing provided by org.eclipse.swt.dnd

The set of transfer types is open ended, because third party tool writers can implement their own transfer types for their domain objects. To implement your own transfer type, it is recommended that you subclass org.eclipse.swt.dnd.ByteArrayTransfer.

4.7 Transfer types supported by the standard views

Many of the basic views you see in Eclipse already support various transfer types. It is important to understand what transfer types are supported by each view, because this dictates how the Drag&Drop support in your view will interact with other basic views found in the Eclipse platform UI.

The Navigator view supports dragging and dropping files (FileTransfer), and resources (ResourceTransfer). For example, you can drag a file from the Navigator view into Windows Explorer or the Windows Desktop. Similarly, you can import resources into Eclipse simply by dragging them from Windows into the Navigator view of your workbench. You can also drag files between two instances of Eclipse, or drag within a single Navigator to copy and move files within your workspace. If your view supports either FileTransfer or ResourceTransfer, then users will be able to transfer resources between your view and the Navigator view.

The Tasks and Bookmarks views support dragging of markers (MarkerTransfer). Dragging a selection of tasks from the Tasks view into an application such as MS Word will generate a textual marker report (TextTransfer). You can also drag markers out of the Tasks and Bookmarks views into other parts of the workbench, such as the editor area. Dragging a marker to the editor area will open the associated resource in the editor and jump to that marker location in the editor.

Finally, the editor area supports dropping of editor inputs (EditorInputTransfer), resources, or markers. Dragging these objects to the editor will cause it to locate and open an appropriate editor for the given resource, editor input or marker. In the case of markers, it will also jump to that marker location in the editor.

4.7.1 Cut, Copy and Paste

Cut and paste can be thought of as the keyboard equivalent of Drag&Drop. Once you've mastered Drag&Drop support, you'll find that cut and paste is a snap. Here is code for

adding cut and paste support from within an Eclipse view (this code goes in the view's createPartControl method):

```
      public void createPartControl(Composite parent) {
      viewer = new TreeViewer(parent, SWT.MULTI | SWT.H_SCROLL | SWT.V_SCROLL);

      //...initialize viewer's content and label providers...

      clipboard = new Clipboard(getSite().getShell().getDisplay());
      IActionBars bars = getViewSite().getActionBars();
      bars.setGlobalActionHandler(
      IworkbenchActionConstants.CUT, new YourCutAction(viewer, clipboard));
      bars.setGlobalActionHandler(ActionFactory.COPY.getId(), new
YourCopyAction(viewer, clipboard));
      bars.setGlobalActionHandler(ActionFactory.PASTE.getId(), new
YourPasteAction(viewer, clipboard));
}
```

This code simply creates a new SWT clipboard, and then defines global actions for cut, copy, and paste using that clipboard. The IActionBars interface is used for hooking into global actions. Note that SWT clipboard objects are operating system resources that must be disposed when no longer needed. We dispose() the clipboard when the view is disposed. Disposing of an SWT clipboard instance does not remove the data from the operating system's clipboard.

The actions that are provided as global action handlers should be subclasses of the org.eclipse.jface.action.Action class. The code for these actions is similar to the Drag&Drop code, except that they use the clipboard as the transfer mechanism, rather than the Drag&Drop event handlers. Here is the code for the run method of the copy action:

```
public void run() {
      IStructuredSelection selection =
(IStructuredSelection)viewer.getSelection();

      Object obj = selection.getFirstElement();

      if(obj != null){
            Clipboard cb = new Clipboard(display);
            String data = obj.toString();

            cb.setContents(new Object [] {data}, new Transfer []
{TextTransfer.getInstance()});
            cb.dispose();
      }
}
```

4.7.2 Launching a plug-in from another plug-in

Let's imagine a simple example: in the SM-Creator editor you have the PIMModel-SDL_Model field and you want to open this model in the SDL Editor to edit and save it. The main issue is how to call the SDL Editor to show up. The key-line is this:

```
IEditorPart page.openEditor(IEditorInput arg0, String arg1)
```

We call the *openEditor* method on a *page* object (of type *org.eclipse.ui.IWorkbenchPage*), which is the active page of the workbench.

The first argument is an IEditorInput – the file/resource to be open in the editor. In our situation there should be an IFile, and we used *new EditorInput(ifile)* (where ifile is of type IFile).

The second (String) argument is the editor's id, which we can get from: IEditorDescriptor editorDesc = IDE.*getEditorDescriptor*(ifile, **true**);. The ifile is the same as above and the boolean tells the function to determine the content type of the ifile.

The method returns an IEditorPart – the reference to the open editor. Also to mention the *PartInitException* that could be thrown by the method if anything goes wrong.

So, in a few words, what you need to open an editor is: the active workbench page, the file/resource to be opened in the editor and the editor's id.

5 Service discovery and execution

One of the architectural principles that drives the development of DBE project is the distributed software components approach[WP21-DEL21.2]. This chapter will describe the interaction between a generic DBE component (e.g. an Eclipse plug-in component, a DBE Service component, or a generic DBE software module) and a DBE Service (e.g. Nervous System, Semantic Service).

Every deployed DBE Service (either functional or structural like the Model Repository or the Semantic Registry), registers its proxy in the Nervous System. A service proxy is serialized Java executable code that actually invokes the back-end part of the service, in a transparent way from the caller point of view, independently from the actual network position of the service [WP21-DEL21.2]. The interaction pattern between a generic component and a DBE Service is depicted in Figure 3: Interaction between a generic component and a DBE service:

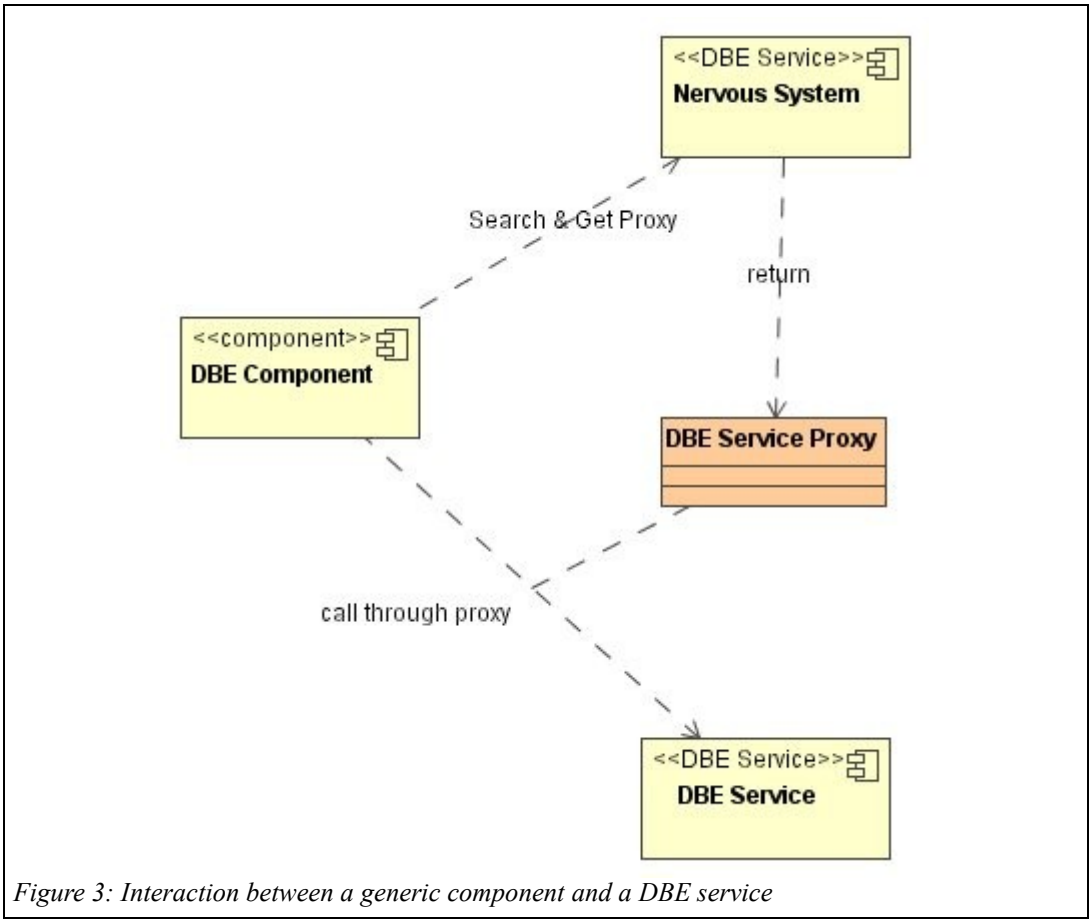

Figure 3: Interaction between a generic component and a DBE service

To establish communication with a DBE Service, the first action to take is to search and download the service proxy, through a call to the DBE Nervous System. A proxy object is returned from this call, and an effective call to the DBE Service endpoint can be made through the proxy: every available DBE Service operation is mapped by the proxy. Every

call to a proxy's method is forwarded to the DBE Service, which is actually deployed in one of the DBE service nodes.

All this complexity is hidden from the service consumer as well as from actual developers because the Servent offers a set of helpers APIs which abstract from the underling network implementation technology. As such, the existence of a P2P based Nervous System is actually hidden: DBE services are P2P implementation agnostic .

6 DBE Components

Term	Description
BML Editor	A tool to model the concerns of the business. The modelling task might be graphical or rule base; it is not a matter in the context of this document. What is important thought is that the output model is an instance of a BML metamodel.
Business Service	The actual SME service supported by the DBE
DBE Composer	A tool for creating a workflow using pre-existing DBE Services
DBE Portal	An infrastructural DBE service that provides a customisable web user interface into the DBE. DBE Services can be searched for and executed via this web interface.
DBE Studio	The GUI based tool that organise the different tools used in the development environment. It is the implemented development environment.
ExE	Ref. Service Execution Environment
FADA	Federated Advanced Directory Architecture, an Open Source implementation of a P2P network that has it origins in the SUN' Jini specification. Nervous System technology implementation.
Form Designer	A tool that ease the drawing/definition of a GUI form. The resulting GUI form is to be used for executing/consuming a service.
Interaction Form	The resulting output of the Form Designer. The resulting GUI form is to be used for executing/consuming a service.
KB	Ref. Knowledge Base
Knowledge Base	The Knowledge Base represent the DBE pervasive storage area that contains: models, metamodels, ontologies and anything else xml-based that is needed to be stored. The Knowledge Base is the Service Factory Environment memory storage area. The Execution Environment memory space is essentially the Semantic Registry.

Term	Description
Model Repository	The part of the Knowledge Base that stores the Models used in the DBE
Recommender	The process that given a query request, is able to inspect the Model Repository and providing a ranked list of results. The recommendation process can retrieve models or Service Manifests.
Repository	Ref. Model Repository
SDL Editor	An editor to create SDL models of a service. It will model concepts like: service name, operations, parameters, exceptions and so forth.
Semantic Service Language	Semantic Service Language, the metamodel for describing service semantics
Servent	The DBE Application Server: Serv(er)(cli)ent. It serves the objective of hosting both a service proxy (the front-end) and the adapter (back-end mediator to the service). It decouples the dependency from a P2P network.
Service DNA	It is the combination of SDL and BML model. It represent the conceptual definition of a service when it is not related to any real service. By "real" we mean a service that exist in the real world: for example the supplier of a real service has a VAT number, a snail mail address or an IP address. The binding defines the tangible relation with a real service. The Service DNA is an element of reuse across services. It is the conceptual definition of a service when not related to any real service; it represents an element of reuse (aka *Service Definition*).
Service Execution Environment	It is where services live, where they are registered, deployed, searched, retrieved and consumed. This parallel world is sometimes referred to as the "run-time of the DBE".
Service Factory Environment	Is devoted to service definition and development. Users of the DBE will utilise this environment to describe themselves, their services and to generate software artefacts for successive implementation, integration and use.

Term	Description
Service Manifest	The container that entirely describes a service, the difference with the Service DNA being the representation of service information (M0 in MDA terms)
SFE	Ref. Service Factory Environment
SM	Ref. Service Manifest
SME Consumer	It represents an SME user that searches and consumes a service in the DBE
SME Provider	It represents an SME user that offers a service in the DBE
SME Software Developer	It represent an SME user that creates the software artefacts in order for a service to be published or consumed in the DBE
SME User	It represent a generic SME user that interacts as a service provider or service consumer
SSD	Semantic Service Description: the Model built using the SSL
SSL	Ref. Semantic Service Language
Structural Service	Support services like, Repository, Accounting, Security, Distribution, transactions...
Transactional Work Flow Manager	An application able to execute a service workflow.
TWFM	Ref. Transactional Work Flow Manager

7 Glossary

Term	Description
ASP	Application Service Provider
B2B	Business to Business
B2C	Business to Consumer
Back End	In client/server distributed computing it is a generic term that refers to the runtime part of an information system that runs remotely with respect to the client hardware
Back Office	A department that has little or no contact with customers. From the *wikipedia*: "A back office is a part of most corporations where tasks dedicated to running the company itself take place."
Basic Service	Basic Services are DBE services that pertain to the following categories: payments, information carriers. The list could increase further on.
BML	Business Modelling Languages
CIM	Computational Independent Model
CWM	Common Warehouse Metamodel
DBE	Digital Business Ecosystem
EAI	Enterprise Application Integration
EDI	Electronic Data Interchange
EDOC	Enterprise Distributed Object Component
ExE	Service Execution Environment
EvE	Evolutionary Environment
Front End	In client/server distributed computing it is a generic term that refers to the runtime part of an information system that runs locally with respect to the client hardware
GUI	A Graphical User Interface, it is a kind of UI

Term	Description
IDE	Integrated Development Environment
M0	MDA layer that references M1 based data ('Mike Phone has invoice #1221')
M1	MDA layer that references M2 based models ('customers may have invoices')
M2	MDA layer that references MOF models (e.g. UML, CWM, EDOC, BML, SDL...)
M3	MDA layer that references MOF language
MDA	Model Driven Architecture
MOF	Meta Object Facility
MR	Model Repository
OMG	Object Management Group (www.omg.org)
P2P	Peer to peer
PIM	Platform Independent Model
PKI	Public Key Infrastructure
Proxy	Executable Java object that can be distributed over the Internet. It is usually a mediator to the actual remote service that provides the required functionality.
PSM	Platform Specific Model
RUP	IBM© Rational Unified Process®, or RUP®, http://www-306.ibm.com/software/awdtools/rup/
SBVR	Semantics of Business Vocabulary and Business Rules (OMG Standard)
SDL	Service Definition Language: a language for the definition of a Platform Independent Model (PIM) of the service interface
SDNA / Service DNA	A Service DNA is the model of a service, it contains a reference to the "BML model" and "SDL model"
Service Proxy	Ref. Proxy
SM	Ref. Service Manifest
Smart Proxy	Ref. Proxy
SME	Small to Medium Enterprise

Term	Description
UDL	Universal Design Language
UI	User Interface
UML	Unified modelling Language
User Interface	Graphical User Interface
UUID	Universal Unique Identifier
VAS	Value Added Service: a service that is the aggregation of other services.
X509	It is a standard that makes it possible to identify someone or something on the Internet.
XMI	XML metadata Interchange
XML	Extensible Markup Language
XUL	XML User Interface Language

8 References

[WP7-DEL7.1] "Description of necessary information about DBE customer, to support a long term evolutionary business relationship"
[WP7DEL7.2] "Initial Description of Profiling mechanism design and rationale with respect to one or two use cases"
[WP21-Architecture] https://dev.digital-ecosystem.net/servlets/ProjectDocumentList?folderID=207&expandFolder=207&folderID=207
[WP14-DBEKB] Knowledge Base Implementation
[WP15-DEL15.3] ISUFI, "BML Framework 2nd release"
[WP17-DEL17.1] "Recommender"
[WP21DEL21.1] Pierfranco Ferronato, "WP 21: DBE Architecture Requirements, DEL 21.1 Preliminary design, usage scenarios, and architecture requirements", October 2004.
[WP26DEL26.3] "DBE Studio Integration Deliverable"
[WP26DEL26.6] "DBE Portal"
[WP21DEL21.2] Pierfranco Ferronato, "WP 21: DBE Architecture Requirements DEL 21.2:Architecture Scope Document", 2005
[WP21DEL21.3] "WP 21: DBE Architecture Requirements DEL 21.3: DBE Architecture High Level Specification".
[ECLIPSE – Articles] Drag&Drop, Copy/Paste articles:
http://www.eclipse.org/articles/Article-Workbench-DND/drag_drop.html and
http://www.eclipse.org/articles/Article-SWT-DND/DND-in-SWT.html
[ECLIPSE] Eclipse Open Source Community, http://www.eclipse.org
[Gioldatis et al,2004] Nektarios Gioldasis, Nikos Pappas, Fotis Kazasis, George Anestis, Stavros Christodoulakis, "A P2P and SOA Infrastructure for Distributed Ontology-Based Knowledge Management"
(http://www.ced.tuc.gr/Staff/Director/Publications/publ_files/C_GPKA_DLA_2004.pdf)

- end of document -

