



Data, Information and Process Integration
with Semantic Web Services

DIP

Data, Information and Process Integration with Semantic Web Services

FP6 - 507483

Deliverable

WP 8: B2B in Telecommunications

D8.6b

Contract Catalogue Prototype v2

Marc Richardson

January 5th, 2007



EXECUTIVE SUMMARY

This deliverable outlines v2 of the Contract Catalogue prototype. The aim of the prototype is to demonstrate how Semantic Web Services and the DIP architecture can overcome the problems that BT currently faces with managing contract catalogues. Explanations of the problems in this area are given in section 1.

The prototype is a WSMO Studio¹ Plug-in, that fully integrates with the WSMO Studio tool developed in WP4. It also integrates the reasoner component developed in WP1, the mediation component developed in WP5, and the composition tool developed in WP4.

This deliverable would be of interest to the other DIP case study partners in WP9 & WP10 and the tool developers in WP1 & WP4. Outside of DIP it would be useful to organisations that currently have problems managing product catalogues and the associated Operation System Support (OSS) required for fulfilment and management of products orders.

Disclaimer: The DIP Consortium is proprietary. There is no warranty for the accuracy or completeness of the information, text, graphics, links or other items contained within this material. This document represents the common view of the consortium and does not necessarily reflect the view of the individual partners.

¹ <http://www.wsmostudio.org>

Document Information

IST Project Number	FP6 – 507483	Acronym	DIP
Full title	Data, Information, and Process Integration with Semantic Web Services		
Project URL	http://dip.semanticweb.org		
Document URL			
EU Project officer	Kai Tullius		

Deliverable	Number	8.5b	Title	B2B in Telecommunications
Work package	Number	8	Title	Contract Catalogue Prototype v2

Date of delivery	Contractual	M 36	Actual	M36
Status	version. 1.1		final X	
Nature	Prototype X <input type="checkbox"/> Report <input type="checkbox"/> Dissemination <input type="checkbox"/> Ontology <input type="checkbox"/>			
Dissemination Level	Public <input type="checkbox"/> Consortium X			

Authors (Partner)	Marc Richardson (BT)			
Responsible Author	Marc Richardson		Email	Marc.richardson@bt.com
	Partner	BT	Phone	+441473609589

Abstract (for dissemination)		
Keywords	Contract Catalogue, BT Telecommunications, Semantic Web Services, Ontology	

Version Log			
Issue Date <dd-mmm-yy>	Rev No. <nnn starting 001>	Author <author name>	Change <Description of the changes that were made to the preceding revision>
05-01-2007	1.1	Marc Richardson	Final Version for Submission

Reviewer			
	Stephan Grimm	Email	Grimm@fzi.de
	Partner	FZI	Phone

	John Domingue	Email	j.b.domingue@open.ac.uk
	Partner OU	Phone	

Project Consortium Information

Partner	Acronym	Contact
National University of Ireland Galway	NUIG  National University of Ireland, Galway <i>Ollscoil na hÉireann, Gaillimh</i>	Dr. Sigurd Harand Digital Enterprise Research Institute (DERI) National University of Ireland, Galway Galway Ireland Email: sigurd.harand@deri.org Tel: +353 91 495112
Fundacion De La Innovacion.Bankinter	Bankinter  .com	Monica Martinez Montes Fundacion de la Innovacion. BankInter Paseo Castellana, 29 28046 Madrid, Spain Email: mmtnez@bankinter.es Tel: 916234238
British Telecommunications Plc.	BT 	Dr John Davies BT Exact (Orion Floor 5 pp12) Adastral Park Martlesham Ipswich IP5 3RE, United Kingdom Email: john.nj.davies@bt.com Tel: +44 1473 609583
Swiss Federal Institute of Technology, Lausanne	EPFL 	Prof. Karl Aberer Distributed Information Systems Laboratory École Polytechnique Fédérale de Lausanne Bât. PSE-A 1015 Lausanne, Switzerland Email : Karl.Aberer@epfl.ch Tel: +41 21 693 4679
Essex County Council	Essex  Essex County Council	Mary Rowlett, Essex County Council PO Box 11, County Hall, Duke Street Chelmsford, Essex, CM1 1LX United Kingdom. Email: maryr@essexcc.gov.uk Tel: +44 (0)1245 436524
Forschungszentrum Informatik	FZI  FZI	Andreas Abecker Forschungszentrum Informatik Haid-und-Neu Strasse 10-14 76131 Karlsruhe Germany Email: abecker@fzi.de Tel: +49 721 9654 0
Institut für Informatik, Leopold-Franzens Universität Innsbruck	UIBK  universität innsbruck	Prof. Dieter Fensel Institute of computer science University of Innsbruck Technikerstr. 25 A-6020 Innsbruck, Austria Email: dieter.fensel@deri.org Tel: +43 512 5076485

Partner	Acronym	Contact
ILOG SA	ILOG  Changing the rules of business	Christian de Sainte Marie 9 Rue de Verdun, 94253 Gentilly, France Email: csma@ilog.fr Tel: +33 1 49082981
inubit AG	Inubit 	Torsten Schmale inubit AG Lützowstraße 105-106 D-10785 Berlin Germany Email: ts@inubit.com Tel: +49 30726112 0
Intelligent Software Components, S.A.	iSOCO 	Dr. V. Richard Benjamins, Director R&D Intelligent Software Components, S.A. Pedro de Valdivia 10 28006 Madrid, Spain Email: rbenjamins@isoco.com Tel. +34 913 349 797
MDR Partners	MDR 	Rob Davies MDR Partners 8 St. Andrew Street Hertford, Herts. United Kingdom, SG14 1JA, Email: rob.davies@mdrpartners.com +44 (0)208 8763121
Hanival Internet Services GmbH	HANIVAL 	Alexander Wahler Hanival Internet Services GmbH Kirchengasse 13/1a A-1070 Wien Email: wahler@niwa.at Tel: +43(0)1 3195843-11
The Open University	OU 	Dr. John Domingue Knowledge Media Institute The Open University, Walton Hall Milton Keynes, MK7 6AA United Kingdom Email: j.b.domingue@open.ac.uk Tel.: +44 1908 655014
SAP AG	SAP 	Dr. Elmar Dörner SAP Research, CEC Karlsruhe SAP AG Vincenz-Priessnitz-Str. 1 76131 Karlsruhe, Germany Email: elmar.dorner@sap.com Tel: +49 721 6902 31
Partner	Acronym	Contact




<p>Sirma AI Ltd.</p>	<p>Sirma</p>  <p>Ontotext Knowledge and Language Engineering Lab of Sirma</p>	<p>Atanas Kiryakov, Ontotext Lab, - Sirma AI EAD Office Express IT Centre, 3rd Floor 135 Tzarigradsko Chausse Sofia 1784, Bulgaria Email: atanas.kiryakov@sirma.bg Tel.: +359 2 9768 303</p>
<p>Unicorn Solution Ltd.</p>	<p>Unicorn</p> 	<p>Jeff Eisenberg Unicorn Solutions Ltd, Malcha Technology Park 1 Jerusalem 96951 Israel Email: Jeff.Eisenberg@unicorn.com Tel.: +972 2 6491111</p>
<p>Vrije Universiteit Brussel</p>	<p>VUB</p>  <p>Vrije Universiteit Brussel</p>	<p>Pieter De Leenheer Starlab- VUB Vrije Universiteit Brussel Pleinlaan 2, G-10 1050 Brussel ,Belgium Email: Pieter.De.Leenheer@vub.ac.be Tel.: +32 (0) 2 629 3749</p>

TABLE OF CONTENTS

EXECUTIVE SUMMARY	I
TABLE OF CONTENTS	VII
1 INTRODUCTION	1
2 BACKGROUND AND MOTIVATION	2
2.1 Aims of Prototype.....	3
2.2 Further Reading	3
3 FEATURES OF PROTOTYPE v2	5
4 INSTALLATION GUIDELINES	9
5 EXAMPLE WALKTHROUGH	11
6 CONCLUSION	23

FIGURES

Figure 1. Loading the Contact Catalogue.....	12
Figure 2. Creating a new Bundle	13
Figure 3. Adding a product to the Bundle	15
Figure 4. Reasoner finds an inconsistency in the Bundle.....	16
Figure 5. Performing queries on the Bundle	17
Figure 6. Goal generated for product Bundle.....	18
Figure 7. Loading the skeleton composition goal.....	19
Figure 8. Designing the Composition Goal	20
Figure 9. Running the Goal Orientated Composition Plug-in.....	22

1 INTRODUCTION

BT Global Services is engaged in a number of large contracts to provide complete Information and Communication Technology (ICT) solutions to an organisation. These range from desktop products and software used directly by the employees, to the core network infrastructures which provide vital services such as telephony, internet/intranet access and private leased lines. The challenge in providing such a broad range of products is to manage them in such a way that ordering and provisioning is coherent and unified across the range. In practice the products come from a large number of areas across BT as well as third party suppliers such as HP. The set of ICT products and services that are offered to the customer in a particular contract is known as the contract catalogue.

The specific problems associated with managing the contract catalogue are:

- Arranging and managing a set of products from many sources to present a consistent view to the customer
- Producing product ‘bundles’ that may have constraints or dependencies between them.
- Identifying dependencies that arise due to existing products and services (i.e. the inventory).
- Representing dependencies as rules and propagating them between the various levels of service offerings.
- Interfacing with the third party suppliers for order querying, processing and fulfilment
- Managing the work flow of complex products

Some of the problems with integration and management can be tackled by introducing Web Services as a means to provide a standardised programmatic means to interface the many systems and suppliers. However, Web Services alone do not solve all of the problems with the management and use of a contract catalogue. Although Web Services allow a more coherent approach to integration by standardizing the method for interfacing, they do not provide a standardized model for the way data is represented.

Another challenge of managing the contract catalogue is that products may have constraints or rules associated with them that govern their ordering and provisioning. For example a network product such as MPLS² can only be provisioned in certain configurations and is dependant on other factors such as current usage and location. The ‘rules’ associated with each product need to be represented and then evaluated before a product can be ordered. Representing each product in ontology offers the opportunity to model some of these rules. SWSs offer the ability to model the capabilities of services associated with a product. This would enable a product manager to automatically discover SWS that can query information about a product, provide configuration information, and provide fulfilment (e.g. ordering and billing). With multiple suppliers offering similar products, the discovery mechanism can be used to find the supplier that best matches the requirements (e.g. a particular supplier can deliver the next day), that are expressed in a WSMO³ Goal.

Product Bundles, which are a collection of ICT products, are something which product managers may wish to create to allow a simple way to order complimentary products (Such as a Desktop PC & Software). Part of the Rules of each product may determine dependencies or constraints on other

² <http://www.mplsarc.com/mplsfaq.shtml>

³ <http://www.wsmo.org>

products (e.g. Software requires minimum RAM of 512mb) that must be evaluated before a product bundle can be created. SWS and ontologies offer the ability to model these rules and allow automatic evaluation of product dependencies.

The development of the contract catalogue prototype is incremental, with the final version delivering all of the required features identified in the design phase. The prototype is aimed at Product Managers who deal with the Contract Catalogues in BT. Generally they do not have previous experience with Semantic Web technologies, so it is important the prototype is useable without any previous knowledge of ontologies or Semantic Web Services. A lot of the features aim to provide an ‘easy to use’ layer on top of the current tool available in DIP.

2 BACKGROUND AND MOTIVATION

BT’s business model is focusing on transforming its revenue streams from the declining ‘Traditional’ telecommunications products such as Fixed Line Voice Services to the ‘New Wave’ products such as Broadband and Networked IT Services.

BT has developed a strategy for moving into this new telecommunications market into the 21st century. BT is moving to a more ‘Networked IT’ outlook rather than straight telecommunications provider it means that it is competing in a lot more markets than it was previously, with a wide range of potential competitors companies. Also to enable BT to reach markets where it does not supply all of the products in that market (e.g. corporate ICT solutions) it has made a number of strategic alliances with companies. This is a formal agreement to work in partnership with that company on particular bids. An important strategic alliance is with Hewlett Packard, who provide most of the Hardware for BT’s corporate ICT customers.

As described in the previous section there are a number of issues associated with allowing a supplier to fully integrate into the contract catalogue. Each Supplier internally manages their own set of products and rules, but must interface to the BT contract catalogue to provide the basic functions of publish catalogue, query configurations and accept orders.

There are several problems with integrating suppliers:

Incompatible interfaces or OSS systems

At present there is no standard communication interface to the BT contract catalogue, the suppliers will have different terminology, data models and processes for their internal systems.

Inconsistencies or differences in modelling and evaluating rules

Part of the product querying and ordering process requires rules associated with a product to be evaluated to ensure all criteria for ordering have been met. These rules may be able to be modelled statically, i.e. they won’t change over time, or they may require dynamic interfacing to a live system for evaluation (e.g. check if there is available bandwidth on a particular network). The task of allowing each supplier to manage their own set of rules, but allowing them to be evaluated when required by the customer or product manager is challenging.

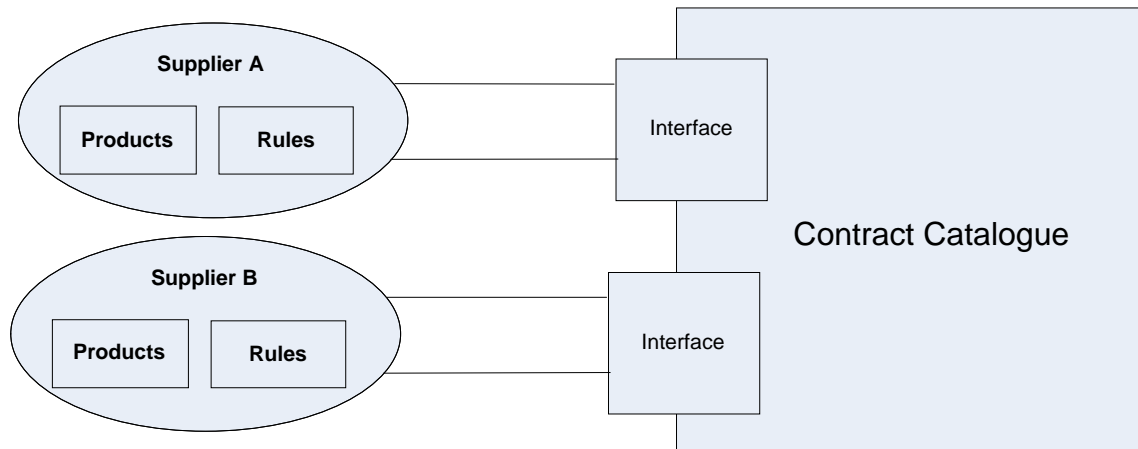


Figure 3. Multiple Suppliers Interfacing Contract Catalogue

No link between Product Catalogue and associated OSS functionality

Currently when products are ordered from the catalogue there is no link to the system needed to fulfil and manage the order. There are many systems operating for different products and services, meaning the process and workflow to order a set of products are complicated and manually intensive.

2.1 Aims of Prototype

The main aims of the prototype are to demonstrate:

- Representing a Contract Catalogue as an ontology provides advantages over standard document or spreadsheet based approaches
- Integration of third party suppliers products (and associated rules) into the catalogue can be done more efficiently with use of ontologies and Semantic Mediation.
- Using Semantic Web Services for functionality associated with products (such as ordering & querying) provides increased automation and efficiency over standard Web Services and other methods for B2B integration.

2.2 Further Reading

For more details on the background and challenges that BT faces and motivation for using Semantic Web Service technology it is recommended to read some previous deliverables in WP8 as well as the

exploitation plan in WP13. These are available for download at <http://dip.semanticweb.org/deliverables.html>

D8.2 - Platform Requirement Specification

D8.3 - B2B in Telecommunications - Prototype Platform Design

D13.4 -Revision of exploitation roadmap

There have also been a number of papers published in the British Telecom Technology Journal (BTTJ) that give a more detailed analysis of the background to BTs problems as well as the motivation and benefits for using SWS. Electronic copies can be found at <http://www.springerlink.com/content/1573-1995/>

Product catalogue management with Semantic Web Services, *M. Richardson and T. Midwinter*, BTTJ October 2006.

Service assurance across organisational boundaries with semantic mediation, *A Duke, M Richardson, S Watkins, A Wahler and B Schreder*, BTTJ January 2006.

Enabling a scalable service-oriented architecture with semantic Web Services, *A. Duke, J. Davies and M. Richardson*, BTTJ July 2005.

3 FEATURES OF PROTOTYPE V2

Ontology based design environment for creating product bundles

The model of Contract Catalogue, associated relationships, properties and rules are defined as a WSML Ontology. However the prototype provides a more simple hierarchical view of the Contract Catalogue to the product manager, so they can easily navigate it and create product bundles. The prototype allows users to ‘Drag and Drop’ products into the window from the ontology, which it then attempts to add to the bundle.

Check the overall consistency of a product bundle against the ontology & evaluate the bundle against product rules (axioms) using the reasoning component.

Once a product has been added to a product bundle, the reasoning component is then automatically invoked to check that adding this product does not violate the ontologies consistency or violate any axioms. For example there may be a rule that if online service (e.g. share price feed) is added to the bundle then a compatible network connection must be included. An example rule for this is shown below

```
axiom SharePriceFeed_requires_network_bandwidth
  definedBy
    !- ?b memberOf Bundle
    and ?b[hasOnlineService hasValue ?o]
    and ?o memberOf SharePriceFeed
    and ?b[hasNetworkconnection hasValue ?n]
    and ?n[providesBandwidth hasValue ?x]
    and ?x < 512
```

This rule states that if the product SharePriceFeed is added to a bundle then the bundle must also contain a product classified as a networkConnection. Also this networkConnection product must provide a bandwidth of at least 512 Kb/sec. If the bundle does not contain a network connection, or the network connection does not meet the minimum bandwidth requirements, the reasoner will detect this and inform the user.

Perform custom and pre-defined queries over the product catalogue ontology

As well as consistency checking of bundles and axiom evaluation, the reasoner also has the ability to perform general queries on the ontology. These queries can infer new information from the ontology given the set of concepts, instances and axioms that exist. These queries are expressed in the Web Services Modelling Language (WSML), so it is not expected that a product manager will understand and

enter these queries themselves. The prototype has the ability to create template queries and then save these with an associated easy to understand English descriptions. The product manager can simply select the English description of the query and the corresponding WSML query is invoked. For advanced users there is the ability to enter queries manually if desired. For example, Once a product bundle has been defined in the ontology it is possible to perform queries to ask specific questions about the bundle. A simple question might be ‘does my bundle contain a network connection product?’ The WSML query for this would be ‘MyBundle Contains NetworkConnection?’. The reasoner will then return a yes or no answer. Queries could also be used to classify the bundle into specific categories according to the products they contain. The prototype ontology defines a number of simple categories such as: Mobile bundle, Broadband Bundle and Graphics Bundles. Each category contains axioms that define rules for membership (e.g. to be a Mobile bundle it must contain a laptop PC). A query can be performed to ask ‘What categories does my bundle fit into?’ The WSML query for this would be ‘MyBundle memberOf ?b and ?b subConceptOf Bundle’. This is a useful as it allows the product manager to create bundles as they desire and then use the reasoner to classify them. This is an example of how a reasoner allows you to infer new information (i.e. what category a bundle is) from an ontology.

Semi-Automatic Discovery of associated Product functionality via SWS

Being able to create the product bundles easily in an ontology based environment is important for product managers, but just as important is linking the creation of a product bundle to the tasks that need to be carried out to deliver this bundle to the customer. The prototype can automatically generate WSML goals (and then discover and invoke the required Semantic Web Services) to achieve certain tasks associated with the product bundles. The most common (and first implemented here), is the Goal to order the product bundle.

A goal is a semantic description of a task that the user wishes to achieve. A goal is defined in WSML and specifies the desired action of a Web Services. The detail in which you specify the goal can vary. It can be a very high level description of the desired action of the Web Services (e.g. “I require a Web Service to order computers”) or can be more specific (e.g. “I require a Web Service to order desktop computers in the UK, which accepts credit cards and provides next day delivery”). Specific inputs and outputs that are expected from a Web Service can be specified (e.g. the computer ordering Web Service must provide an order confirmation number as an output) Goals use WSML ontologies to describe the concepts relating to the desired action of the Web Service. A discovery engine can then use the goal to automatically find Web Services that provide the functionality requested.

An example goal to order a Desktop PC is given below:

```
goal _"http://www.bt.com/examples/order_desktop_computer"
  importsOntology                                     {cat
  _"http://www.wsmo.org/ontologies/catalogue.wsml",
    {loc _"http://www.wsmo.org/ontologies/location.wsml",
    {tr _"http://www.wsmo.org/ontologies/transaction.wsml
  capability
```

```
postcondition
  definedBy
    ?Computer_order memberOf cat#computer_order_service[
      ships_from hasValue loc#UK,
      ships_to hasValue loc#UK,
      payment hasValue tr#creditcard
    ]
```

The top four lines define the name of the goal and specify which ontologies are imported to describe the goal. Ontologies are usually split up into distinct domains of interest. In this case the goal needs to describe concepts relating to catalogues, locations and transactions. The rest of the goal defines the desired action of the Web Service in terms of these ontologies. Firstly it requires a service that is classified as a `computer_order_service`. Secondly it requires that the service ships in the UK and accepts credit cards.

As well as discovering Web Services and important requirement in this scenario is the ability to join these services together to create a composed service, so that all the products in a bundle can be ordered together. This can be achieved by use of the Goal Orientated Composition prototype developed in DIP. The prototype allows you to create a workflow to describe the relationships between the different individual services in the composition. This is described in more detail in section 5.

Import (mediate) between third party product ontologies

One of the problems with managing a product catalogue containing products from many suppliers is that there will be inconsistencies in the way products and data are represented by each supplier. Individual companies are likely to have their preferred method of representing their products which may not be compatible with how BT wishes to present the combined catalogue to the customer. Ontologies incorporate the idea of mediation where it is possible to define a mapping between concepts in different ontologies with equivalent semantic meaning.

Mediation is generally achieved through the use of mediators, i.e. components which enable heterogeneous systems to interact. In a practical sense, mediators have generally been realised as pieces of program code that perform point-to-point, low-level translations. Although such mediators satisfy the short-term goal in that they allow two systems to talk to each other, they suffer from maintainability and scalability problems. In general, it is not likely to be feasible to automate their application in a dynamic environment because of their close coupling with the implementation.

Semantic mediation enables a more dynamic approach through the use of ontologies, which provide formal conceptualisation of a given domain. Mediators can be used to convert from a source implementation interface to that of a target implementation. Modelling the processes and data in the source and target interfaces using ontologies, enables the definition of relationships between semantically equivalent concepts. The mediator can use these relationships to dynamically map between the source and target. Instead of defining individual mappings for each message that is passed between systems, semantic mediation allows the mappings to be defined once in the ontologies and then individual messages are dynamically mapped as they are exchanged. This reduces the overhead in creating the individual mappings manually and also increases flexibility. With a manual approach the system is inflexible to changes in the message format or content. Within the DIP project a tool has been

developed called the Ontology Management Suite (OMS) which allows mapping to be defined between ontologies.

4 INSTALLATION GUIDELINES

Pre-requisites

- A PC with minimum 512mb Ram and 1 GHz processor
- Java version 5 (can be downloaded from <http://java.sun.com>)
- WSMO Studio Version 0.5.1 (see point 1)
- Ontology management suite (OMS) plug-in (see point 2)
- KAON2 reasoning plug-in (see point 3)
- Goal Orientated Composition prototype plug-in (see point 5)
- UML2 compatible tool such as Visual Paradigm (see point 6)

The Contract Catalogue prototype has been developed as WSMO Studio plug-in so must be downloaded in order for the plug-in to work. It has been optimised for use in WSMO Studio Version 0.5.1. It should work on later versions, although this cannot be guaranteed.

In addition a number of plugins are required and need to be downloaded separately, as they are not included with the standard WSMO studio release.

1. Download WSMO studio Version 0.5.1 from <http://wsmostudio.org/download.html>
2. Download the Ontology Management Suite from <http://www.omwg.org/tools/dip/factsheets/OntologyEditorFactSheet-20061018.html>
3. Download the Goal Orientated Composition prototype from <http://www.massidia.fr/semanticweb/gobac/>
4. Download the Contract Catalogue prototype <http://ngwr.labs.bt.com/DIP/downloads/DIP8.5v2.zip>
3. Download the KAON2 reasoning module from <http://kaon2.semanticweb.org/#download>
6. Any UML2 tool supporting activity diagrams can be used. A free evaluation version of Visual Paradigm can be downloaded at <http://www.visual-paradigm.com/>

The steps for installing the prototype are given below

4. For items 1-3 follow the individual instructions in the download packages to install
5. To install the contract catalogue prototype. Unzip the downloaded file and copy the folder 'com.bt.exact_tech' to the plug-in directory your WSMO studio folder. By default this should be <unzip location>\ WSMO-Studio-0.5.1\plugins
6. Unzip the KAON2 file and copy 'kaon.jar' to <unzip location>\ WSMO-Studio-0.5.1\plugins\com.bt.exact_tech\lib.
7. Launch 'WSMOSudio.exe' from the WSMO-Studio-0.5.1\ directory. Note: Java version 5 is required to run WSMO Studio.

8. WSMO studio should now load. The prototype includes a WSMO project with example ontologies and Web Service descriptions. The project is contained in the directory 'Catalogue'.

10. To see how to use the prototype, follow the walkthrough section below.

5 EXAMPLE WALKTHROUGH

This section gives an example of how the Contract Catalogue prototype is used to create product bundles. The installation includes an example product catalogue modelled in WSMO and a number of WSMO Web Services descriptions associated with the products in the catalogue. The following walkthrough shows how to:

- Create a new empty bundle
- Browse the product catalogues to add products.
- Use the reasoning component to evaluate the bundle against product rules (axioms)
- Check the overall consistency of you product bundle against the ontology
- Automatic creation of WSMO goal to order product bundle
- Design of goal composition workflow
- Generation of composition with the goal orientated composition module (using discovery)
- View the 3 layer orchestration created

1. Select File→Import from the top menu. The select ‘import existing project into workspace’. Navigate to the folder:

`\WSMO-Studio 0.5.1\plugins\com.bt.exact_tech\examples\catalogue`

Then click ‘OK’ to import the project.

2. Double click the file ‘Catalogue_BT’ to open the BT catalogue ontology. The WSMO Studio navigator window in the bottom right should now display a hierarchical view of the Contract Catalogue ontology. This contains a high level view of all of the entities involved in the management of a contract catalogue (e.g. contract managers, contracts, products, service agreements). You can navigate the ontology by expanding the tree in the WSMO Studio Navigator Window.

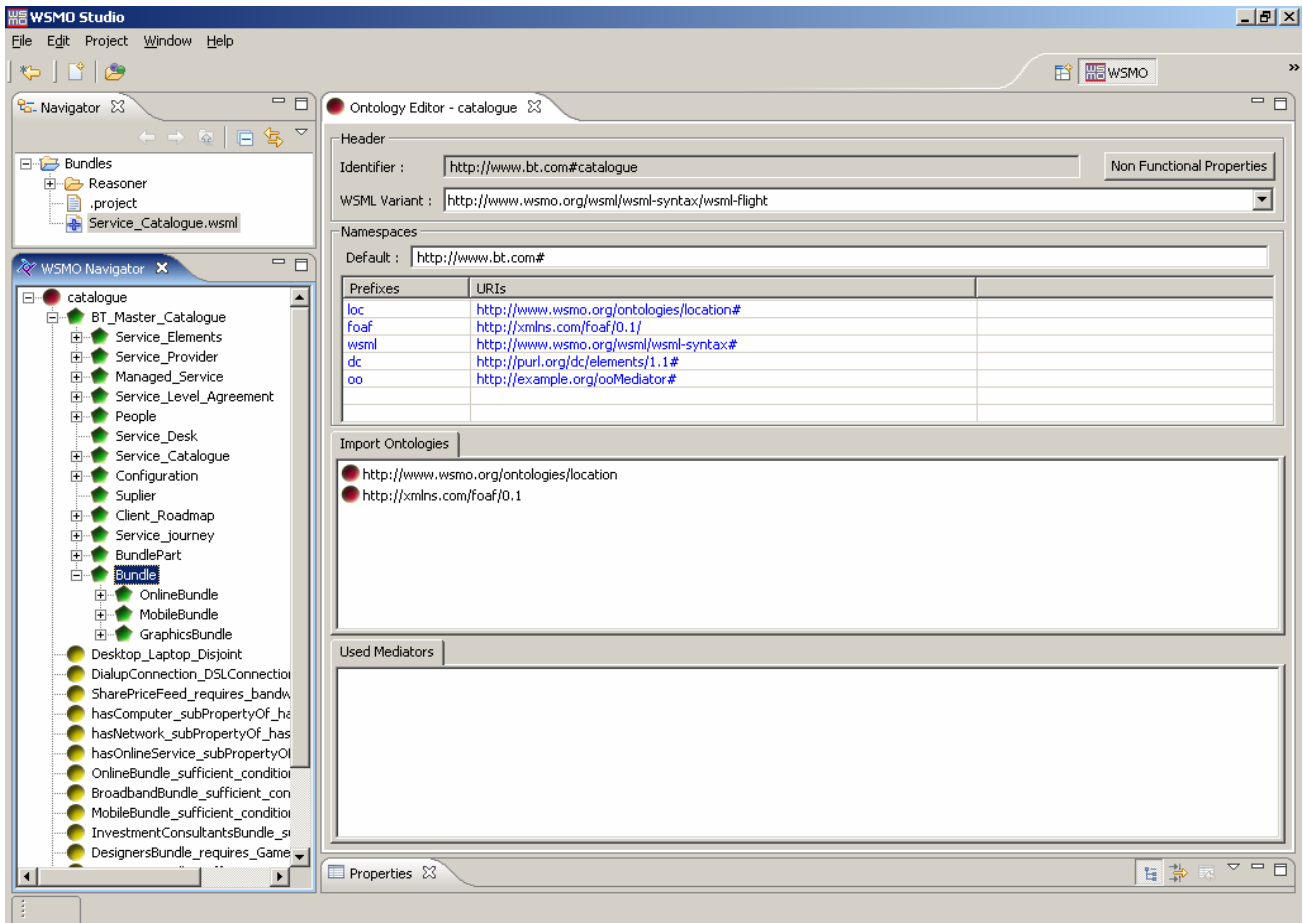


Figure 1. Loading the Contact Catalogue

3. Click on the concept Bundle and expand it to see all its sub concepts.

The concept Bundle in the ontology is used to describe a collection of products that are combined together to form a new bundled product. The ontology defines a number of axioms (the yellow bullets in the Navigator Window) that detail rules and constraints inside bundles and between individual products.

We will now create a new bundle and use the Prototype plug-in to allow us to drag and drop products into to bundle and automatically evaluate the axioms.

4. Right Click on the Concept Bundle and select ‘Create Instance’ from the context menu. Select a name for you new bundle (suggested MyBundle) and press enter. The name of your bundle instance should appear at the bottom of the Bundle sub-tree.

5. Next right click the MyBundle instance and select ‘Edit with → Manager Editor’. This will initialise the prototype plug-in. You will be presented with the screen shown in Figure 2.

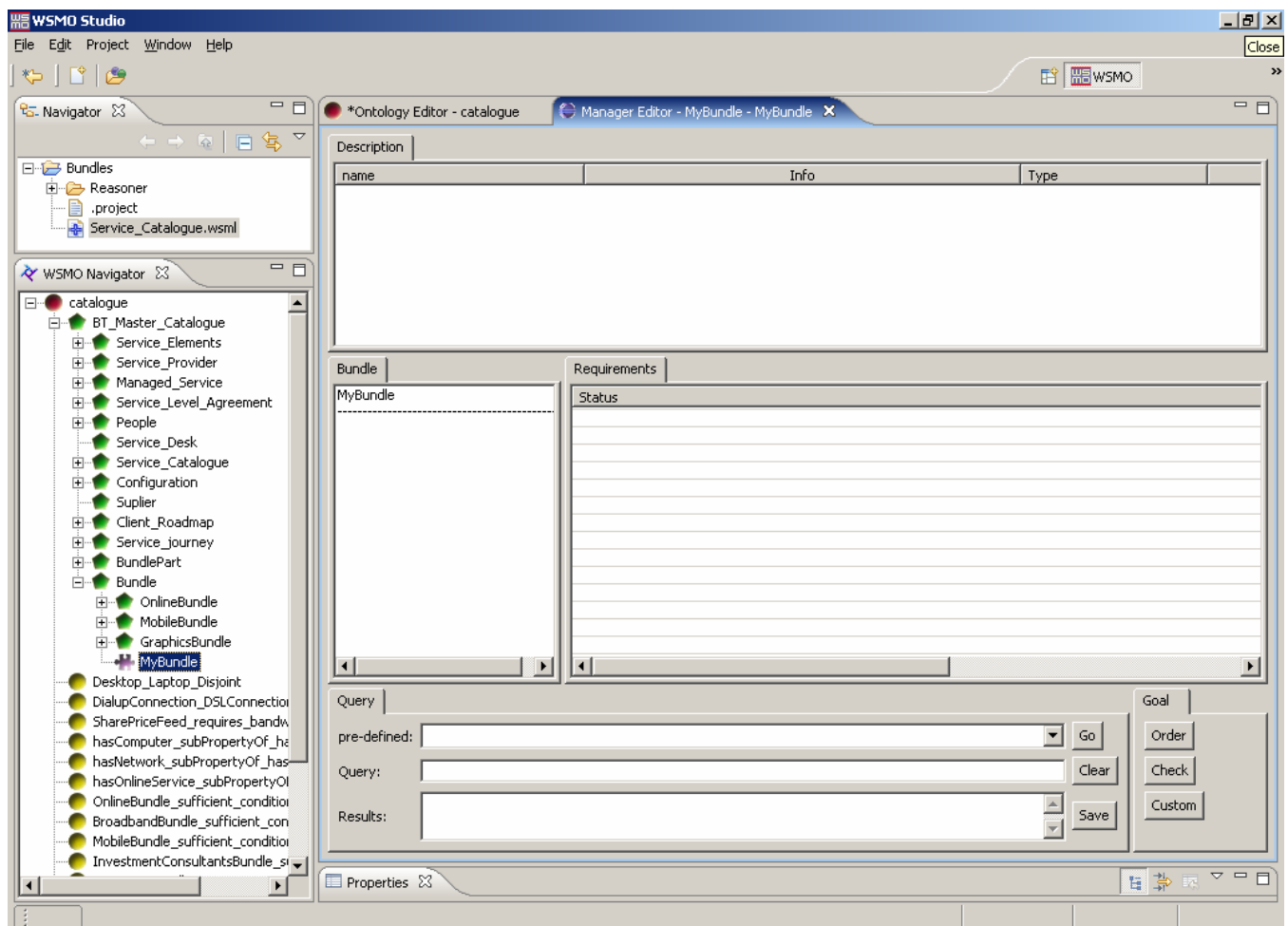


Figure 2. Creating a new Bundle

6. The prototype User interface is split up into 5 panes which perform various functions:

Description (top pane) – This is used to show the full ontological description of the bundle you are currently creating. It contains all of the attributes of the bundle, which are used to define the relationship between the bundle and its contents. This view is useful for people with some ontology experience as it gives a more detailed view of the bundle

Bundle (mid left pane) – This is used to give a simple view of the bundle. It lists the bundle name and all the products that are contained in it. For product managers with less ontology experience this would be their preferred view of a bundle. This bundle window is also interactive, it lets product managers ‘Drag and Drop’ products into the window from the ontology, which then attempts to add them to the bundle. The Reasoning component is then automatically invoked to check that adding this product does not violate the ontologies consistency or violate any axioms.

Requirements (mid right pane) – This gives information to the product managers if the reasoner has found any problems following a product being added to the bundle (resulting from consistency or axiom violations). The plug-in converts the general consistency check messages given by the reasoner into more user friendly ‘bundle specific’ messages that can be understood by a product manager

Toolbar (bottom left pane) – As well as consistency checking of bundles and axiom evaluation, the reasoner also has the ability to perform general queries on the ontology. These queries can infer new information from the ontology given the set of concepts, instances and axioms that exist. These queries are expressed in the Web Services Modelling Language (WSML),⁴ so it is not expected that a product manager will understand and enter these queries themselves. The plug-in has the ability to create template queries and then save these with an associated easy to understand English descriptions. The product manager can simply select the English description of the query and the corresponding WSML query is invoked. For advanced users there is the ability to enter queries manually if desired.

Goal (bottom right) – Being able to create the products bundles easily in an ontology based environment is important for product managers, but just as important is linking the creation of a product bundle to the tasks that need to be carried out to deliver this bundle to the customer. Using this Goal bar allows the manager to automatically generate the WSML goals (and then discover and invoke the required Semantic Web Services) to achieve certain tasks associated with the product bundles. The most common (and first implemented here), is the Goal to order the product bundle. As the product catalogue and associated entities are ontologies described in WSML, it makes the task of creating goals to order the products easier. The DIP runtime architecture (in this case WSMX) can then be used to discover Semantic Web Services that supply the desired products and then invoke the Web Services to order them.

7. The Next stage is to add some products to the bundle. This is done by selecting instances of products from the Navigator window and dragging them into the bundle window. For simplicity in this example

⁴ <http://www.wsmo.org/wsml/>

there is a concept called `BundlePart` which has a small subset of all the products in the ontology, along with the some axioms defining their rules.

Expand Concept `BundlePart` and drill down to `Computer`→`Desktop`, you will see one instance of Desktop computer `HPxyz`. Add this to the bundle by dragging it into the bundle box. You should see its name appear in the bundle box and description pane at the top. The prototype now calls the reasoner to perform a consistency check. If there are any problems with adding `HPxyz` to the bundle they will be displayed. You should not see any messages come up indicating that product is ok to add to the bundle (Figure 3).

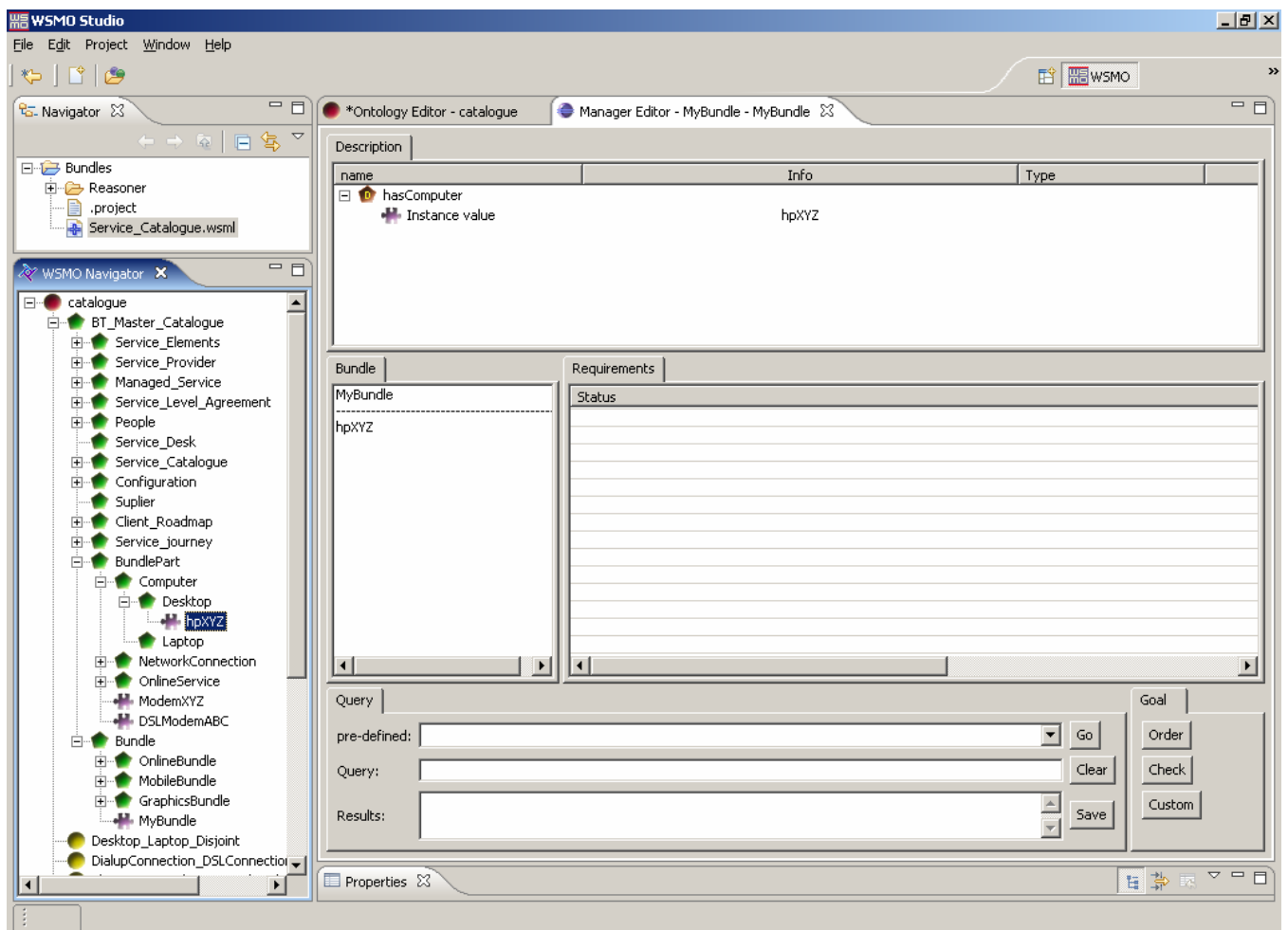


Figure 3. Adding a product to the Bundle

8. Next Drill Down to `NetworkConnection`→`DSLConnection` from `BundlePart` and drag the `AccorDSL` instance into the Bundle Box. This should be added in a similar manner to 7.

9. Next Drill down to `NetworkConnection`→`DialUpConnection` from `BundlePart` and drag the `BTdialup` instance into the bundle box. As before a consistency check is performed by the reasoner, but this time it has found a problem. The bundle now contains two `NetworkConnection` products and the ontology states that only one is allowed per bundle. You should see the message in the Requirement

window. To fix the problem select `BT_Dialup` from the bundle window, right click and select ‘remove’ from the context menu. This product is now removed from the bundle and is consistent again.

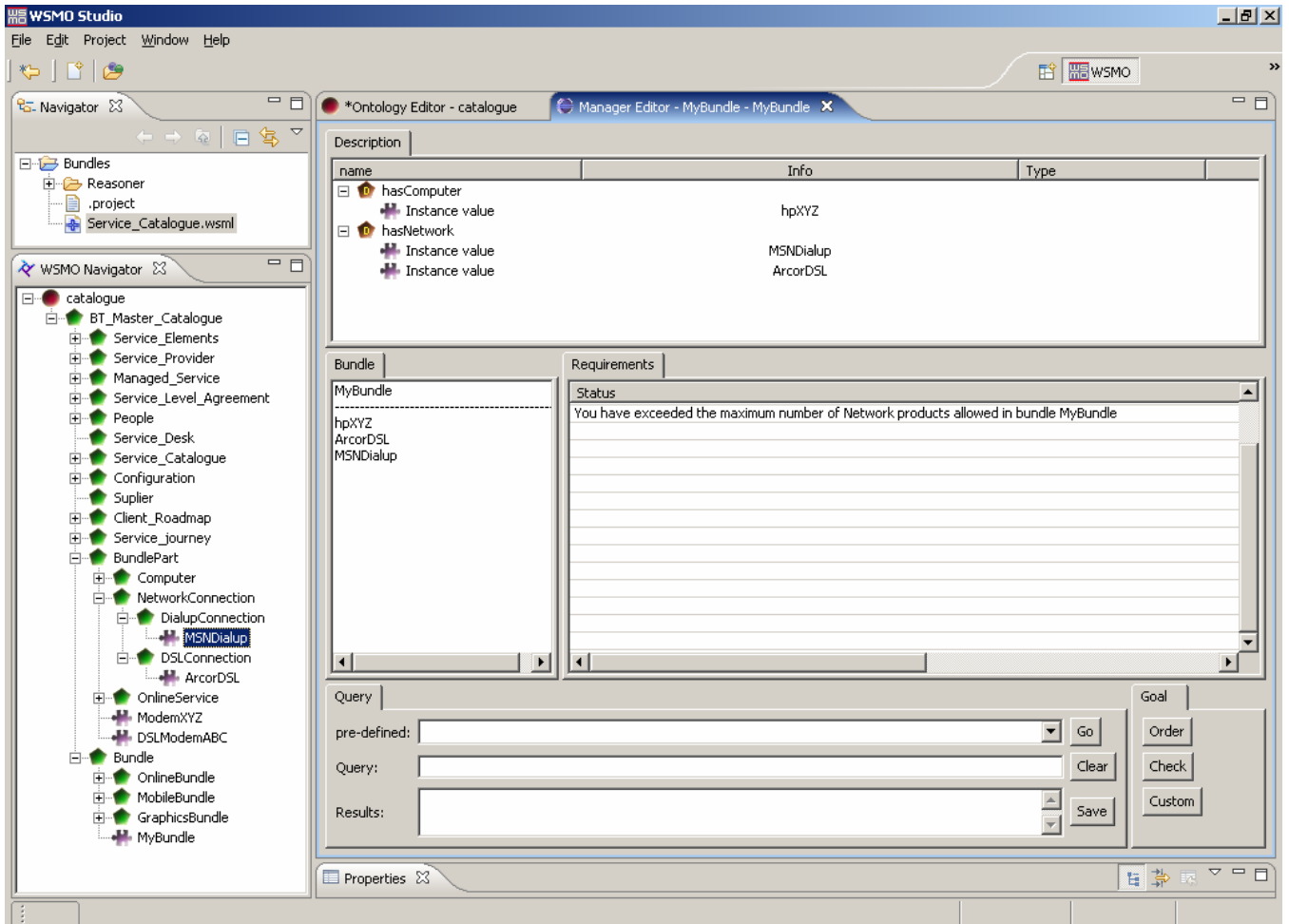


Figure 4. Reasoner finds an inconsistency in the Bundle

10. To complete the bundle Drill down to `OnlineService`→`SharePriceFeed` from `BundlePart` and drag the instance `uBiqBankShareInfo` into the bundle box.

11. Now the bundle is complete it would be useful to query the reasoner to infer some extra knowledge about the new bundle. The ontology defines a number of specialisations of the general `Bundle` concept which have extra axioms to define what is required for membership. For example to be part of a `MobileBundle` the bundle must contain a `Laptop`. We can use the toolbar at the bottom of the interface to launch a number of queries.

From the ‘pre-defined’ drop down box select the first query ‘Is MyBundle an online Bundle’. The results from the query should be ‘yes’. The rule for membership to `OnlineBundle` is that the bundle must contain a `NetworkConnection` product, which `MyBundle` does (`AccorDSL`).

You can also do more complicated queries such as ‘What bundles can MyBundle be’. This will tell you all the specialised bundles that `MyBundle` can be a member of (Figure 5).

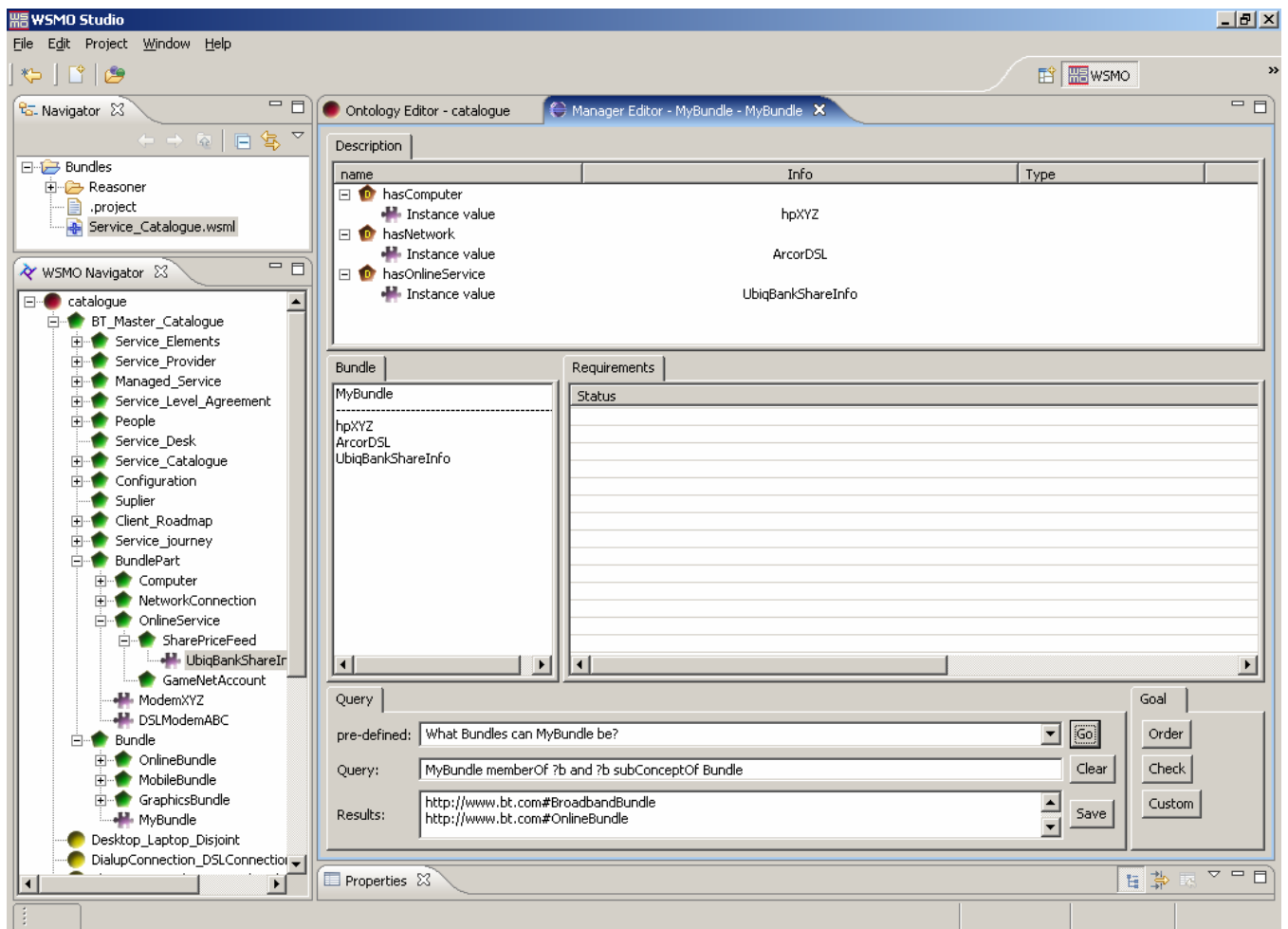


Figure 5. Performing queries on the Bundle

12. The next stage in the prototype v1 is to automatically generate a WSMO goal to order the product bundle. This can be used to discover and invoke the Semantic Web Services that supply the products in the bundle via WSMX.

To carry out this step, click the ‘Order’ button from the Goal pane in the bottom left of the interface. This will now automatically create a Goal and a UML2 (XMI file) Skeleton Composition file (for creating the composition). At this stage you can edit the Goal further or save it by selecting File→Save from the top menu (Figure 6).

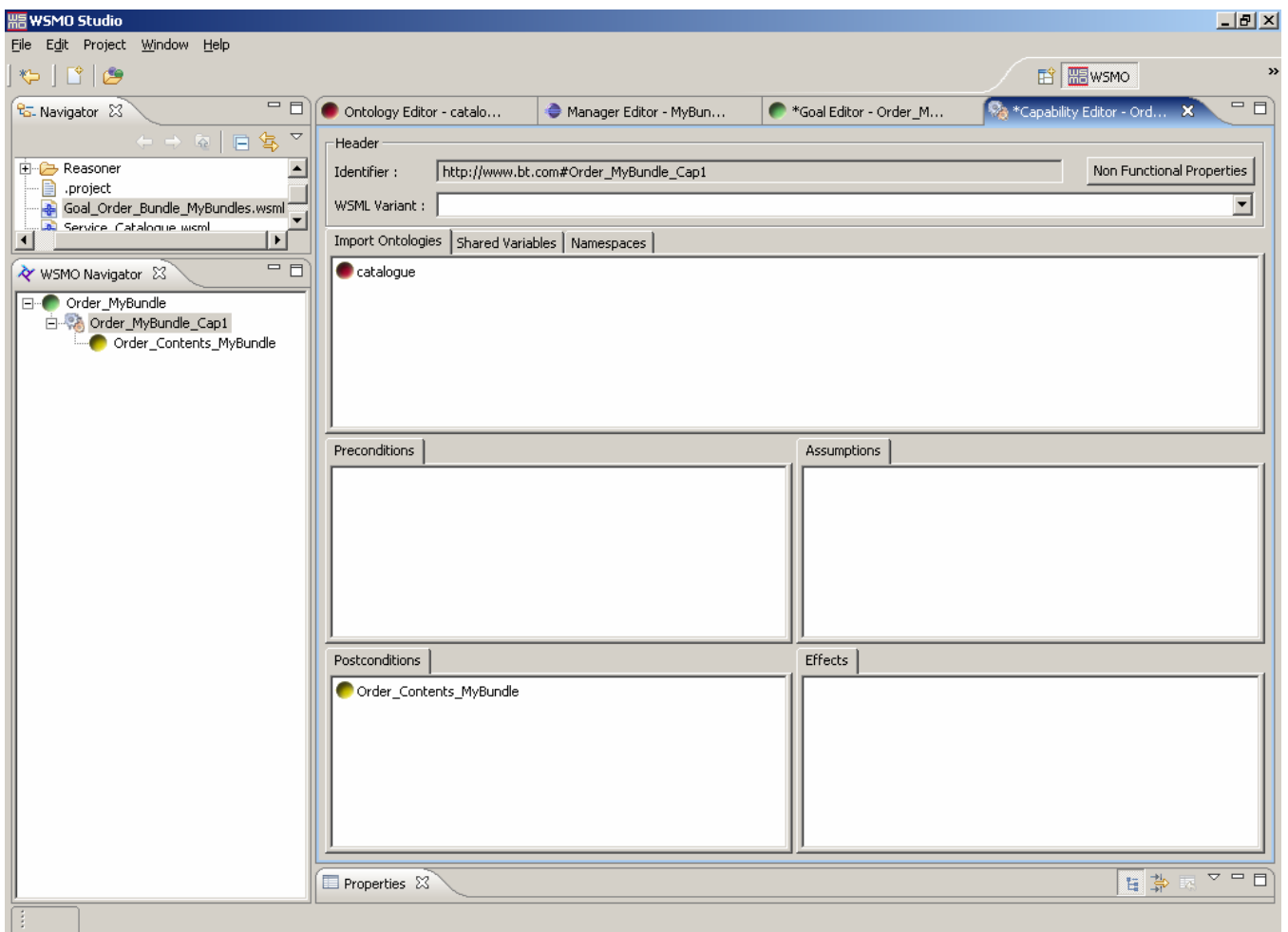


Figure 6. Goal generated for product Bundle

13. The prototype can automatically generate the skeleton composition file to order the products in a bundle, with an aim to creating a composite Web Service to order all the products. At this stage the dependencies between the different ordering Web Services are not determined automatically. For example if there is a service to order a Network Connection, this needs to be invoked before the service

to order the Network Modem (if the Network Connection order fails then you don't need to order the Modem). This workflow of dependencies between the different Web Services in the composition needs to be expressed using a UML2 tool (specifically UML2 activity diagrams), which is the required input for the Goal Orientated Composition module to generate the orchestration for the composite Web Service.

Using a UML2 compliant tool, open up the skeleton composition file. You should see something similar to Figure 7.

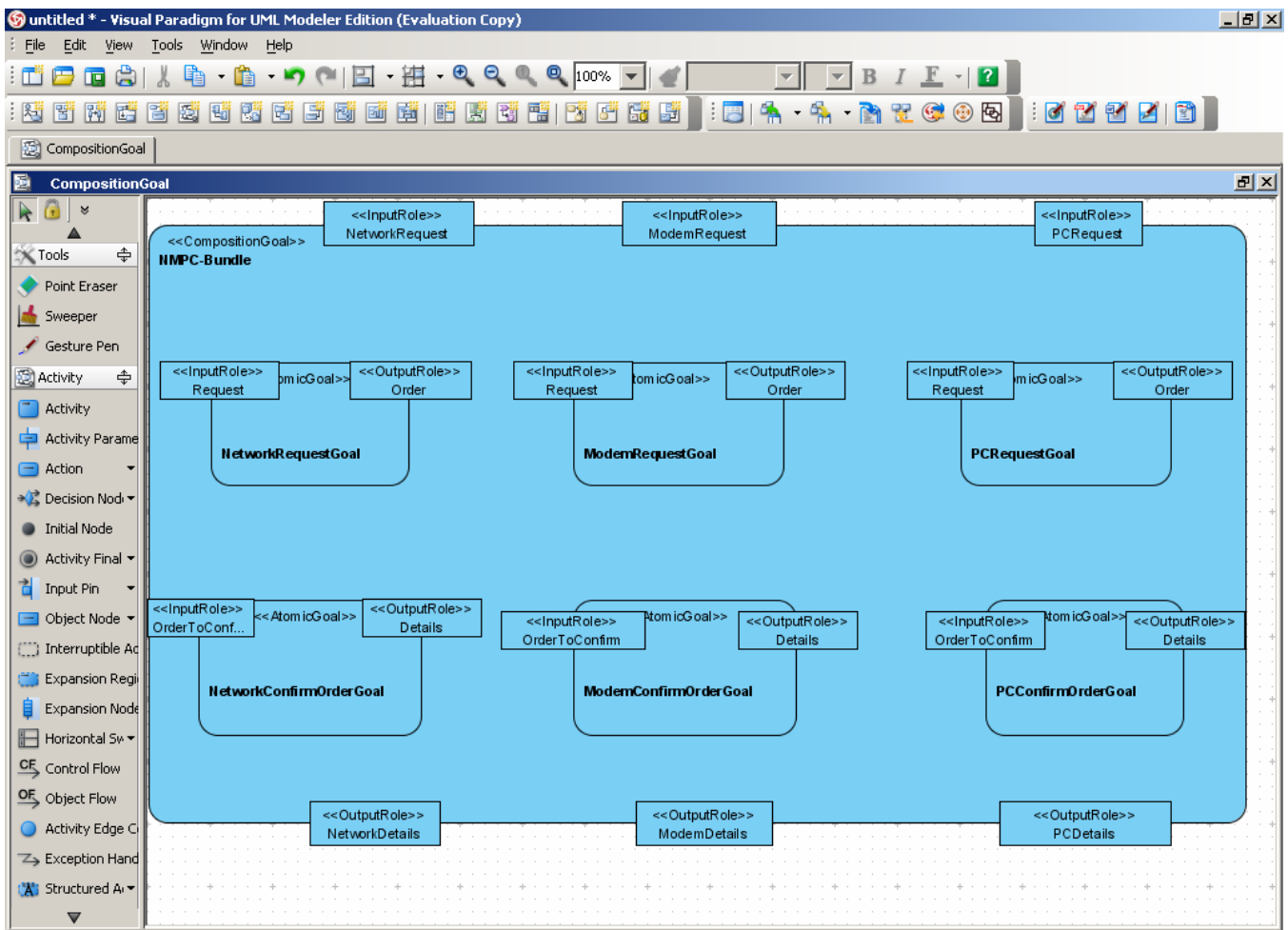


Figure 7. Loading the skeleton composition goal

14. The diagram shows a graphical representation of each of the individual or ‘atomic’ goals in the composition, i.e. the goals to query and order the individual products in a bundle. Now you can describe the relationship between the different goals using the UML2 activity diagram modelling primitives. For more information on how to do this refer to DIP deliverable **D4.22** Goal-orientated SWS composition prototype. The example below describes a workflow where the ordering Web Services of three products in the bundles needs to be invoked in a specific order, with the whole order being cancelled if all three products cannot be ordered. Once the workflow has been described, save the file in XMI format.

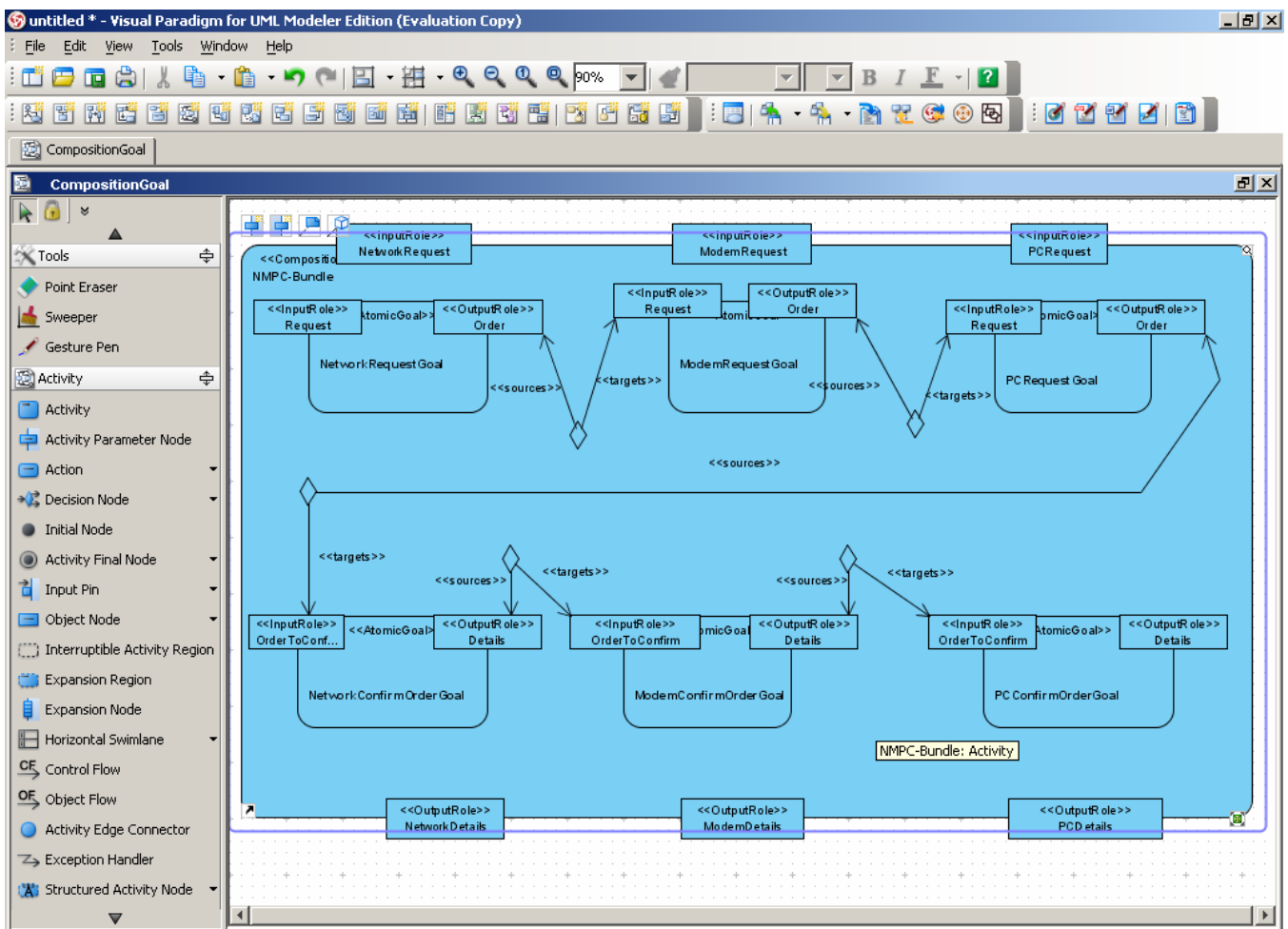


Figure 8. Designing the Composition Goal

15. Once the workflow of the goal composition has been determined it is possible to pass this to the Goal Orientated Composition Module, to generate a concrete orchestration for the composite Web Service. This stage involves a number of steps:

- Reading in the composition workflow in the UML2 file
- Using the discovery module to find concrete ‘Atomic Web Services’, that provide the ordering functionality for the individual products in the bundle
- Creating the Orchestration (i.e. instructions for how to invoke the atomic Web Services to achieve the desired workflow described in the composition)
- Passing the Orchestration to WSMX or IRSIII to invoke the composition.

For more detailed information of how this works refer to the DIP deliverable **D4.22** Goal-orientated SWS composition prototype

16. In WSMO Studio select ‘Show View’ then select the ‘Composer’ option.

The first tab in the Composer screen asks you to select a repository. This is where the Semantic Web Service descriptions are held. There is a repository setup with example Web Services for ordering the bundle items described in this example walkthrough (for others you need to setup your own repository). Select ‘BT_repository’ from the drop down list, as shown in the diagram:

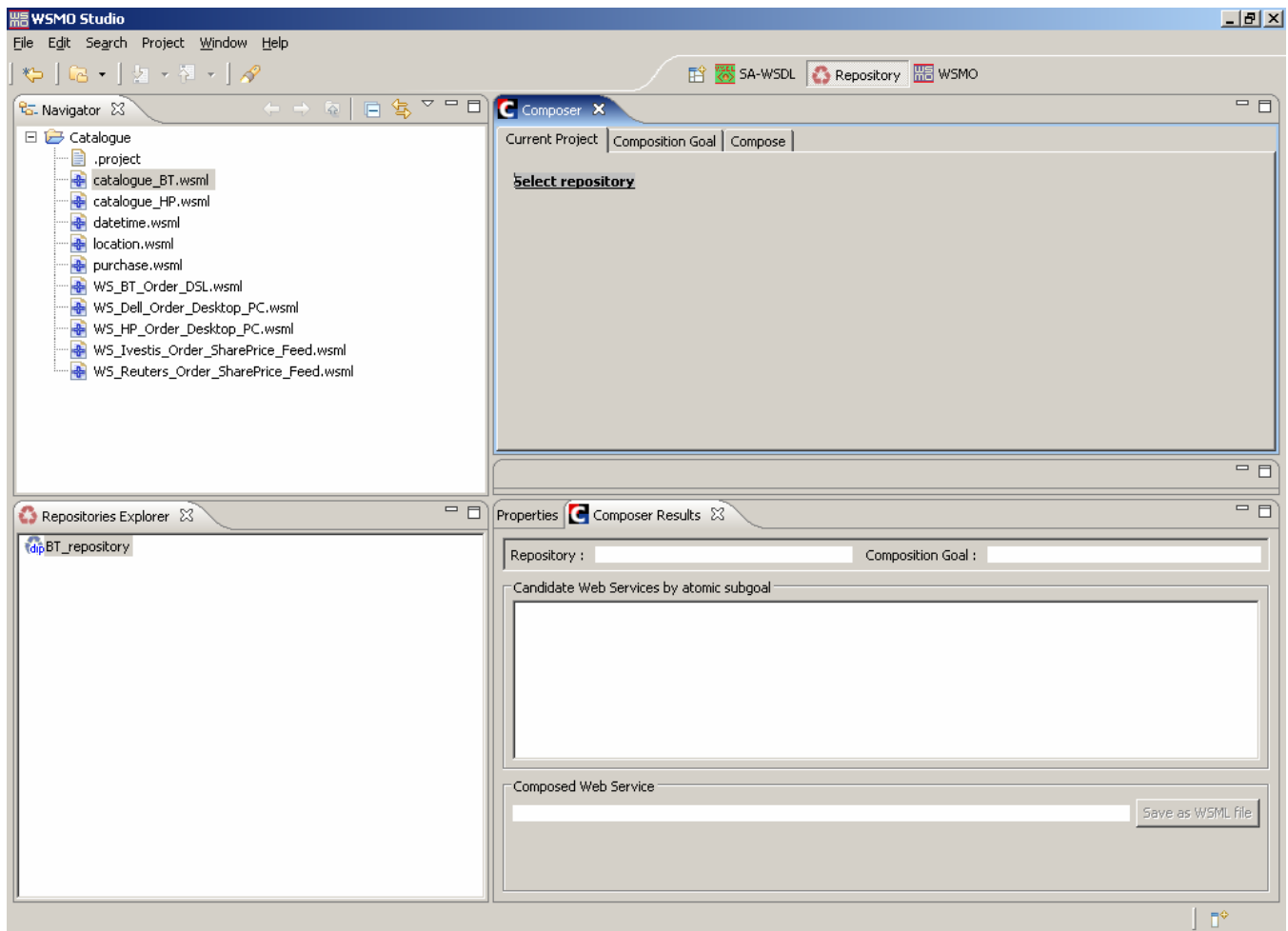


Figure 9. Running the Goal Orientated Composition Plug-in

15. Next task is to load the composition goal created (in step 14.). Select the 'Composition Goal' tab in the composer windows, and then select 'Import a composition goal'. Navigate to the XMI file saved in step 14. and select 'OK' to import. The composer will now load the composition goal. To begin the process of discovering the Web Services and building the concrete orchestration, select the 'Compose' tab and select 'Compose Orchestration'.

16. The composer will now begin the task of discovering suitable Web Services in the repository and creating the Orchestration. You will see various messages appear on the screen as the composer carries out the different tasks. If a successful composition can be created the composer will output an orchestration file. You can save this by selecting the 'save as WSMML file' option.

17. The Orchestration now gives concrete instructions for how to invoke the different individual ordering Web Services for the product in the bundle. This file can now be passed to the WSMX orchestration engine for invoking. For more information on executing the orchestration please read DIP deliverable **D4.20** DIP Orchestration prototype

6 CONCLUSION

The deliverable outlines v2 of the Contract Catalogue prototype. It has shown that by representing a Contract Catalogue as a WSMO ontology and its product ordering interfaces as Goals & Semantic Web Services; it enables a robust and flexible system that allows product manager to design product bundles. Using the Goal Orientated Composition tool it was shown how the ordering Web Services for the individual products in a bundle can be combined to create a composed Web Service. It overcomes significant problems in the current system that is detailed in section 1. Plans for continued development and exploitation of the prototype can be found in deliverable **D13.4**.