SWWS – Semantic Web Enabled Web Services	**Title: VISP Case Study: Ontologies and Services** **Version: 1.0** **Date: 31 March 2004** **Pages: 54** **Responsible Authors:** Alistair Duke, Marc Richardson **Co-Author(s):**

Status:	**Confidentiality:**	
[ ] Draft	[ ] Public	- for public use
[ ] To be reviewed	[✓] INT	- for SWWS consortium (and Project Officer if requested)
[ ] Proposal		
[✓] Final / Released to CEC	[ ] Restricted	- for SWWS consortium and Project Officer only

Project ID: IST-2002-37134

Deliverable ID: D12.2

Workpackage No: 12

Title: VISP Case Study: Ontologies and Services

Summary / Contents:

This document is the second deliverable of the Virtual Internet Service Provider (VISP) case study. It describes the ontologies and services that are required to support the integration of system components in the VISP environment. The case study is focussing upon a particular area within service provision – problem handling. A storyboard that describes the interaction between actors and components in this area is introduced. This is considered in terms of the messages that pass between the components. A telecommunications industry wide initiative to provide information and process models is then described and its applicability as a domain ontology for the case study is considered. Finally, the component services required for the case study are described first as web services (using WSDL) and then as semantic web services using OWL-S.

SWWS Consortium

This document is part of a research project funded by the IST Programme of the Commission of the European Communities as project number IST-2002-37134. The partners in this project are: Leopold-Franzens Universität Innsbruck (IFI, Austria)); National University of Ireland, Galway (NUI, Galway, Ireland); Forschungszentrum Informatik (FZI, Germany); Intelligent Software Components S.A. (iSOCO, Spain); OntoText Lab. - Sirma AI Ltd. (SAI, Bulgaria); Hewlett Packard (HP, UK), British Telecom (BT, UK)

Leopold-Franzens Universität Innsbruck (IFI)
Institut für Informatik
Technikerstrasse 13
A-6020 Innsbruck Austria
Tel: +43 512 507 6489
Fax: +43 512 507 9872
Contact person: Juan Miguel Gomez
E-mail: juan.miguel@uibk.ac.at

National University of Ireland, Galway (NUI)
National University of Ireland,
University Road
Galway, Ireland
Tel: +353 91 750414
Fax: +353 91 562894
Contact person: Liam Caffrey
E-mail: Liam.Caffrey@nuigalway.ie

FZI – Forschungszentrum Informatik
Haid-und-Neu-Str. 10-14
76131 Karlsruhe, Germany
Tel: +49 721 9654816
Fax: +49 721 9654817
Contact person: Adreas Abecker
E-mail: abecker@fzi.de

Intelligent Software Components S.A. (iSOCO)
Francisco Delgado 11, 2nd Flor
28108 Alcobendas, Madrid, Spain
Tel: +34 913 349797
Fax: +34 913 349799
Contact person: Richard Benjamins
E-mail: rbenjamins@isoco.com

OntoText Lab.- Sirma AI Ltd. (SAI)
OntoText Lab.
38A Chr. Botev Blvd.
Sofia 1000, Bulgaria
Tel: +35 92 9810018,
Fax: +35 92 9819058
Contact person: Atanas Kiryakov
E-mail: Atanas.Kiryakov@sirma.bg

Hewlett Packard (HP)
HP European Laboratories
Filton Road, Stoke Gifford
BS34 8QZ Bristol, UK
Tel: +44 117 3128631
Fax: +44 117 3129285
Contact person: Janet Bruten
E-mail: janet.bruten@hp.com

Associated Partner:

British Telecommunications plc. (BT)
Orion 5/12, Adastral Park
Ipswich ip5 3RE, UK
Tel: +44 1473 609583
Fax: +44 1473 609832
Contact person: John Davies
E-mail: john.nj.davies@bt.com

Table of Contents

1 Introduction

This document is the second deliverable of the Virtual Internet Service Provider (VISP) case study. A VISP is an enterprise that delivers Internet services to its customers without necessarily owning or managing any of the infrastructure that is required by them. VISPs make use of telecommunications products and services provided by wholesale operators. In order for these services to be delivered, the parties involved need to communicate at a number of levels including contract agreement, ordering and fault management. There is an obvious requirement to integrate the business systems of the collaborating parties in order to reduce cost and increase customer service.

In this deliverable the ontologies and services that are required to support the integration of system components in the VISP environment are considered. The case study is focussing upon a particular area within service provision – problem handling. A storyboard that describes the interaction between actors and components in this area is introduced. This is considered in terms of the messages that pass between the components. A telecommunications industry wide intitiative to provide information and process models is then described and its applicability as a domain ontology for the case study is considered. Finally, the component services required for the case study are described first as web services (using WSDL) and then as semantic web services using OWL-S.

2 Case Study Storyboard

The case study scenario is based around providing problem resolution capabilities for a telecommunications service. The intention of the case study is to illustrate how semantic web services can improve the process of delivering these and other capabilities through reduced effort and increased reuse.

The scenario involves the following systems and actors:

- Network Alarm. An alarm is triggered by a network fault. The alarm contains details of the resource (in general a piece of hardware) that is faulty and the type of fault that has occurred e.g. loss of power.
- Inventory Management. The inventory manager holds details for resources including which services and customers make use of them.
- Trouble Ticket system. This system is used to track problems. Each problem is assigned with a trouble ticket which holds information about the problem and its current status. Trouble Tickets are closed when the problem is resolved
- Workforce Management system. This system is used to allocate jobs to members of the workforce and track the progress of those jobs.
- Customer. The customer is the end user of the affected service
- Workforce member. The person who carries out the job to address the problem
- Administrator. A person who reviews incoming trouble tickets, determines how they should be resolved and informs the customer.
- Process Manager. A system to control the process and record its state.
- Solution Designer. The person designing the problem resolution system.

2.1 Storyboard Detail

The following figures illustrate the four main stages in the scenario and the major messages that are passed between the actors / systems.

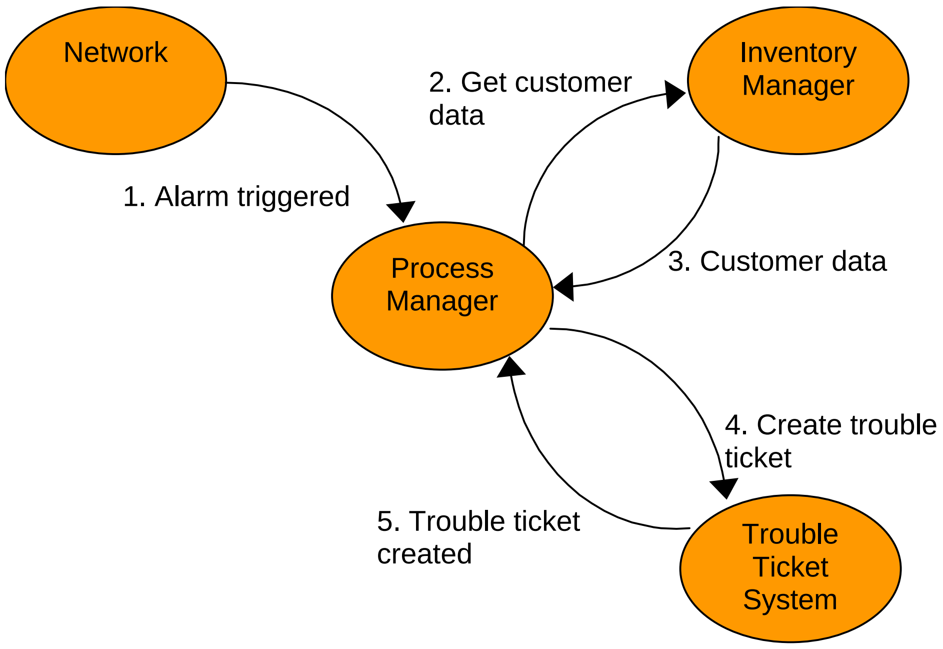


Figure 1. Alarm is Triggered

Steps 1-5 are illustrated in figure 1.

1. A network problem results in an alarm being triggered. This is captured by the process manager.
2. The process manager reads the alarm to determine the affected resource. It then carries out a request to the inventory manager to determine the customer affected by the resource[1].
3. The inventory manager responds with details of the customer.
4. The process manager requests that the Trouble Ticket system creates a new trouble ticket and provides details of the problem.
5. The Trouble Ticket system creates a new ticket (and informs the customer of the problem) then responds to the Process Manager with an ID for the created ticket.

[1] In practise more than one customer may be affected. This is constrained to just one for the purposes of the scenario.

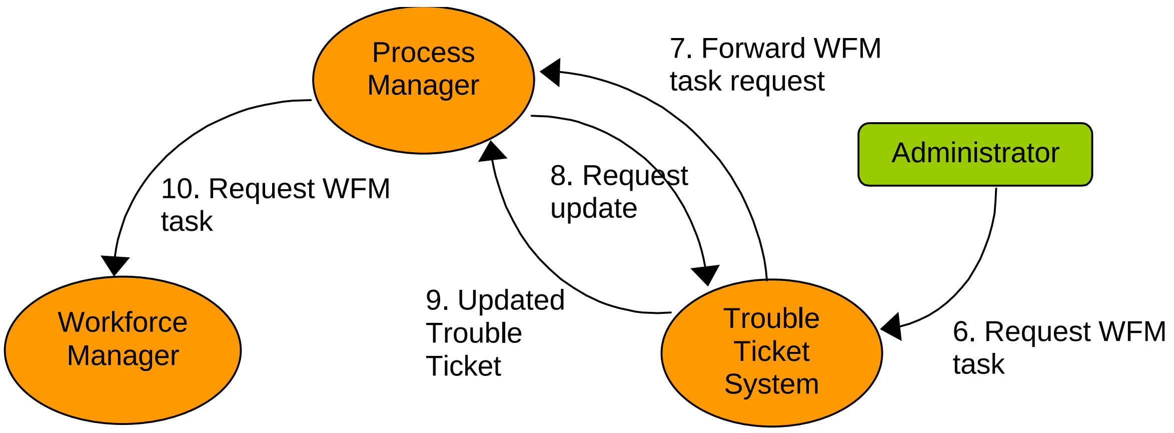


Figure 2. Workforce Task Request

Steps 6-10 are illustrated in figure 2.

6. A trouble ticket administrator is assigned with the task to review the new trouble ticket by the trouble ticket system. They decided that a job should be created on the Workforce management system to resolve the problem. They update the trouble ticket and make the request.
7. The Trouble Ticket system forwards this request to the Process Manager with an ID of the ticket
8. The process manager receives this request and requests that the full, updated ticket details are forwarded.
9. The updated details are forwarded.
10. The Process Manager forwards the job request with full details to the Workforce Manager

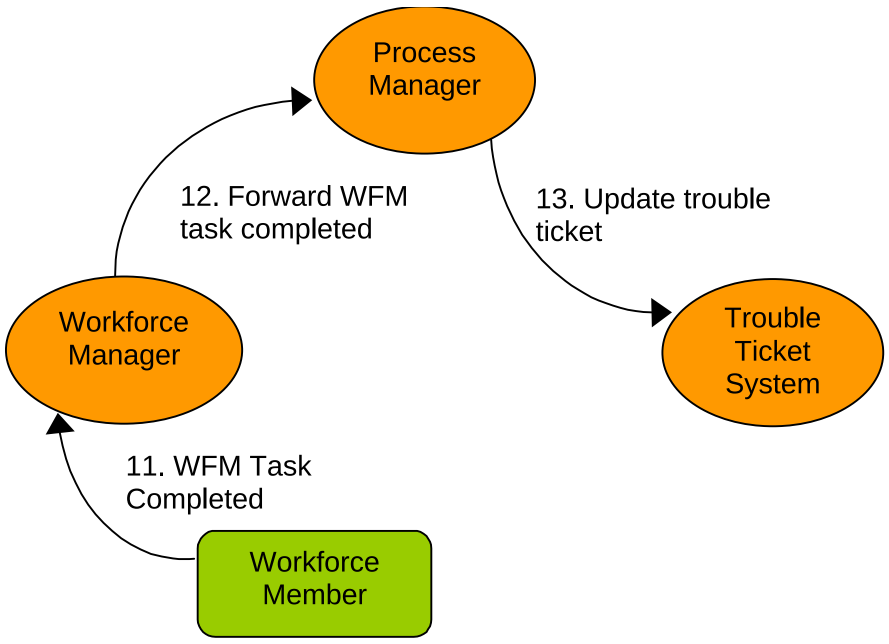


Figure 3. Problem Resolved

Steps 11-13 are illustrated in figure 3.

11. A member of the workforce, who has been assigned to the job by the Workforce Management system, updates the Workforce Management system upon its completion
12. The status of the job is forwarded to the Process Manager.
13. The Process Manager sends an update to the Trouble Ticket system stating that the task is complete and that the customer can be informed that the problem is resolved.

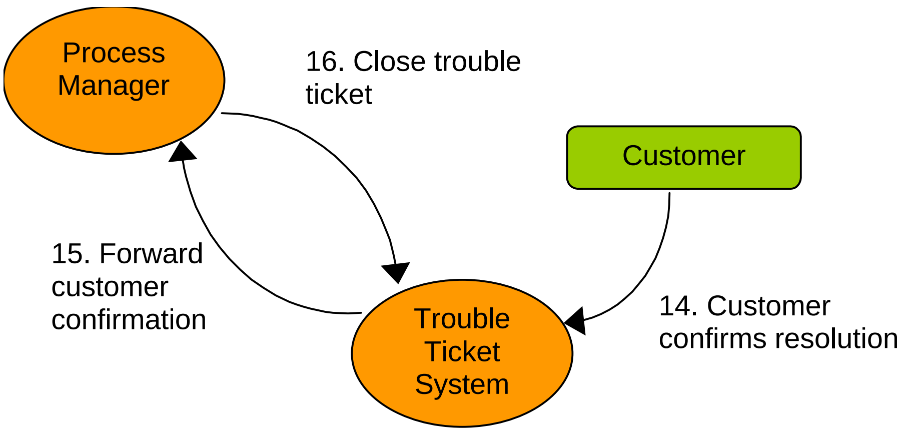


Figure 4. Close trouble ticket

Steps 14-16 are illustrated in figure 4.

14. The customer confirms that the problem is resolved.
15. The confirmation is forwarded to the process manager.
16. The process manager closes the trouble ticket.

The contents of the messages and the changes in state that they cause will be considered in the following sections.

2.2 Case Study Ontology

The scenario described in section 2.1 is encompassed by the ontology shown in figure 5.

A service problem is the central entity with which the scenario is concerned. It is embodied by six other key entities. An alarm is the means by which the problem is recognised. This is raised when a resource (generally a piece of hardware) fails or experiences a problem. It will be raised by some diagnostic function on the resource itself or closely related to it and contain an ID for the resource, the type of alarm, a unique ID for the alarm and a time stamp indicating when the fault was first detected. The affected resource, identified in the alarm by its ID is embodied by the Resource class which contains its full details. A resource generally forms part of a service which is provided to a customer. The inventory manager provides mappings between resources and services and customers, enabling the affected entities to be discovered. Once these are known, the trouble ticket can be created to handle the problem. Following evaluation of the problem, a task is created on the workforce management system to resolve it.

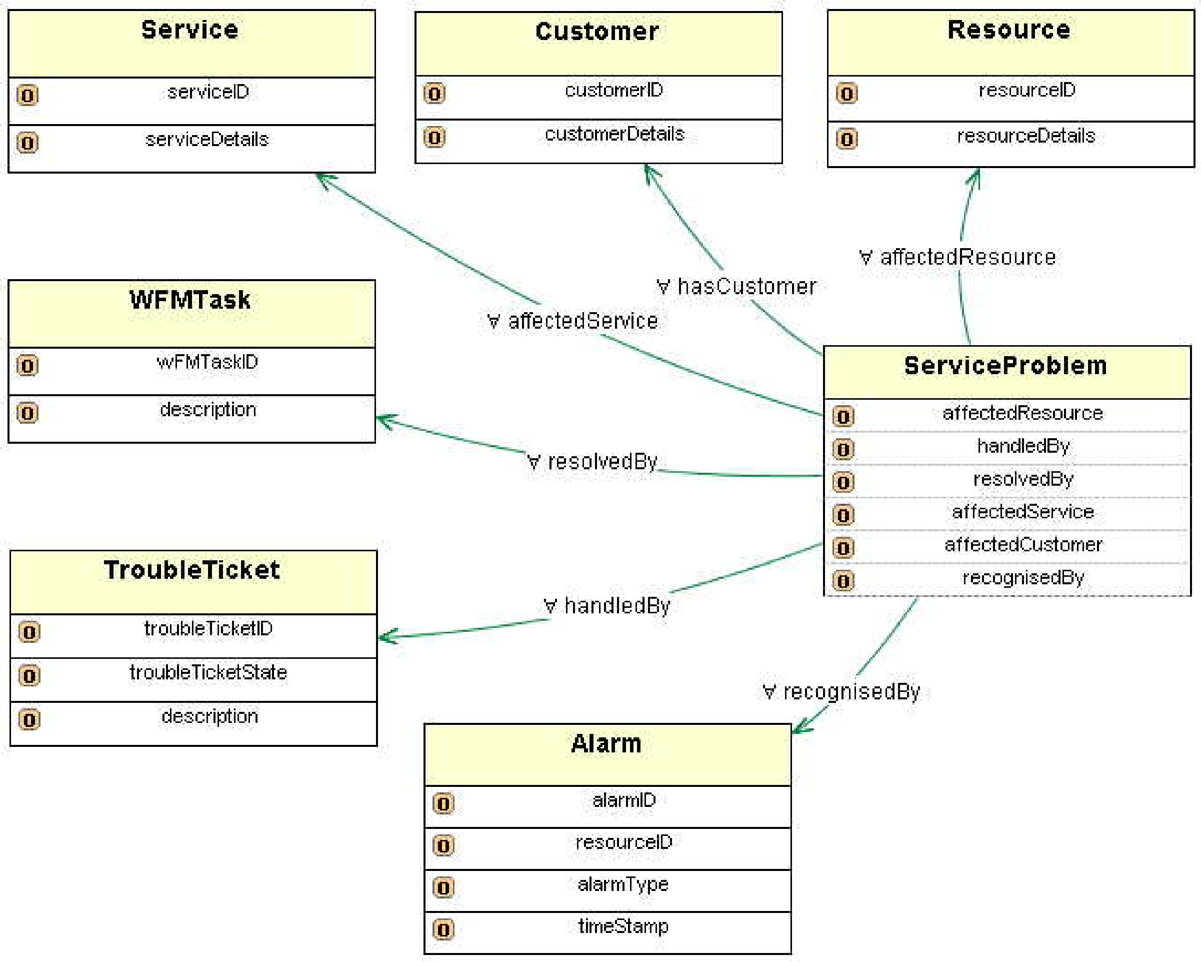


Figure 5. Case Study Ontology

3 Service Descriptions

This section identifies the services that are used in the case study then describes how they can be semantically annotated using OWL-S.

3.1 Required Services

This section further develops the storyboard to include considerations of the services that are required and the messages that must be passed between them. The four figures (1-4) of the storyboard are considered in turn. The following descriptions are considered from the point of view of the process manager. It is the job of the designer to design a composed web service that will handle incoming messages from other systems by using local or remote services to progress and resolve the problem.

Figure 1 is concerned with isolating the problem and initiating resolution. The first task it to receive the alarm and identify the source of it. The local alarmHandler service contains the operation getAlarmResource to extract the resourceID from the alarm (see figure 6).

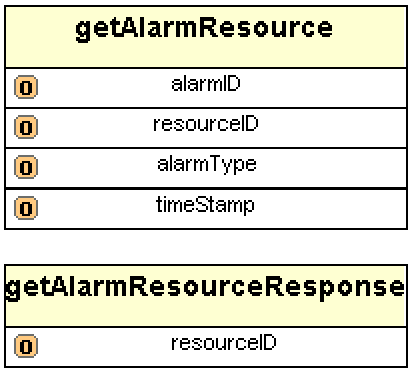


Figure 6. getResourceID messages

Once the resourceID has been determined, the service and customer details are required. The inventoryManagement service with its getServiceID and getCustomerID operations is required (see figure 7).

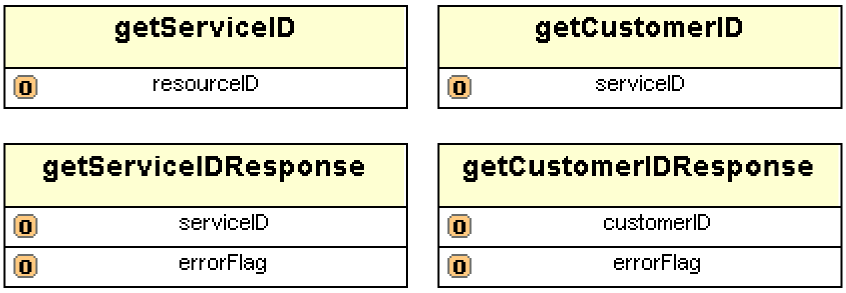


Figure 7. getServiceID and getCustomerID

Following this, the trouble ticket can be created using the troubleTicket service. This is actually a two step process. The ticket is created and then populated with the known information (see figure 8).

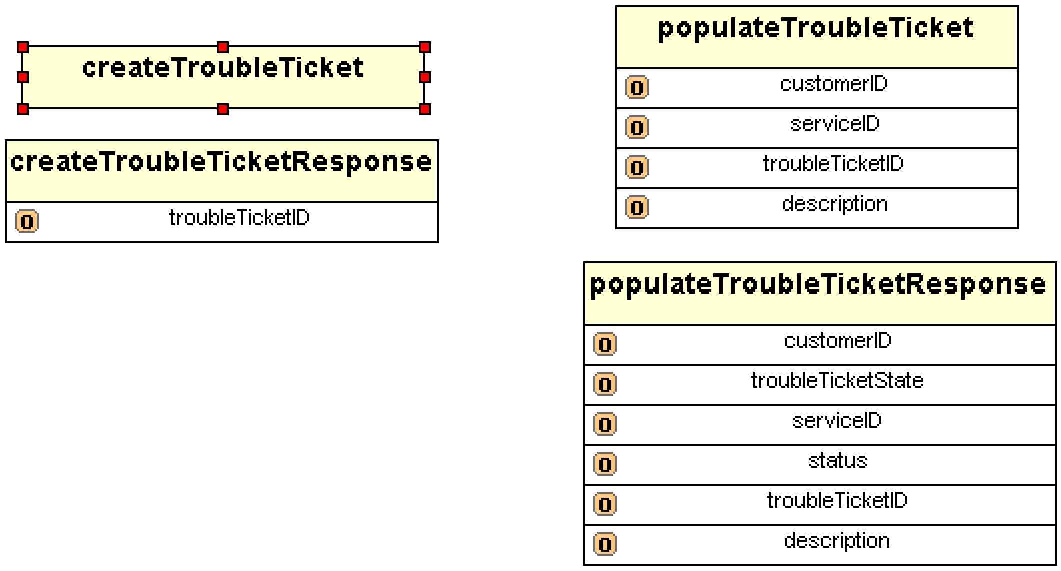


Figure 8. Create and populate trouble ticket.

The data returned by populateTroubleTicket service should then be stored locally (see figure 9). The response from this service will simply be an indication that the store worked or did not. At this stage the troubleTicketState is set to 'QUEUED'

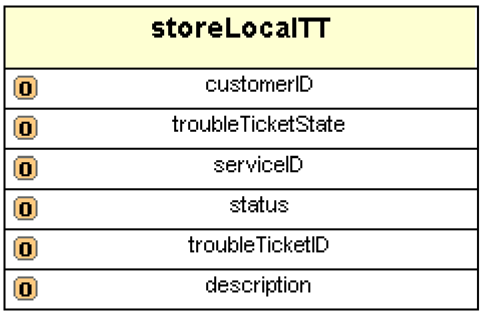


Figure 9. Store TT at Process Manager

The next part of the storyboard (see figure 2) is concerned with assigning the problem to a member of the workforce to resolve it. The first activity within the process manager is to receive a message from the trouble ticket system indicating that a job is required to fix the problem. Following this, a request for the full trouble ticket is made (see figure 10) and stored before a WFM task request is carried out. The troubleTicketStatus has been set to 'OPEN' at this stage.

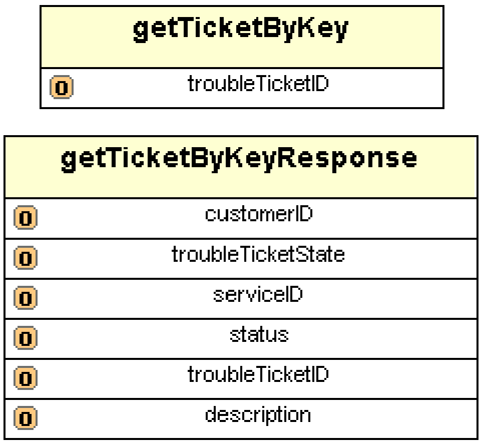


Figure 10. Get trouble ticket by key

A local store operation is then required (which is identical to figure 9). Along side this, a request to create a job is sent to the Workforce Management service (using the trouble ticket ID to simplify things) as shown in figure 11.

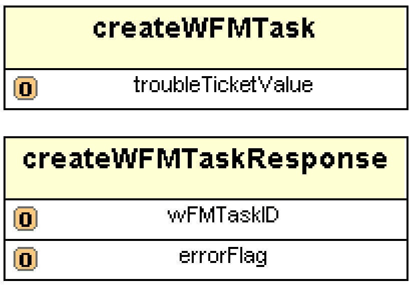


Figure 11. Create WFM task

The response from the Workforce management system is stored locally along with the appropriate trouble ticket (see figure 12).

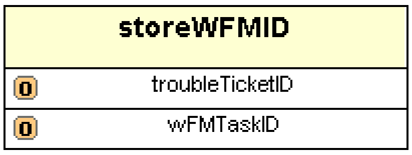


Figure 12. Store the task ID

The next major step (see figure 3) occurs when the workforce member indicates that the problem has been cleared. The process manager then updates the trouble ticket system. The first task is to get the appropriate trouble ticket for the wFMTaskID i.e. the reverse of the process in figure 12. Following this, a status update is sent to the trouble ticket system. Here the updateTTState service (see figure 13) can be used. This update the state of the trouble ticket and set the status flag which is used (amongst other things) to indicate whether the customer is aware of the fault status. These inputs should be 'hard-wired' to the appropriate settings for this stage by the designer i.e troubleTicketState is 'CLEARED' and status is'customerNotAdvised'. At this state, they are not aware. A local update is also required at this state (see figure 14).

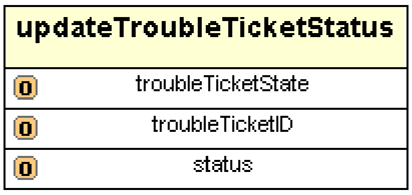
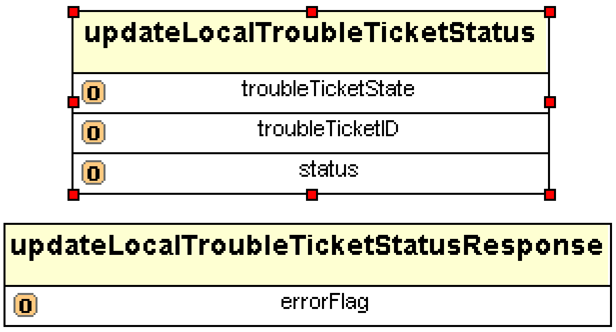


Figure 13. Update trouble ticket status

The final major task is to receive notification from the trouble ticket system that the customer has been notified and has confirmed that the fault has been cleared. This is followed up by the process manager closing the trouble ticket. The notification is handled by a local service which updates the local copy as shown if figure 14. At this stage, the troubleTicketState should be set to 'CLOSED' buy the designer.



updateLocalTroubleTicketStatus
troubleTicketState
troubleTicketID
status
updateLocalTroubleTicketStatusResponse
errorFlag

Figure 14. Update local trouble ticket status

The final step is to close the trouble ticket. Another updateTroubleTicketStatus (see figure 13) call is required here.

4 Domain Ontologies

One of the aims of the case study is to make use of existing ontologies that exist for the telecommunication sector and understand how they can be used to enhance service descriptions. This section describes the modelling work of the TeleManagement Forum[2] then illustrates how this can be converted to the Web Ontology Language (OWL) for use in the case study.

4.1 Next Generation OSS

The TeleManagement Forum's Next Generation OSS is a "comprehensive, integrated framework for developing, procuring and deploying operational and business support systems and software" [1]. It is available as a toolkit of industry-agreed specifications and guidelines that cover key business and technical areas including:

- Business Process Automation delivered in the enhanced Telecom Operations Map (eTOM™)
- Systems Analysis & Design delivered in the Shared Information/Data Model (SID)

The eTOM and SID have been considered in this project as ontologies in that they can provide a level of shared understanding for a particular domain of interest. The eTOM provides a framework that allows processes to be assigned to it. It describes all the enterprise processes required by a service provider and analyses them to different levels of detail according to their significance. It provides a reference point for internal process reengineering needs, partnerships, alliances and general working agreements [2]. The SID provides a common vocabulary allowing these processes to communicate. It identifies the entities involved in OSS and the relationships between them. The SID can therefore be used to identify and describe the data that is consumed and produced by the processes.

In order to make use of the eTOM and SID within the project, it was necessary to express them in a formal ontology language i.e. OWL. As identified in section 1, OWL provides a way to describe a domain in way that is understandable by both humans and computers.

[2] http://www.tmforum.org/

Techniques such as reasoning can then be carried out using the formalised ontology. Reasoning basically means that you can infer new facts from the facts that are known to you.

The eTOM and SID are subject to ongoing development by the TMF. The current version of the eTOM (3.6) is expressed in a set of documents although there are plans to provide a clickable HTML version (a previous version is already available in this form) and an XML version. The SID is also expressed in a set of documents but is also available as a set of UML[3] models.

UML (Unified Modelling Language) is a standard from the Object Management Group that aims to aid the process of specifying and developing software. However, its powerful visual approach means that it can also be used for business modelling. In order to capture the various aspects of complex systems, UML consists of 12 types of diagrams allowing descriptions of static application structure, dynamic behaviour and the organisation of application modules. The class diagram in the first category is the one most commonly used in the SID.

4.2 The eTOM Ontology

As stated in section 2, the eTOM can be regarded as a Business Process Framework, rather than a Business Process Model, since its aim is to categorise the process elements business activities so that these can then be combined in many different ways, to implement end-to-end business processes (e.g. fulfilment, assurance, billing) which deliver value for the customer and the service provider. [2]. The eTOM can be decomposed to lower level process elements. It is to these elements that business specific processes can be mapped.

Figure 15 shows the highest conceptual view of the eTOM known as the level 0 view. It shows the differentiation between operations processes and strategy and lifecycle processes and the five key functional areas (Market, Product & Customer, Service, etc.). It also shows the internal and external entities that interact with the enterprise (customers, employees, etc.).

[3] www.omg.org/**uml**/

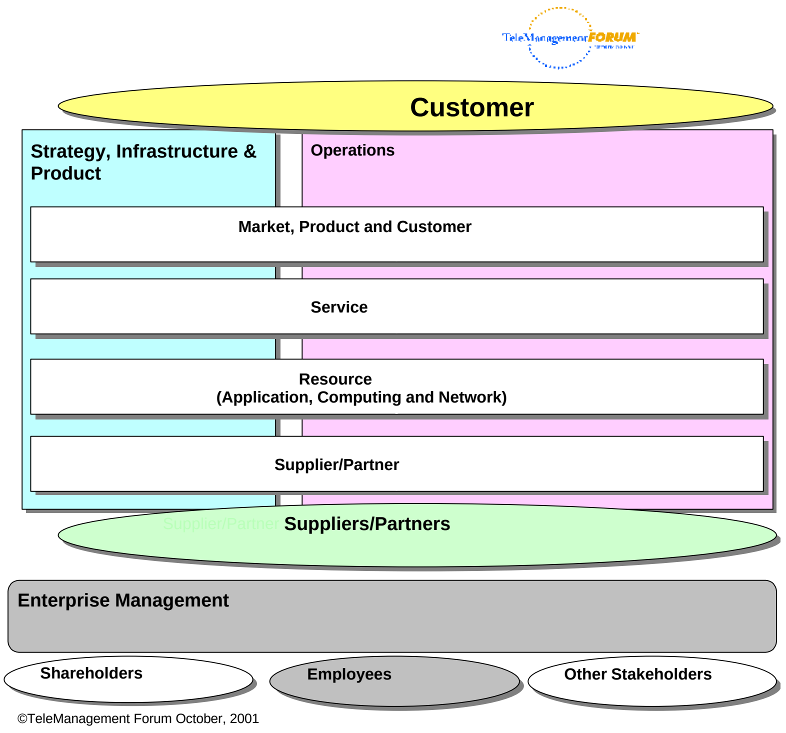


Figure 15. eTOM Business Process Framework—Level 0 Processes

Figure 16 shows the Level 1 Processes. The seven vertical groupings are the end-to-end processes that are required to support customers and manage the business. The horizontal functional processes are also divided between the process areas.

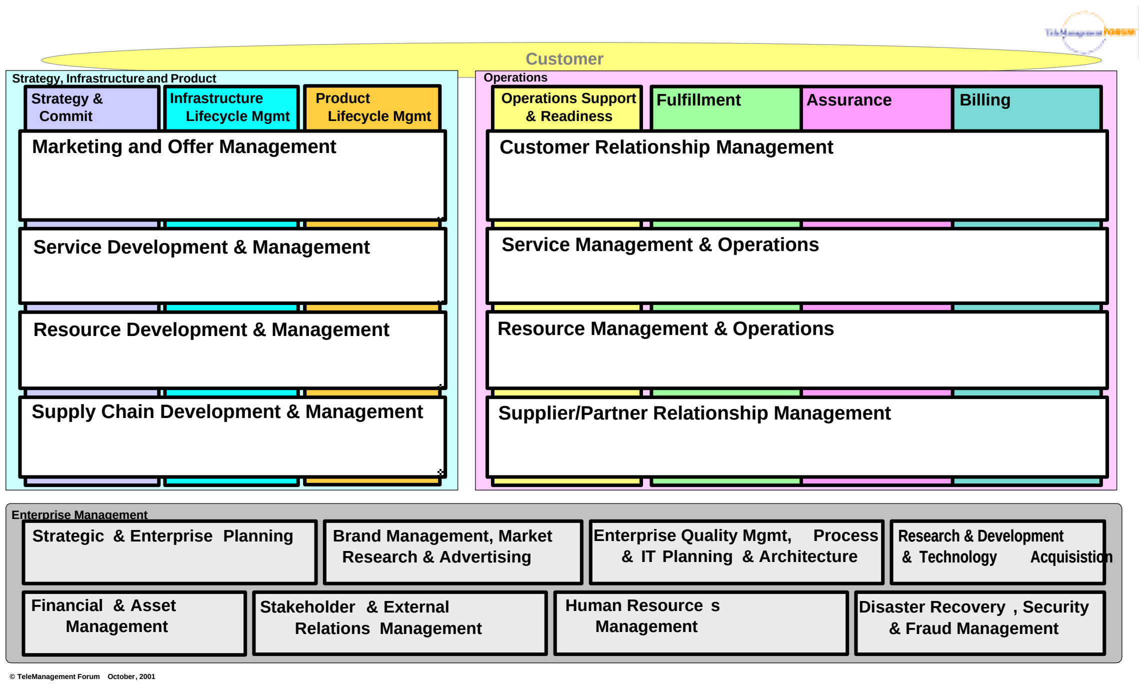
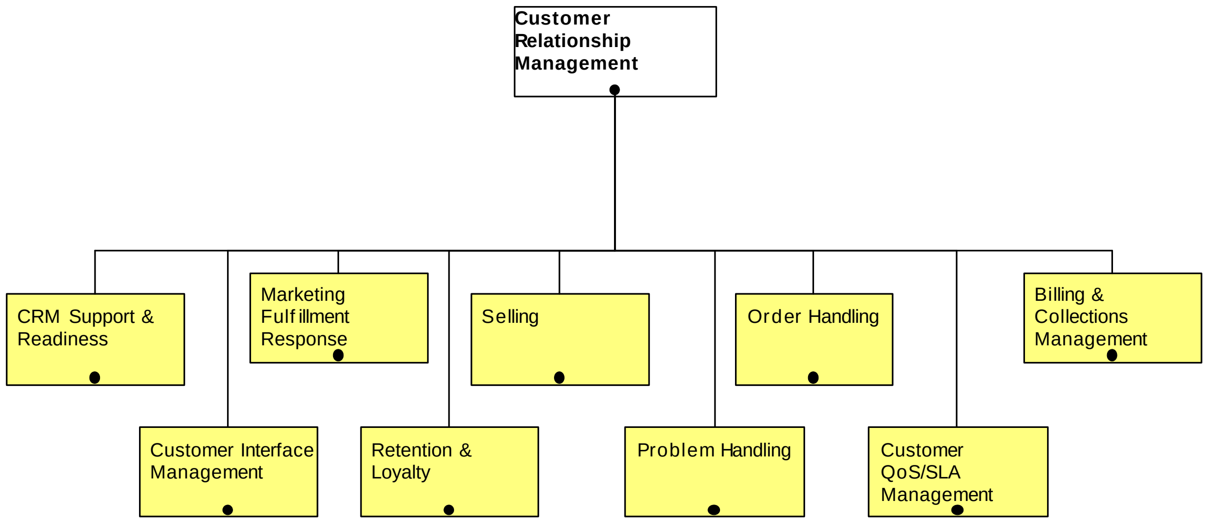

TeleManagement FORUM
Customer
Strategy, Infrastructure and Product
Strategy & Commit
Infrastructure Lifecycle Mgmt
Product Lifecycle Mgmt
Operations
Operations Support & Readiness
Fulfillment
Assurance
Billing
Marketing and Offer Management
Customer Relationship Management
Service Development & Management
Service Management & Operations
Resource Development & Management
Resource Management & Operations
Supply Chain Development & Management
Supplier/Partner Relationship Management
Enterprise Management
Strategic & Enterprise Planning
Brand Management, Market Research & Advertising
Enterprise Quality Mgmt, Process & IT Planning & Architecture
Research & Development & Technology Acquisition
Financial & Asset Management
Stakeholder & External Relations Management
Human Resources Management
Disaster Recovery, Security & Fraud Management
© TeleManagement Forum October, 2001

Figure 16. eTOM Business Process Framework—Level 1 Processes

This document will now consider a decomposition for one of these horizontal functional processes i.e. ‘Customer Relationship Management’. This grouping considers the fundamental knowledge of customers needs and includes all functionalities necessary for the acquisition, co-ordination, enhancement and retention of a relationship with a customer The level 2 process diagram for CRM is shown in figure 17.



Customer Relationship Management
CRM Support & Readiness
Marketing Fulfillment Response
Selling
Order Handling
Billing & Collections Management
Customer Interface Management
Retention & Loyalty
Problem Handling
Customer QoS/SLA Management

Figure 17: Customer Relationship Management Decomposition into Level 2 Processes

Each level 2 process is described in detail with a number of attributes. These are Process Name, Process Identifier a Brief and Extended Description and Known Process Linkages (n.b. The ‘Known Process Linkage’ attributes have not been completed in the current version of the eTOM). The data for the level 2 process ‘Problem Handling’ is shown below:

Process Name	Problem Handling
Process Identifier	1.06
Brief Description	Responsible for receiving trouble reports from customers, resolving them to the customer’s satisfaction and providing meaningful status on repair and/or restoration activity to the customer
Extended Description	Problem Handling processes are responsible for receiving trouble reports from customers, resolving them to the customer’s satisfaction and providing meaningful status on repair and/or restoration activity to the customer. They are also responsible for customer contact and support in relation to any service-affecting problems detected by the Resource Management & Operations processes or through analysis, including proactively informing the customer and resolving these specific problems to the customer’s satisfaction.
Known Process	

Linkages	

Level 2 processes can also be decomposed into level 3 processes. Figure 18 shows the diagram for 'Problem Handling'.

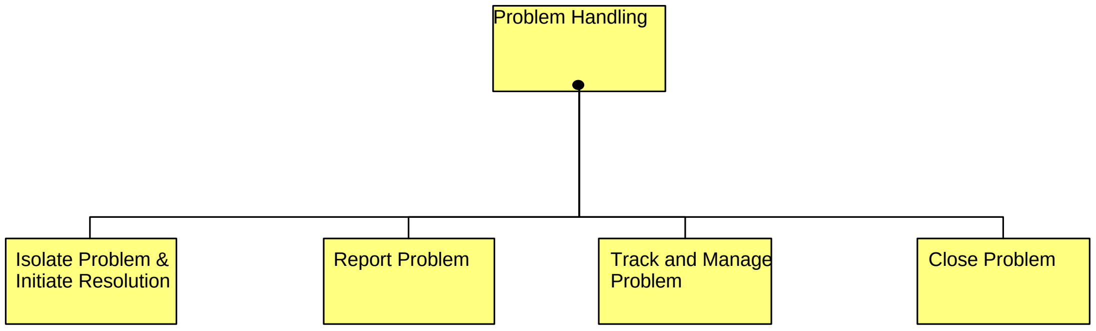


Figure 18: Problem Handling Decomposition into Level 3 Processes

Each level 3 process is described using the same attributes as level 2 processes. The description for 'Isolate Problem & Initiate Resolution' is:

Process Name	Isolate Problem & Initiate Resolution
Process Identifier	1.06.01
Brief Description	Receive & isolate problem, and initiate resolution actions
Extended Description	The purpose of this process is to register and analyze received trouble reports from customer; to register received information about customers impacted by service affecting problems, and reported problem information; to isolate the source / origin of the problem in order to determine what actions have to be taken; and to initiate the resolution of the problem
Known Process Linkages	

Representing the eTOM using an ontological formalism such as the Web Ontology Language (OWL) [3] is not complex. The process categories can be modelled using the OWL Class construct.

```
<owl:class rdf:about="CustomerRelationshipManagement">
```

The relationships in the process decompositions can be embodied using the OWL Subclass construct i.e. 'Problem Handling' can be represented as a subclass of 'Customer Relationship Management' as shown below:

```
<owl:class rdf:about="ProblemHandling">
    <owl:subClassOf rdf:resource=
            "#CustomerRelationshipManagement" />
    </owl:class>
```

The # symbol signifies that this is an element already defined in the local namespace.

Each process category is a class in the ontology. Invokable processes can then be categorised according to that ontology by defining them as instances of one or more classes in the ontology

```
    <owl:Thing rdf:about="#ActualProcess">
       <rdf:type rdf:resource="#ProblemHandling"/>
    </owl:Thing>
```

All process categories can inherit from a class ‘Process Category’ which has a number of properties i.e. ‘Process Name’, ‘Process Identifier’, ‘Brief Description’, ‘Extended Description’ and ‘Known Process Linkages’. The first four of these are OWL Datatype properties in that they have as a range one of the XML Schema Datatypes [4] (in this case String). The fifth property is an OWL Object property in that it’s range is another process category.

```
    <owl:DatatypeProperty rdf:ID=”ProcessName”>
        <rdfs:range rdf:resource=“xsd:string”/>
        <rdfs:domain rdf:resource  = “#ProcessCategory”/>
    </owl:DatatypeProperty>

    <owl:ObjectProperty rdf:ID=“KnownProcessLinkages”>
        <rdfs:range rdf:resource=“#ProcessCategory”/>
        <rdfs:domain rdf:resource=“#ProcessCategory”/>
    </owl:ObjectProperty>
```

The contents of these properties are particular to the process category in question. Since these are represented as classes as opposed to instances it is necessary to use the OWL hasValue restriction on the class description. All instances of the class will then inherit the contents of the properties of its parent class(es)

```
    <owl:Class rdf:ID="#Isolate Problem & Initiate Resolution">
      ...
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:onProperty rdf:resource="#ProcessIdentifier" />
          <owl:hasValue rdf:resource="1.06.01" />
        </owl:Restriction>
      </rdfs:subClassOf>
    </owl:Class>
```

In OWL, classes can be subclasses of more that one class (this is similar to the concept of multiple inheritance in Object Orientated Analysis). This permits the representation of e.g. the ‘Billing and Collections Management’ process category as a subcategory of both the ‘Customer Relationship Management’ Horizontal Process Group and the ‘Billing’ Vertical Process Group as is the case in the eTOM.

4.2.1 Using the eTOM ontology

As stated above, classes in the eTOM ontology can be used to provide a categorisation for invokable processes within an organisation. Section 5 describes how such processes can be represented as Semantic Web Services using the OWL services layer OWL-S. One of the aims of OWL-S is to improve service discovery by relating services instances to service categories within an ontology (see section 5.4). The aim is to use the ontology to reason about appropriate services. For example, a solution designer seeking a service could specify their search goal by selecting one of more of the concepts in the ontology. The goal can then be expanded using the relationships in the eTOM ontology – both Sub / Superclass and the 'Known Process Linkages' property allow this to occur. For example, if the goal is to discover services in the 'Report Problem' category a search can be carried out by expressing this category as the search criteria. If no suitable services instances are found the user can modify their search by choosing service categories specified in the 'Known Process Linkages' property or by widening their search to the Superclass which in this case is 'Problem Handling'. The eTOM ontology was produced using an Ontology Editor called Protégé [5] from Stanford University. The editor has support for OWL.

4.3 The SID Ontology

The SID [6] is much more complex than the eTOM in both its aims and form. It provides a data model for a number of domains described by a collection of concepts known as Aggregate Business Entities. These use the eTOM as a focus to determine the appropriate information to be modelled. As has already been stated, the SID can be described as an ontology that defines the semantics of its domain. The aims of this work are to express this ontology in a form that allows it to be exploited by the emerging techniques and tools of the Semantic Web. The benefits of the SID are:

- it allows IT investments to be reused, by standardising their definition & behaviour
- it allows for simplification of information management, by providing a common terminology and reducing unnecessary variation
- it allows for unification of information both within an enterprise and between enterprises
- it allows for a combined commercial and technical framework

These benefits then enable business benefits relating to cost, quality, timeliness and adaptability of enterprise operations, allowing an enterprise to focus on value creation for their customers [6].

The SID includes:

- things in which the business is interested (domain entities)
- how they are related to one another (associations)
- key details about those things which help to define them unambiguously (domain-level attributes)

Figure 19 shows a high level view of the SID with concepts and the relationships between them.

The SID is expressed as a combination of textual descriptions in a set of documents and as a set of UML class models. It is these models that provide the most scope for creating OWL ontologies. Two approaches have been used within this task. Both are described in the following sections. The first is a manually built ontology that relies upon the individual to create the ontology using an editor. This is a time-consuming process but enables an accurate representation to be built. The second is an automatic approach using a tool to interpret the models and create the ontology. UML models can be exported to XMI (XML Metadata Interchange) [7] which is a XML-based language for exchanging data between different tools. This allows a third-party tool to import a UML model, create its own data model and output in another format or language e.g. OWL. Both approaches rely upon mappings between the UML representations and OWL constructs. These will be considered in the following section.

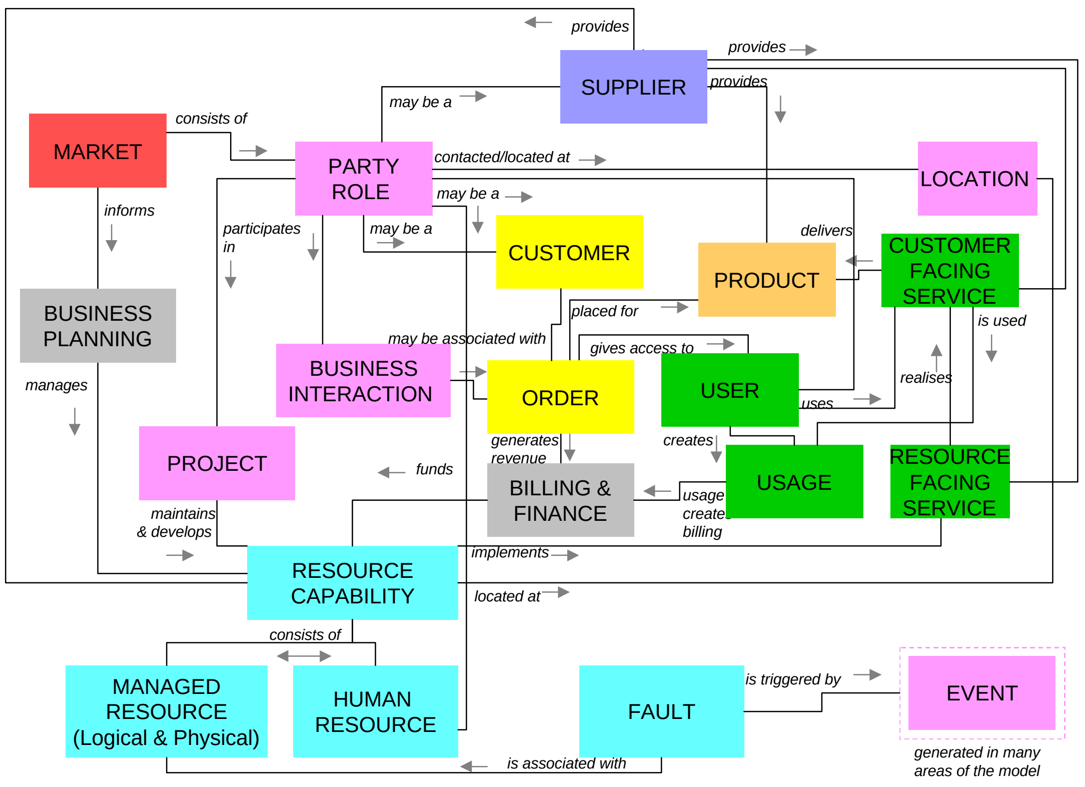


Figure 19. High Level view of the SID

4.3.1 Converting UML to OWL

Various approaches concerned with conversion between UML and OWL exist. These are summarised by Falkovych et al [8] who recognise that 'the wide acceptance of UML makes it an ideal language to be used by a critical mass of people to build high quality models of information semantics for the semantic web'. Two different use cases exist. The first is due to the fact that ontology representation languages lack visual modelling tools (although these are beginning to emerge and improve in quality). As such, tools such as Rational Rose can be used to provide support for modelling complex ontologies and managing the ontology development process. The second use case (which is the one this report is concerned with) addresses the problem of reusing knowledge previously specified as UML in a form that allows it to be 'on the Web' and can be reasoned with.

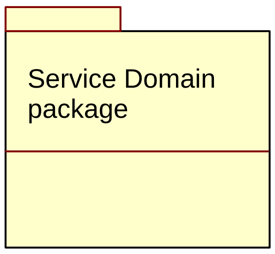
Many of the elements of UML class diagrams have an obvious relationship with elements in OWL e.g. classes in UML have a direct counterpart in OWL. These are now considered.

4.3.1.1 Package Element.

The UML package element is a logical abstraction element that exists as a container for lower level elements of the UML model. This can be represented using the OWL ontology element since this is also a container for description concerned with a particular domain or sub-domain. A UML package element will typically contain a *name*, a *property ID* and a *description*. These can be mapped to the *ID*, *label* and *comment* properties of the OWL ontology element.

The following example shows an OWL representation for the Package element.

UML:



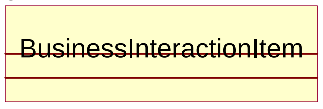
OWL:

```
<owl:Ontology rdf:ID="Service Domain package">
    <rdfs:label>Service Domain package</rdfs:label>
    <rdfs:comment>This package contains entities related to the
            service domain</rdfs:comment>
    <rdfs:versionInfoVersion> 1</rdfs:versionInfo>
</owl:Ontology>
```

4.3.2 Class

The UML *Class* element describes a set of objects and basic types. This is also the case with the OWL *Class* element.

UML:



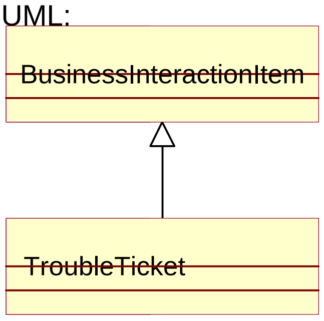
OWL:

```
<owl:class rdf:ID="BusinessInteractionItem"/>
```

4.3.3 Generalisation

In UML, a class can exist as a generalisation for one or more other classes. The generalisation element is synonymous with the OWL:subClassOf construct:

UML:



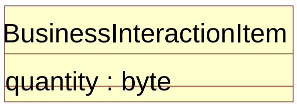
OWL:

```
<owl:class rdf:about="TroubleTicket">
    <owl:subClassOf rdf:resource="#BusinessInteractionItem" />
</owl:class>
```

4.3.3.1 Attributes

UML classes can contain attributes. These are local in scope to the class in which they are defined. The value of an attribute is generally a data value e.g. an integer or a string although it can also have an object as it's type. The most appropriate mapping in OWL for a UML attribute is the owl:DatatypeProperty. The range of these properties must always be a member of one of the XML standard datatypes (xsd). The scope of these properties is not limited to any particular class. This could prove a problem in translation as two or more UML attributes in different scopes may have the same name. To overcome this problem a unique identifier must be added to the OWL property in order to distinguish it from others:

UML:



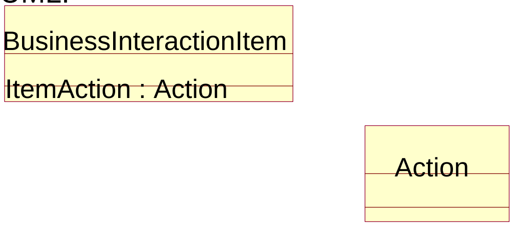
OWL:

```
<owl:DatatypeProperty rdf:ID=
            "BusinessInteractionItem_quantity">
    <rdfs:range rdf:resource = "xsd:byte"/>
    <rdfs:domain rdf:resource  = "#BusinessInteractionItem"/>
</owl:DatatypeProperty>
```

Note that the ID of the property is appended with the relevant class name to ensure uniqueness.

A stated above, UML attributes may also have objects as values. In this case owl:ObjectProperty is the most appropriate mapping. This also required a unique identifier to be created in the transformation process.

UML:



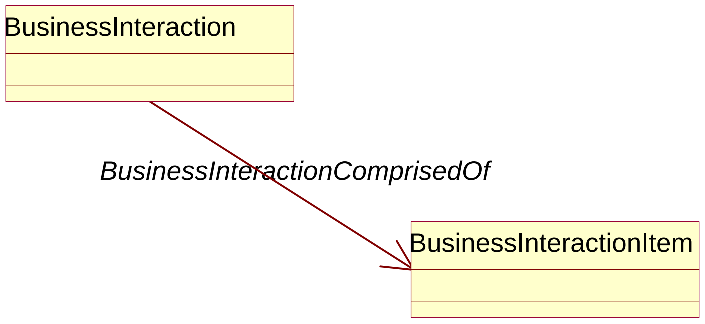
OWL:

```
<owl:ObjectProperty rdf:ID=
            "BusinessInteractionItem_ItemAction">
    <rdfs:range rdf:resource = "#Action"/>
    <rdfs:domain rdf:resource  = "#BusinessInteractionItem"/>
</owl:ObjectProperty>
```

4.3.3.2 Associations

The mapping of the various UML associations is the most problematic of the transformations. In the general case, the mapping is simple. The owl:ObjectProperty construct can be used:

UML:

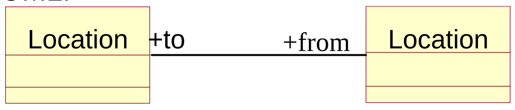


OWL:

```
<owl:ObjectProperty rdf:ID="BusinessInteractionComprisedOf">
    <rdfs:range rdf:resource="#BusinessInteractionItem"/>
    <rdfs:domain rdf:resource ="#BusinessInteraction"/>
<owl: objectProperty>
```

The above example uses a unidirectional association as signified by the arrow. It would not make sense to apply the association in the other direction. In UML it is also possible to have associations that are bidirectional. These are generally accompanied by roles which describe the role of each party in the association. In the UML diagrams roles can be identified by their '+' prefix. These are not possible in OWL so it is necessary to create two datatype properties and then declare them as the inverse of each other using the owl:inverseOf construct. It makes sense to use the roles to name the OWL properties in this example.

UML:



OWL:

```
<owl:objectProperty rdf:ID="Location_from">
    <rdfs:range rdf:resource="#Location"/>
    <rdfs:domain rdf:resource ="#Location"/>
<owl:objectProperty>

<owl:objectProperty rdf:ID="Location_to">
    <owl:inverseOf rdf:about="#Location_from"
    <rdfs:range rdf:resource="#Location"/>
    <rdfs:domain rdf:resource="#Location"/>
</owl:objectProperty>
```

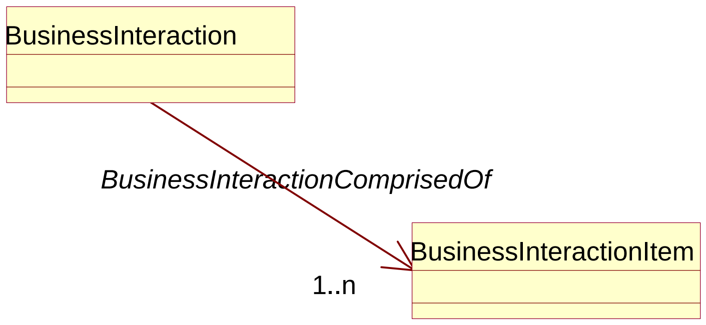
If the association in the above UML example is named 'journey' then this can be created as an abstract objectProperty that the other two properties can be subclasses of. If it is not named then a unique key must be created for it. In the former case, the additional OWL would be:

```
<owl: objectProperty rdf:ID="Location_from">
    <rdfs:subPropertyOf rdf:resource="Journey"/>
<owl: objectProperty>
```

```
<owl: objectProperty rdf:ID="Location_to">
   <rdfs:subPropertyOf rdf:resource="Journey"/>
<owl: objectProperty>
```

Associations can also be subject to multiplicity constraints. These are expressed at the endpoints of associations and can either be a single value e.g. 0 or 1 or a range of values e.g. 0..1. OWL has cardinality which allows the mapping of these constraints. There are three cardinality constraints: minCardinality, maxCardinality and cardinality (for specifying precise values). The semantics of cardinality in OWL are different to those in UML e.g. if a property has a cardinality of 1 but the range of the property has two entities, a UML validation would produce an error. An OWL reasoner would assume that the two entities are the in fact just one entity but with two different names. This is due to the closed world semantics of UML (which is akin to databases) where you assume that facts not stated are false and the open world semantics of OWL where you cannot assume that facts not stated are false (the fact in this case being that the two entities are <owl:sameAs>). Care should be taken to ensure that intended semantics are preserved following a conversion.

UML:



In OWL the cardinality constraints can be introduced as property restriction. This uses the subclass construct to create an anonymous class that is the subclass of all objects that satisfy the restriction. This form could have been used in the previous property examples. It is more flexible but can initially appear rather confusing. The restriction for the BusinessInteractionItemComprisedOf association is shown below:

```
<owl:Class rdf:ID="BusinessInteraction">
   <owl:subClassOf>
          <owl:Restriction>
                 <owl:minCardinality>1</daml:cardinality>
                 <owl:onProperty rdf:resourse=
                        "#BusinessInteractionComprisedOf"/>
                 <owl:allValuesFrom rdf:resourse=
                        "#BusinessInteractionItem"/>
          </owl:Restriction>
   </owl:subClassOf>
</owl:Class>
```

4.3.3.3 Association Class

An association in UML may also have an association class attributed to it that may contain additional attributes about the association. This is shown in figure 7 below where a BusinessInteractionPrice is included as an association class for the

BusinessInteractionInvolvesProductOffering association. This is problematic in OWL because a property can exist as the domain of another property. Neither Falkovych [8] or Koryak [9] provide satisfactory solutions to this issue.

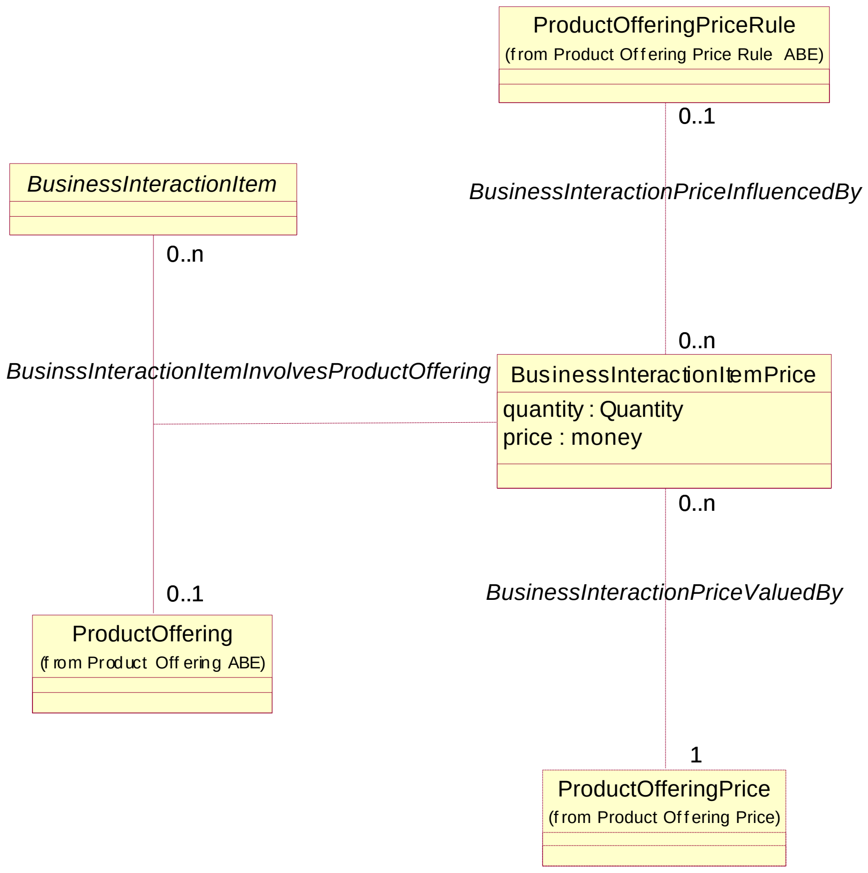

ProductOfferingPriceRule
(from Product Offering Price Rule ABE)
0..1
BusinessInteractionPriceInfluencedBy
BusinessInteractionItem
0..n
0..n
BusinssInteractionItemInvolvesProductOffering
BusinessInteractionItemPrice
quantity : Quantity
price : money
0..n
0..1
BusinessInteractionPriceValuedBy
ProductOffering
(from Product Offering ABE)
1
ProductOfferingPrice
(from Product Offering Price)

Figure 20. Association Class Example

One approach would be to remodel the UML to remove the association class whilst maintaining the semantics. In the example shown in figure 20 this could be carried out by inserting an extra class: BusinessInteractionItemInvolvesProductOffering. By replacing the association with a class, an association can be made to the BusinessInteractionPrice class. The altered class diagram is shown in figure 21. It is not as intuitive as the original but the semantics are retained and a conversion to OWL is possible.

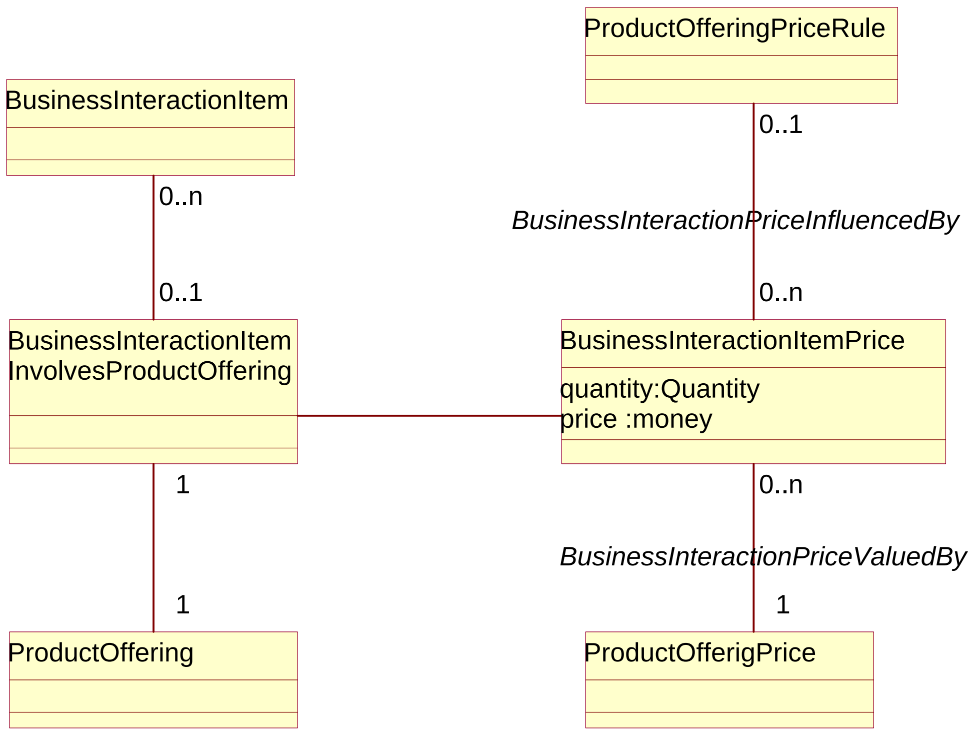
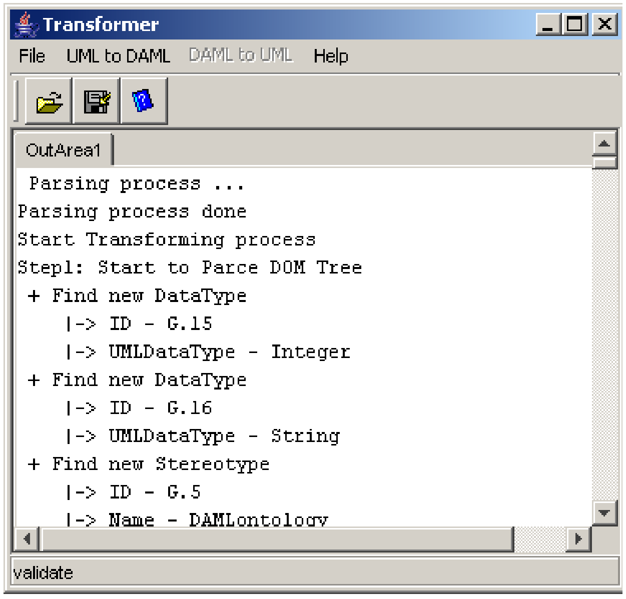

ProductOfferingPriceRule
BusinessInteractionItem
0..1
0..n
BusinessInteractionPriceInfluencedBy
0..1
0..n
BusinessInteractionItem InvolvesProductOffering
BusinessInteractionItemPrice
quantity:Quantity
price :money
1
0..n
BusinessInteractionPriceValuedBy
1
1
ProductOffering
ProductOfferigPrice

Figure 21. Association Class Removed

4.4 Automatic Conversion

The Flexible Transformation System [9] is a tool that has been developed at the Free University of Amsterdam. It is currently in prototype form. The tool aims to provide conversion for both UML to DAML+OIL and OWL and DAML+OIL and OWL to UML. The first of these is important for this work. The tool can handle XMI 1.0 [7] as input. It parses the input using a XML parser library and then creates a language neutral Document Object Model. The system then analyses the DOM and classifies each element according to its type. Each element is then transformed to the output language using a mapping table. This table is a separate file allowing the nature of the mapping to be changed or updated without altering the program code. A screenshot for the FTS is shown in figure 22.



```
 Parsing process ...
Parsing process done
Start Transforming process
Step1: Start to Parce DOM Tree
 + Find new DataType
    |-> ID - G.15
    |-> UMLDataType - Integer
 + Find new DataType
    |-> ID - G.16
    |-> UMLDataType - String
 + Find new Stereotype
    |-> ID - G.5
    |-> Name - DAMLontology
```

Figure 22. The Flexible Transformation System

In order to convert the SID to OWL, it is necessary to use the Rational Rose[4] tool to export it as XMI (since the SID has been developed using Rose and is only currently available in the proprietary Rose format). An export toolkit for Rose is required for this purpose. At the time of writing, only a small subset of the SID has been converted.

There are a number of issues to be resolved with the SID conversion:

- Many bi-directional associations do not have role names. This causes a problem when naming the two OWL properties (i.e. one in each direction)
- The superclass of the two OWL properties in the issue above is not named correctly with the bi-directional association name from the UML
- Association classes are not handled at all
- The SID uses 'n' to indicate there is no maximum multiplicity. This is translated as '-1' by the FTS. It would be better not to translate this at all as it is not required in OWL (no maximum cardinality is assumed to be the case unless otherwise stated).

Work is ongoing to address these issues

4.5 Mapping the SID to the Case Study

The ontology described in Figure 5 Can be mapped to the SID in order to allow fuller descriptions of the entities to be made. This allows the SID entities to model the complex data required for customer, services, etc. and the scenario ontology to be, in the main, only concerned with ensuring the unique identified for these are communicated. The customer entity maps onto the part of the SID model shown in figure 23.

[4] IBM Rational Software, http://www.rational.com/

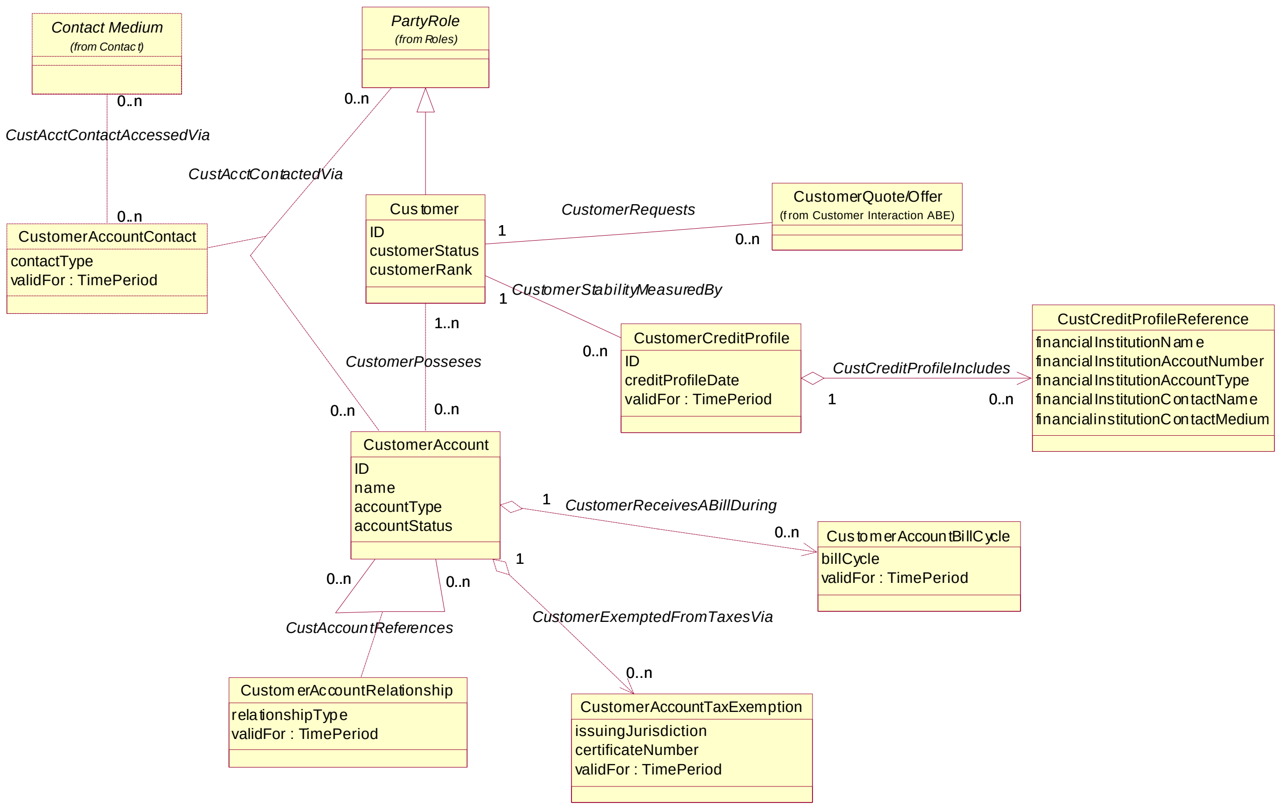


Figure 23. Customer Business Entity Model

5 Semantic Services

The services used in the case study are, wherever possible, based upon the those defined by the OSS/J [10] consortium. The requirements of e-business, the proliferation of mobile workers, and the ever-increasing need for bandwidth have led to an increased need to reduce the development costs of communications services. Current implementations of OSS technology are unsuitable for rapidly increasing scale of networks, the diversity of communications technology, shortened time to market for new services, and heightened expectations for availability and reliability. The OSS through Java Initiative was started to develop solutions to allow service providers develop carrier-grade OSS solutions in response to the rapidly change business environment.

The members of the OSS through Java Initiative are convinced that the fastest and most flexible way to develop OSS solutions based on reusable components and container technology [10]. The client can access those components through either tightly or loosely coupled mechanisms. The two interface mechanisms supported by the APIs are the Java Message Service (JMS), using XML based messages, and Java Value Types (JVT), using stateless Java session beans.

The OSS/J consortium believe that the Java 2 Platform, Enterprise Edition (J2EE) technology is the simplest and most reliable means of implementing such an architecture. The consortium members are promoting the adoption of this component-based approach, to developing OSS solutions, by undertaking a number of development activities aimed at kickstarting a component marketplace for OSS solutions. Moreover, the members of the consortium are convinced that such an approach has benefits for all stakeholders along the value chain: equipment vendors, independent software vendors (ISVs), system integrators, and service providers.

To that end, the goals of the OSS/J initiative include:

- The development, through the Java Community Process (JCP) program, of various component API specifications, Reference Implementations, and Technology Compatibility Kits, for OSS integration and deployment.
- The development of multi-vendor demonstrations based on the OSS/J APIs.

The OSS/J consortium concentrated upon the development of the XML messaging and JVT interfaces, with the associated Reference Implementation and test suites. The APIs are described through the use of Xml Schema Description (XSD) files, which define the form and content of XML messages. However, at the moment, there are no standards for mapping to and from the OSS/J APIs to web services. A standard web service interface, with associated use cases, is currently under development, by the OSS/J Architecture Board.

An important point is that there should be very little OSS/J specific regarding this WSDL mapping process [9]. Already today it is possible to generate WSDL files from Enterprise Java Bean (EJB) interfaces. There are already proprietary bridges between XML requests embedded in Web Services and EJB calls. At the moment, this process is not standardised and the tools available are working only for simple Java classes and not with the OSS/J complex interfaces.

The OSS/J initiative also does not provide direct support for Business-to-Business (B2B) protocols, such as ebXML and RosettaNet. However, the OSS/J APIs do support XML based messaging interfaces, which is an enabler for B2B. Since the XML payload is transport independent it is possible that you could transport the OSS/J XML payloads with SOAP and other messaging services. In theory, it should be possible to bridge the OSS/J enterprise components with B2B applications via some B2B Gateway in charge of XSLT and transport adaptations. The use for XML based messaging makes OSS/J applicable to the case study and provides a view of the likely granularity the service interfaces that will be exposed by component vendors in the future.

5.1 WSDL Descriptions

The case study makes use of OSS/J interfaces in order to ensure that the services modelled and their level of granularity are as close to reality as possible. Although OSS/J has yet to be adopted commercially, extensive work by the OSS/J consortium has gone on to ensure that it meets the requirements of product vendors and consumers in delivering interfaces at the appropriate level.

In order to make use of the interfaces in this case study, it was necessary to wrap them as WSDL web services since OWL-S only supports a grounding to WSDL. This section will describe the WSDL produced for one of these services as a result of this wrapping process. The service in question is the TroubleTicket service and its getTroubleTicketByKey operation.

WSDL describes a web service and a set ports to which operations that can be carried out upon the web service can be attributed.

```
<service name="TTicket">
```

```
  <port name="TTicketSoapPort" binding="s0:TTicketSoapBinding">
    <soap:address location="http://BTG022823/TTWSWeb/TTicket.jws"/>
  </port>
</service>
```

In this case a SOAP port is described so a binding for the SOAP port is given together with the address of the invokable web service.

The binding defines the protocol format and the message format for the service. It also creates a link between the port and a type. Operations are also defined in terms of the SOAP action that must be carried out and the encoding of the inputs and outputs.

```
<binding name="TTicketSoapBinding" type="s0:TTicketSoapType">
  <soap:binding transport=http://schemas.xmlsoap.org/soap/http
        style="document"/>
  <operation name="getTTicketByKey">
    <soap:operation soapAction="http://www.openuri.org/getTTicketByKey"
        style="document" />
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
  </operation>
</binding>
```

The port type referred to in the binding gives a description of the operations that can be carried out upon it. The following fragment shows the port type definition for the TTicket service together with the getTTicketByKey operation:

```
<portType name="TTicketSoapType">
  <operation name="getTTicketByKey">
    <input message="s0:getTTicketByKeySoapIn" />
    <output message="s0:getTTicketByKeySoapOut" />
  </operation>
</portType>
```

The operation is defined in terms of its input and output messages. Messages can have many parts which are named, although the messages for the getTTicketByKey operation (shown below) have just one part each.

```
<message name="getTTicketByKeySoapIn">
  <part name="parameters" element="s0:getTTicketByKey" />
</message>
<message name="getTTicketByKeySoapOut">
  <part name="parameters" element="s0:getTTicketByKeyResponse" />
</message>
```

These message parts are defined by XSD elements which can themselves be complex types. The element for the input message of the operation is shown below. In this case it is a string that holds the ID of the trouble ticket required.

```
<s:element name="getTTicketByKey">
  <s:complexType>
```

```
    <s:sequence>
      <s:element name="TTicketID" type="s:string" minOccurs="0" />
    </s:sequence>
  </s:complexType>
</s:element>
```

The element for the output is shown below.

```
<s:element name="getTTicketByKeyResponse">
  <s:complexType>
    <s:sequence>
      <s:element name="getTTicketByKeyResult" type="#TTicket"
         minOccurs="0" />
    </s:sequence>
  </s:complexType>
</s:element>
```

This element refers to a a result element of type TTicket. This is shown below.

```
<s:complexType name="TTicket">
  <s:sequence>
    <s:element name="TTicketID" type="s:string" minOccurs="0" />
    <s:element name="CustomerID" type="s:string" minOccurs="0" />
    <s:element name="ServiceID" type="s:string" minOccurs="0" />
    <s:element name="Description" type="s:string" minOccurs="0" />
    <s:element name="Location" type="s:string" minOccurs="0" />
    <s:element name="State" type="s:int" />
    <s:element name="Status" type="s:int" />
  </s:sequence>
</s:complexType>
```

5.2 Service Grounding

In OWL-S, a service grounding creates a link between the semantic description of a service and the service itself which is described in WSDL as above. One aim of the case study the aim is to illustrate discovery and composition at the level of WSDL operations e.g. getTTicketByKey. For this reason, the decision has been made to model operations as OWL-S services. This is because a service has only one service profile, which is the means by which discovery is carried out. If a WSDL service i.e. TTicket had been modelled as an OWL-S service, then the profile would not allow advertisments of the operations within the service to be made. The following fragment identifies the atomic processes in the OWL-S service.

```
<grounding:WsdlGrounding rdf:ID="getTTicketByKey_Grounding">
  <grounding:hasAtomicProcessGrounding
        rdf:resource="#WsdlGrounding_getTTicketByKey" />
</grounding:WsdlGrounding>
```

The following fragment further develops the grounding by identifying the inputs and outputs required by the atomic process. Mappings are created between the inputs described in the process model and those from the WSDL.

```
<grounding:WsdlAtomicProcessGrounding
        rdf:ID="WsdlGrounding_getTTicketByKey">
  <grounding:owlsProcess rdf:resource="&getTTicketByKey_process;
```

```
        #getTTicketByKey_Process"/>
    <grounding:wsdlOperation rdf:resource="#getTTicketByKey_operation"/>
    <grounding:wsdlInputMessage>
      <xsd:anyURI rdf:value="&getTTicketByKeyWSDL;#getTTicketByKeySoapIn" />
    </grounding:wsdlInputMessage>

    <grounding:wsdlInputs rdf:parseType="Collection">
      <grounding:WsdlInputMessageMap>
        <grounding:owlsParameter rdf:resource="
                 &getTTicketByKey_process;#TTicketID_In"/>
        <grounding:wsdlMessagePart>
          <xsd:anyURI rdf:value="&getTTicketByKeyWSDL;#TTicketID" />
        </grounding:wsdlMessagePart>
      </grounding:WsdlInputMessageMap>
    </grounding:wsdlInputs>
  </grounding:WsdlAtomicProcessGrounding>
```

The remaining requirement of the grounding is to link the atomic service to the WSDL port and the operations on that port as shown below.

```
<grounding:WsdlOperationRef rdf:ID="getTTicketByKey_operation">
  <grounding:portType>
    <xsd:anyURI rdf:value="&getTTicketByKeyWSDL;#TicketSoap" />
  </grounding:portType>
  <grounding:operation>
  <xsd:anyURI rdf:value="&getTTicketByKeyWSDL;#getTTicketByKey" />
  </grounding:operation>
</grounding:WsdlOperationRef>
```

5.3 Service Model

The Service Model describes how the service works; it describes what happens when the service is executed. The Process Model is a subclass of the Service Model and gives a detailed perspective of the service. The Process Model has two main components:

- Process, which describes a service in terms of inputs, outputs, preconditions, effects, component sub-processes, and aims at enabling planning, composition and agent/service interoperation — a process is defined as an entity in the process ontology, described below,

- Process Control Model, which allows agents to monitor the execution of a service request — this is defined in the process control ontology, described below.

The primary class in the process ontology is Process, which can be viewed as one of three types (each of which is a subclass of Process):

- Atomic — these are directly invocable, have no sub-processes, and execute in a single step, from the perspective of the service requester.

- Simple — these are not invocable and are not associated with grounding, but like atomic processes are conceived of as having a single-step execution (simple processes are used as elements of abstraction on which to build more complicated processes),

- Composite — these are composed of multiple other (non-composite or composite) processes, with their decomposition being specified by using controlConstructs such as sequence and repeatUntil (decomposition will show how various inputs and outputs are accepted and returned by particular sub-processes).

This allows agents to monitor execution of a process. At the time of writing, the current version of OWL-S does not have a formal definition of the process control ontology, although a number of desirable features have been identified:

- Mapping rules for input properties to the corresponding output properties,
- A model of the dependencies described by the constructs,
- Representations for messages about the execution state of processes, allowing tracking of, and response to, executions.

The addition of Process descriptions as outlined above in OWL-S is an important step forward from today's Web Services technology, providing for a more precise definition of the pre-conditions and effects of a Web Service and allowing sets of services to be combined into a composite service using control constructs. However, as mentioned, OWL-S currently lacks a process control ontology and furthermore does not offer a formal semantics for states and state transitions.

The service getTTicketByKey is an example of an atomic service as it takes a number of inputs and returns a number of outputs. It maps directly to a WSDL description and can be invoked directly.

Inputs (e.g., TTicketID), outputs (e.g., StateOutput), preconditions and effects are described separately. For brevity, inputs, outputs, preconditions and effects will henceforth be referred to as "iopes."

```
<process:AtomicProcess rdf:ID="getTTicketByKey_Process">
  <process:hasInput rdf:resource="#TTicketID_In" />
  <process:hasOutput rdf:resource="#TTicketID_Out" />
  <process:hasOutput rdf:resource="#CustomerID_Out" />
  <process:hasOutput rdf:resource="#ServiceID_Out" />
  <process:hasOutput rdf:resource="#Description_Out" />
  <process:hasOutput rdf:resource="#Location_Out" />
  <process:hasOutput rdf:resource="#State_Out" />
  <process:hasOutput rdf:resource="#Status_Out" />
</process:AtomicProcess>
```

Associated with each process is a set of properties. Using a program or function metaphor, a process has parameters to which it is associated. Two types of parameters are the OWL-S properties *input* and (conditional) *output*.

An example of an input for getTTicketByKey is the Trouble Ticket ID. The following OWL fragment shows how this is defined. The input is declared as a subProperty of the general input property. It is related to an ontological concept TTicketID with the parameterType declaration.

```
<process:Input rdf:ID="TTicketID_In">
  <rdfs:subPropertyOf rdf:resource="&process;#input" />
  <process:parameterType rdf:resource="
        &getTTicketByKey_concepts;#TTicketID" />
</process:Input>
```

This allows the service model's inputs and outputs to be related to ontological concepts thus providing a frame of reference for the data requirements of the service.

Preconditions and conditional effects are described analogously to inputs and conditional outputs. Preconditions specify things that must be true of the world in order for an agent to execute a service. Unfortunately, there is no standard way to express preconditions within OWL-S although placeholders for these have been provided in the OWL-S ontology.

In order to specify preconditions for the atomic services in the case study it is first necessary to consider the case study scenario in terms of the states that can exist between receiving a service alarm and closing a trouble ticket. These states can be characterised by the things that must be true for that state to exist. In the scenario, these things are embodied by the existence of data in variables or the value of those variables. Naturally, the variables are exactly the input and output data that is consumed and produced by the atomic processes. Figure 24 shows the states within the case study scenario characterised by conditions. These conditions can be seen as postcondition of the preceding process and preconditions of the following process. For simplicity, the figure only shows the processes that cause a change in the state. Other processes such as viewing trouble tickets that do not change the state have been omitted. In addition, only those conditions that have changed with a state transition are shown. Obviously, in order to close a trouble ticket, the condition that customer data has been received must still hold but this is omitted.

In some situations the same WSDL operation can be used to ground multiple atomic processes. For example when informing the trouble ticket system that a job has been completed the WSDL operation updateTroubleTicket can be used with the status input set to 'CLEARED'. The same operation can be used to close the trouble ticket, this time with the status input set to 'CLOSED'. It is advantageous to use two different atomic processes here because it allows conditions to be added over the status input so that only those status values that are permitted at the current state are provided. For example, it would not make sense (at least in this scenario) to allow a ticket to be set to 'QUEUD' after it had been cleared. Where conditions are not provided, it is up to the designer to ensure that the correct input is provided to the generic WDSL operation. With Semantic Web Service it should be possible to specify the process to the extent that the designer no longer has the ability to get this wrong.

Considering the createTT process to create a trouble ticket, it has a precondition that the customer data is known and also that an alarm has been received. These two facts can be expressed as preconditions of the process as follows:

```
<process:AtomicProcess rdf:ID="createTT_Process">
  <process:hasPrecondition rdf:resource="#GotCustomerData" />
  <process:hasPrecondition rdf:resource="#AlarmReceived" />
</process:AtomicProcess>
```

The conditions are themselves defined as follows:

```
<owl:Class rdf:ID="GotCustomerData"/>
  <rdf:type rdf:resource="&process;#Condition" />
</owl:Class>

<owl:Class rdf:ID="AlarmReceived"/>
  <rdf:type rdf:resource="&process;#Condition" />
</owl:Class>
```

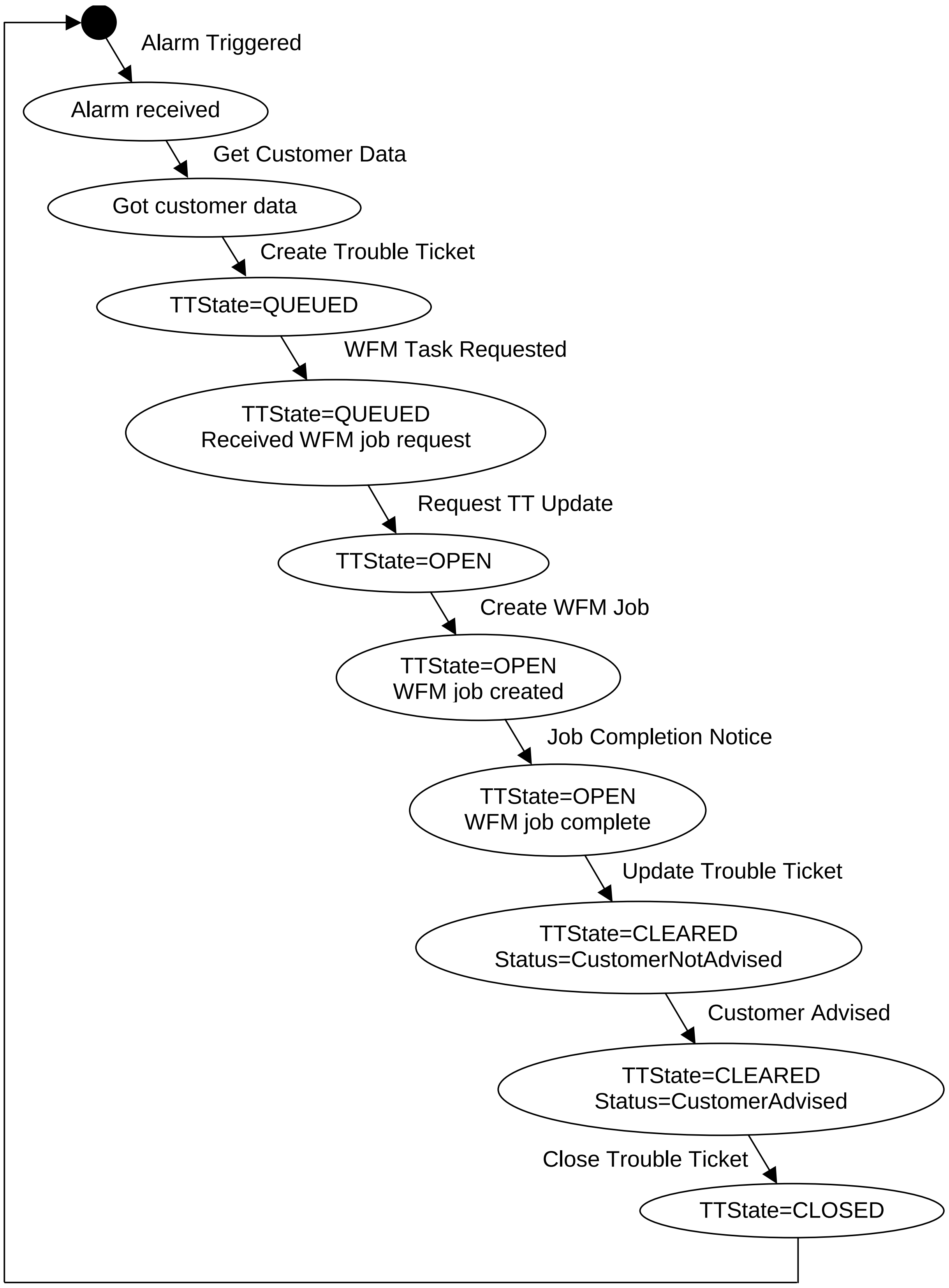


Figure 24. States with pre/post conditions.

Although these preconditions can be expressed, currently there is now standard way to evaluate them in OWL-S. However, the proposed Semantic Web Rule Language and other such initiatives should allow this.

Similarly, preconditions on the state of variables can be expressed. As mentioned above, in the scenario, the state of the trouble ticket can only be set to ‘CLOSED’ if the current status is ‘CLEARED’. The following represents this requirement:

```
<process:AtomicProcess rdf:ID="closeTT_Process">
  <process:hasPrecondition rdf:resource=
        "#updateTroubleTicketStatusOutput_State_Out_CLEARED"/>
</process:AtomicProcess>
```

This requires the output from the previous process (which included an output State_Out) to be set to the required value.

In addition to preconditions, OWL-S has the notion of effects. These are the things that are true once a process has completed. Figure 24 can also be used to find the appropriate effects or services that need to be represented. For example, the effect of updating the trouble ticket once a job complete notice has been received is that the TTState is set to ‘CLEARED’. That is of course if everything is correct with process e.g. that the trouble ticket ID sent is correct. The underlying WSDL operation contains an error flag that could be set if anything was wrong. Obviously, it would not be wise to set the TTState to ‘CLEARED’ under those circumstances. For this reason, the effects are conditional upon certain facts. In this case that the error flag is false. This can be modelled in OWL-S in the following way:

```
<process:AtomicProcess rdf:ID="updateTT_Process">
  <process:hasEffect>
    <process:ConditionalEffect>
      <process:ceCondition rdf:resource=
               "#updateTT_ProcessOutput_ErrorFlag_FALSE" />
      <process:ceEffect rdf:resource="#TT_State_CLEARED" />
     </process:ConditionalEffect>
  <process:hasEffect>
</process:AtomicProcess>
```

If an error has occurred then it might be appropriate to express a different effect which might lead to a different path being taken in the state diagram (although these have not been shown in figure 24).

The aim of the case study is to illustrate how a designer can compose services together to satisfy a high-level goal. The output of this activity will be a composed service. OWL-S allows atomic process to be composed together using a number of different constructs such as sequence, split-join, etc. The following example considers the first composition that is possible i.e. following an alarm, collect details from the inventory manager then create a trouble ticket. This is the end of the logical composition since the next event is dependent upon an administrator acting on the ticket which is an asynchronous event.

There are four possible states in this part of the process i.e. ‘start’, ‘alarm received’, ‘got customer data’ and ‘trouble ticket queued’. As stated above, the process is simplified in that it does not contain any error handling states. In the following example, if errors are received then there will be no state transition i.e. the process will return to the state that was current at the start of the attempted transition.

The following fragment shows the top level description of the composed process. The designer would name this and refer it to a service profile, allowing it to be advertised. The description also includes a pointer to the start state. All other states are encapsulated within the description of the start state so this is the only reference to the actual composition that is required.

```
<process:ProcessModel rdf:ID="handleAlarmWithTroubleTicket_Process">
  <service:describes rdf:resource=
                 "&service;#handleAlarmWithTroubleTicketService"/>
  <process:hasProcess rdf:resource="#StartState"/>
</process:ProcessModel>
```

The start state is described below as a composite process.

```
<process:CompositeProcess rdf:ID="StartState">
  <processComposedOf>
    <process:Sequence>
      <process:components rdf:parseType="Collection">
        <process:AtomicProcess rdf:about="#getAlarmResource"/>
        <process:CompositeProcess rdf:about="#AlarmReceivedState"/>
      </process:components>
    </process:Sequence>
  </processComposedOf>
</process:CompositeProcess>
```

The composition for this state is a simple sequence of two processes. The first is the atomic process ‘getAlarmResource’ which as described earlier takes the alarm as input and outputs the resource on which the alarm has occurred. The second process is the next state in the composition i.e. the ‘AlarmReceivedState’. This is another composite process described below.

```
<process:CompositeProcess rdf:ID="AlarmReceivedState">
  <processComposedOf>
    <process:Sequence>
      <process:components rdf:parseType="Collection">
        <process:AtomicProcess rdf:about="#getCustomerID"/>
        <process:If-Then-Else>
          <process:ifCondition rdf:resource=
               "#getCustomerIDOutput_ErrorFlag_FALSE"/>
          <process:then rdf:resource="#GotCustomerDataState"/>
          <process:else rdf:resource="#AlarmReceivedState"/>
        </process:If-Then-Else>
      </process:components>
    </process:Sequence>
  </processComposedOf>
</process:CompositeProcess>
```

This composite process includes a selection which determines the state transition based upon the output from the getCustomerID atomic process. In fact the description above has been simplified since a successful call to getServiceID is also required for a state transition. getSeviceID and getCustomerID can be run in parallel so the unordered control construct can be used. If neither process returns an error then the transition can be made. This requires a class to be constructed using <owl:unionOf> from the two error outputs. This examples also shows how control constructs can be nested.

```
<process:CompositeProcess rdf:ID="AlarmReceivedState">
  <processComposedOf>
    <process:Sequence>
      <process:components rdf:parseType="Collection">
        <process:Unordered>
          <process:components rdf:parseType="Collection">
            <process:AtomicProcess rdf:about="#getCustomerID"/>
            <process:AtomicProcess rdf:about="#getServiceID"/>
          </process:components>
        </process:Unordered>
        <process:If-Then-Else>
          <process:ifCondition rdf:resource=
               "#getCustomerIDOutput_ErrorFlag_FALSE_AND_
                 getCustomerIDOutput_ErrorFlag_FALSE "/>
          <process:then rdf:resource="#GotCustomerDataState"/>
          <process:else rdf:resource="#AlarmReceivedState"/>
        </process:If-Then-Else>
      </process:components>
    </process:Sequence>
  </processComposedOf>
</process:CompositeProcess>
```

The remaining two composite processes are shown below. These use the same constructs.

```
<process:CompositeProcess rdf:ID="GotCustomerDataState">
  <processComposedOf>
    <process:Sequence>
      <process:components rdf:parseType="Collection">
        <process:AtomicProcess rdf:about="#createTroubleTicket"/>
        <process:AtomicProcess rdf:about="#populateTroubleTicket"/>
        <process:AtomicProcess rdf:about="#storeLocalTT"/>
        <process:CompositeProcess rdf:about=
               "#TroubleTicketQueuedState"/>
      </process:components>
    </process:Sequence>
  </processComposedOf>
</process:CompositeProcess>
```

OWLS-S provides the <process:sameValues> construct to allow the data flow to be constructed. This allows the output from one atomic process to be aligned to an input from another. The following example shows one such construct linking the createTroubleTicket and the populateTroubleTicket processes via the troubleTicketID input / output.

```
<process:sameValues rdf:parseType="Collection">
  <process:ValueOf process:atClass="#createTroubleTicket"
   process:theProperty="#createTroubleTicketOutput_troubleTicketID"/>
  <process:ValueOf process:atClass="#populateTroubleTicket"
   process:theProperty="#populateTroubleTicketOutput_troubleTicketID"/>
</process:sameValues>
```

Although useful in this example, the construct is limited where more complex data flow is required such as when two outputs should be combined to form one input or where an output is a complex data type from which only a portion is required.

The output of the design process would be an OWL-S composed service along the lines of that described above. The composed service could then be advertised and discovered using its own process model without regard to the atomic processes that form it.

5.4 Service Profile

The service profile describes the service in terms of what it does. It is intended to advertise the capabilities of the service allowing it to be discovered. The profile has two major portions i.e. non-functional and functional descriptions. Non-functional descriptions cover a number of areas such as descriptions of the service provider, the quality rating of the service, etc. The most interesting non-functional description is classification of the service according to a domain ontology via the creation of a subclass of a class within it. This allows the services described in the case study to be classified according to the eTOM. The service profile exists as an instance of this class. The fragment below shows this for the getTTicketByKey service.

```
<owl:class rdf:ID="getTTicketByKey">
  <rdfs:subClassOf rdf:resource="&etom;#Track_and_Manage_Problem"/>
</owl:class>

<getTTicketByKey rdf:ID="getTTicketByKey_Profile"/>
```

This would allow the service to be discovered by a matchmaking process that used the eTOM ontology.

Functional properties describe the service in terms of their inputs, outputs, preconditions and effects (iopes). These are intended to aid discovery by allowing goal services to be described in these terms. There are no encoded logical constraints between the inputs in the process model and the inputs in the profile model, therefore, at least in theory, the two sets may be totally unrelated. This is a major deficiency of OWL-S since during match-making knowledge of how the iopes are used by the service would be of benefit.

6 Conclusion

This document describes the services and ontologies required and used by the Virtual Internet Service Provider. A storyboard for the case study is presented which is then considered in terms of its service and messaging requirements. A telecommunications industry wide initiative to provide information and process models is then described and its applicability as a domain ontology for the case study is considered. Finally, the component services required for the case study are described first as web services (using WSDL) and then as semantic web services using OWL-S.

This process has allowed a number of observations to be made regarding the domain ontology and the suitability of OWL-S to describe web services semantically.

The Web Ontology Language is in general flexible enough to capture the semantics of the TMF NGOSS models. Tool support for this process is poor. None of the major UML vendors support any Semantic Web languages. However, a research prototype is available that tackles some of the issues. This requires further development in key areas.

The semantics of UML and OWL differ. One of the key barriers to the adoption of the Semantic Web is likely to be a shortage of skills. Database modelers and information architects could help solve this problem but in order to utilize them efficiently, methodologies for creating ontologies and a clear understanding of the differences between the closed world model and the open world model are required.

The TMF NGOSS initiative will provide process and data models for the telecommunications industry. These are under development but it is clear that they are at a high level and require further modeling within a particular context e.g. a company of supply chain if they are to perform as a domain ontology for Semantic Web Services. There is currently a mismatch between these model and underlying service components. Having said this, the eTOM provides a useful process framework for categorising processes or service functions. The SID provides a useful starting point when constructing a canonical data dictionary and/or exchange model for a particular environment.

The coupling of OSS/J to web services promises a significant set of benefits for a telecom service provider, but the maturity of the underlying technologies are insufficient at this moment in time. Currently, there is no standardised mapping from OSS/J services to web services, which is crucial when inter working between two, or more, companies. Also, WSDL does not currently define a standardised, agreed way to describe and implement asynchronous services. Both of these features, however, are under development, and should be become available in the next few years.

OWL-S is an approach to allow the semantics of services to be expressed. It is the most concrete of the emerging initiatives in this area. OWL-S in its current form provides good support for mapping services and their data requirements (i.e. inputs and outputs) to ontological concepts. This can improve service discovery and promote a better understanding of the capabilities of a service within a wider domain. There are a number of outstanding issues with OWL-S that would require addressing if it is to achieve its stated aims. Firstly, support is required for expressing rules. This will allow the preconditions and effects of a service to be expressed and evaluated in a standardized way. Secondly, the OWL-S' process model is too simple. The minimal set of control structures provided does not have formally specified semantics and the support for complex data flow is poor. Thirdly it does not distinguish between public and private processes. Fourthly, it only supports grounding to WSDL web services. Finally it has little in the way of tool support.

Alternative approaches to OWL-S have been proposed but are in the early stages of development. The Web Services Modelling Framework and Ontology is of course one of these. This case study will monitor this as it emerges and attempt to apply it where appropriate.

References

1. TeleManagement Forum: NGOSS Overview Document. Available on the web at: http://www.tmforum.org/
2. TeleManagement Forum: Enhanced Telecom Operations Map® (eTOM) data sheet. Available on the web at: http://www.tmforum.org/
3. D. L. McGuinness & F. van Harmelen (eds): OWL Web Ontology Language Overview. Available on the web at: http://www.w3.org/TR/owl-features/
4. W3C, XML Schema, published on the web at http://www.w3.org/XML/Schema
5. The Protégé Ontology Editor and Knowledge Acquisition System. Available on the Web at: http://protege.stanford.edu/index.shtml
6. TeleManagement Forum: Shared Information/Data Model (SID). Available on the web at: http://www.tmforum.org/
7. Object Management Group: XML Metadata Interchange. Available on the web at: http://www.omg.org/technology/documents/formal/xmi.htm
8. K. Falkovych, M. Sabou, and H. Stuckenschmidt. UML for the Semantic Web: Transformation-Based Approaches. In B. Omelayenko and M. Klein, editors, Knowledge Transformation for the Semantic Web, pages 92--106. IOS Press, 2003.
9. O. Koryak. A Flexible Transformation System for UML to XML based knowledge representational languages. Msc. Thesis. Dept. of Artificial Intelligence, VU Amsterdam.
10. Sun Microsystems, Inc., OSS through Java Initiative Overview, http://java.sun.com/products/oss/overview.html,

Appendices

WDSL for TroubleTicket service

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:conv="http://www.openuri.org/2002/04/soap/conversation/"
xmlns:cw="http://www.openuri.org/2002/04/wsdl/conversation/" xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:jms="http://www.openuri.org/2002/04/wsdl/jms/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:s0="http://www.openuri.org/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
targetNamespace="http://www.openuri.org/">
    <types>
        <s:schema xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:ope="http://www.openuri.org/"
elementFormDefault="qualified" targetNamespace="http://www.openuri.org/">
            <s:element name="PopulateTTicket">
                <s:complexType>
                    <s:sequence>
                        <s:element name="TTicketID" type="s:string" minOccurs="0"/>
                        <s:element name="customerID" type="s:string" minOccurs="0"/>
                        <s:element name="serviceID" type="s:string" minOccurs="0"/>
                        <s:element name="Description" type="s:string" minOccurs="0"/>
                    </s:sequence>
                </s:complexType>
            </s:element>
            <s:element name="PopulateTTicketResponse">
                <s:complexType>
                    <s:sequence>
                        <s:element name="PopulateTTicketResult" type="ope:TTicket" minOccurs="0"/>
                    </s:sequence>
                </s:complexType>
            </s:element>
            <s:element name="TTicket" nillable="true" type="ope:TTicket"/>
            <s:element name="getTTicketByKey">
                <s:complexType>
                    <s:sequence>
                        <s:element name="TTicketID" type="s:string" minOccurs="0"/>
                    </s:sequence>
                </s:complexType>
            </s:element>
            <s:element name="getTTicketByKeyResponse">
                <s:complexType>
                    <s:sequence>
                        <s:element name="getTTicketByKeyResult" type="ope:TTicket" minOccurs="0"/>
                    </s:sequence>
                </s:complexType>
            </s:element>
            <s:element name="createTTicket">
                <s:complexType>
                    <s:sequence/>
                </s:complexType>
            </s:element>
            <s:element name="createTTicketResponse">
                <s:complexType>
                    <s:sequence>
                        <s:element name="createTTicketResult" type="s:string" minOccurs="0"/>
                    </s:sequence>
                </s:complexType>
            </s:element>
            <s:element name="string" nillable="true" type="s:string"/>
            <s:element name="updateTTicket">
                <s:complexType>
                    <s:sequence>
                        <s:element name="ticket" type="ope:TTicket" minOccurs="0"/>
                        <s:element name="State" type="s:int"/>
                        <s:element name="Status" type="s:int"/>
                    </s:sequence>
                </s:complexType>
            </s:element>
            <s:element name="updateTTicketResponse">
                <s:complexType>
                    <s:sequence>
                        <s:element name="updateTTicketResult" type="s:boolean"/>
                    </s:sequence>
                </s:complexType>
            </s:element>
            <s:complexType name="TTicket">
                <s:sequence>
                    <s:element name="TTicketID" type="s:string" minOccurs="0"/>
                    <s:element name="CustomerID" type="s:string" minOccurs="0"/>
                    <s:element name="ServiceID" type="s:string" minOccurs="0"/>
                    <s:element name="Description" type="s:string" minOccurs="0"/>
                    <s:element name="Location" type="s:string" minOccurs="0"/>
                    <s:element name="State" type="s:int"/>
                    <s:element name="Status" type="s:int"/>
                </s:sequence>
            </s:complexType>
        </s:schema>
    </types>
    <message name="PopulateTTicketSoapIn">
        <part name="parameters" element="s0:PopulateTTicket"/>
    </message>
    <message name="PopulateTTicketSoapOut">
        <part name="parameters" element="s0:PopulateTTicketResponse"/>
    </message>
    <message name="getTTicketByKeySoapIn">
        <part name="parameters" element="s0:getTTicketByKey"/>
    </message>
    <message name="getTTicketByKeySoapOut">
        <part name="parameters" element="s0:getTTicketByKeyResponse"/>
    </message>
    <message name="createTTicketSoapIn">
        <part name="parameters" element="s0:createTTicket"/>
    </message>
    <message name="createTTicketSoapOut">
        <part name="parameters" element="s0:createTTicketResponse"/>
    </message>
    <message name="updateTTicketSoapIn">
        <part name="parameters" element="s0:updateTTicket"/>
    </message>
    <message name="updateTTicketSoapOut">
        <part name="parameters" element="s0:updateTTicketResponse"/>
    </message>
    <message name="PopulateTTicketHttpGetIn">
        <part name="TTicketID" type="s:string"/>
        <part name="customerID" type="s:string"/>
        <part name="serviceID" type="s:string"/>
        <part name="Description" type="s:string"/>
    </message>
    <message name="PopulateTTicketHttpGetOut">
        <part name="Body" element="s0:TTicket"/>
    </message>
    <message name="getTTicketByKeyHttpGetIn">
        <part name="TTicketID" type="s:string"/>
    </message>
    <message name="getTTicketByKeyHttpGetOut">
        <part name="Body" element="s0:TTicket"/>
    </message>
    <message name="createTTicketHttpGetIn"/>
```

```
<message name="createTTicketHttpGetOut">
    <part name="Body" element="s0:string"/>
</message>
<message name="PopulateTTicketHttpPostIn">
    <part name="TTicketID" type="s:string"/>
    <part name="customerID" type="s:string"/>
    <part name="serviceID" type="s:string"/>
    <part name="Description" type="s:string"/>
</message>
<message name="PopulateTTicketHttpPostOut">
    <part name="Body" element="s0:TTicket"/>
</message>
<message name="getTTicketByKeyHttpPostIn">
    <part name="TTicketID" type="s:string"/>
</message>
<message name="getTTicketByKeyHttpPostOut">
    <part name="Body" element="s0:TTicket"/>
</message>
<message name="createTTicketHttpPostIn"/>
<message name="createTTicketHttpPostOut">
    <part name="Body" element="s0:string"/>
</message>
<portType name="TicketSoap">
    <operation name="PopulateTTicket">
        <input message="s0:PopulateTTicketSoapIn"/>
        <output message="s0:PopulateTTicketSoapOut"/>
    </operation>
    <operation name="getTTicketByKey">
        <input message="s0:getTTicketByKeySoapIn"/>
        <output message="s0:getTTicketByKeySoapOut"/>
    </operation>
    <operation name="createTTicket">
        <input message="s0:createTTicketSoapIn"/>
        <output message="s0:createTTicketSoapOut"/>
    </operation>
    <operation name="updateTTicket">
        <input message="s0:updateTTicketSoapIn"/>
        <output message="s0:updateTTicketSoapOut"/>
    </operation>
</portType>
<portType name="TicketHttpGet">
    <operation name="PopulateTTicket">
        <input message="s0:PopulateTTicketHttpGetIn"/>
        <output message="s0:PopulateTTicketHttpGetOut"/>
    </operation>
    <operation name="getTTicketByKey">
        <input message="s0:getTTicketByKeyHttpGetIn"/>
        <output message="s0:getTTicketByKeyHttpGetOut"/>
    </operation>
    <operation name="createTTicket">
        <input message="s0:createTTicketHttpGetIn"/>
        <output message="s0:createTTicketHttpGetOut"/>
    </operation>
</portType>
<portType name="TicketHttpPost">
    <operation name="PopulateTTicket">
        <input message="s0:PopulateTTicketHttpPostIn"/>
        <output message="s0:PopulateTTicketHttpPostOut"/>
    </operation>
    <operation name="getTTicketByKey">
        <input message="s0:getTTicketByKeyHttpPostIn"/>
        <output message="s0:getTTicketByKeyHttpPostOut"/>
    </operation>
    <operation name="createTTicket">
```

```
            <input message="s0:createTTicketHttpPostIn"/>
            <output message="s0:createTTicketHttpPostOut"/>
        </operation>
    </portType>
    <binding name="TicketSoap" type="s0:TicketSoap">
        <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="PopulateTTicket">
            <soap:operation soapAction="http://www.openuri.org/PopulateTTicket" style="document"/>
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
        </operation>
        <operation name="getTTicketByKey">
            <soap:operation soapAction="http://www.openuri.org/getTTicketByKey" style="document"/>
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
        </operation>
        <operation name="createTTicket">
            <soap:operation soapAction="http://www.openuri.org/createTTicket" style="document"/>
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
        </operation>
        <operation name="updateTTicket">
            <soap:operation soapAction="http://www.openuri.org/updateTTicket" style="document"/>
            <input>
                <soap:body use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
        </operation>
    </binding>
    <binding name="TicketHttpGet" type="s0:TicketHttpGet">
        <http:binding verb="GET"/>
        <operation name="PopulateTTicket">
            <http:operation location="/PopulateTTicket"/>
            <input>
                <http:urlEncoded/>
            </input>
            <output>
                <mime:mimeXml part="Body"/>
            </output>
        </operation>
        <operation name="getTTicketByKey">
            <http:operation location="/getTTicketByKey"/>
            <input>
                <http:urlEncoded/>
            </input>
            <output>
                <mime:mimeXml part="Body"/>
            </output>
        </operation>
        <operation name="createTTicket">
```

```
            <http:operation location="/createTTicket"/>
            <input>
                <http:urlEncoded/>
            </input>
            <output>
                <mime:mimeXml part="Body"/>
            </output>
        </operation>
    </binding>
    <binding name="TicketHttpPost" type="s0:TicketHttpPost">
        <http:binding verb="POST"/>
        <operation name="PopulateTTicket">
            <http:operation location="/PopulateTTicket"/>
            <input>
                <mime:content type="application/x-www-form-urlencoded"/>
            </input>
            <output>
                <mime:mimeXml part="Body"/>
            </output>
        </operation>
        <operation name="getTTicketByKey">
            <http:operation location="/getTTicketByKey"/>
            <input>
                <mime:content type="application/x-www-form-urlencoded"/>
            </input>
            <output>
                <mime:mimeXml part="Body"/>
            </output>
        </operation>
        <operation name="createTTicket">
            <http:operation location="/createTTicket"/>
            <input>
                <mime:content type="application/x-www-form-urlencoded"/>
            </input>
            <output>
                <mime:mimeXml part="Body"/>
            </output>
        </operation>
    </binding>
    <service name="Ticket">
        <port name="TicketSoap" binding="s0:TicketSoap">
            <soap:address location="http://BTG022823:7001/TTWSWeb/Ticket.jws"/>
        </port>
        <port name="TicketHttpGet" binding="s0:TicketHttpGet">
            <http:address location="http://BTG022823:7001/TTWSWeb/Ticket.jws"/>
        </port>
        <port name="TicketHttpPost" binding="s0:TicketHttpPost">
            <http:address location="http://BTG022823:7001/TTWSWeb/Ticket.jws"/>
        </port>
    </service>
</definitions>
```

Service Grounding for getTroubleTicketBykey

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE uridef [
      <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns">
      <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema">
      <!ENTITY owl "http://www.w3.org/2002/07/owl">
      <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
      <!ENTITY service "http://localhost:8080/esrOntologies/OWL-S_1.0/Service.owl">
      <!ENTITY grounding "http://localhost:8080/esrOntologies/OWL-S_1.0/Grounding.owl">
      <!ENTITY getTTicketByKey_service "http://localhost:8080/esrExampleServices/getTTicketByKey/getTTicketByKey-
Service.owl">
      <!ENTITY getTTicketByKey_process "http://localhost:8080/esrExampleServices/getTTicketByKey/getTTicketByKey-
Process.owl">
      <!ENTITY getTTicketByKeyWSDL "http://localhost:8080/esrExampleServices/getTTicketByKey/Ticket.jws">
      <!ENTITY DEFAULT "http://localhost:8080/esrExampleServices/getTTicketByKey/getTTicketByKey-
Grounding.owl">
]>
<!--
This document uses entity types as a shorthand for URIs.
Download the source for a version with unexpanded entities.
  -->
<rdf:RDF xmlns:rdf="&rdf;#" xmlns:rdfs="&rdfs;#" xmlns:owl="&owl;#" xmlns:xsd="&xsd;#" xmlns:service="&service;#"
xmlns:grounding="&grounding;#" xmlns="&DEFAULT;#">
      <owl:Ontology rdf:about="">
            <owl:versionInfo>
      $Id: getTTicketByKey-Grounding.owl, v 1.0 2004/Feb/25 15:10:14 darko Exp $
    </owl:versionInfo>
            <rdfs:comment>
      This ontology represents the OWL-S grounding description for the
      getTTicketByKey web service.
    </rdfs:comment>
            <owl:imports rdf:resource="&service;"/>
            <owl:imports rdf:resource="&grounding;"/>
            <owl:imports rdf:resource="&getTTicketByKey_service;"/>
            <owl:imports rdf:resource="&getTTicketByKey_process;"/>
            <owl:imports rdf:resource="&getTTicketByKeyWSDL;"/>
      </owl:Ontology>
      <!-- ############################################################ -->
      <!-- # Instance Definition of getTTicketByKey Grounding           # -->
      <!-- ############################################################ -->
      <!-- ############################################################ -->
      <!-- # service:ServiceGrounding                                   # -->
      <grounding:WsdlGrounding rdf:ID="getTTicketByKey_Grounding">
            <service:supportedBy rdf:resource="&getTTicketByKey_service;#getTTicketByKey"/>
            <!-- Collecton of all the groundings specifications -->
            <grounding:hasAtomicProcessGrounding rdf:resource="#WsdlGrounding_getTTicketByKey"/>
      </grounding:WsdlGrounding>
      <!-- ############################################################ -->
      <!-- #  getTTicketByKey                                          # -->
      <grounding:WsdlAtomicProcessGrounding rdf:ID="WsdlGrounding_getTTicketByKey">
            <!-- Grounding for the Atomic Process getTTicketByKey -->
            <grounding:owlsProcess rdf:resource="&getTTicketByKey_process;#getTTicketByKey_Process"/>
            <!-- Reference to the corresponding WSDL operation -->
            <grounding:wsdlOperation rdf:resource="#getTTicketByKey_opeartion"/>
            <!-- Reference to the WSDL input message -->
            <grounding:wsdlInputMessage>
                  <xsd:anyURI rdf:value="&getTTicketByKeyWSDL;#getTTicketByKeyHttpPostIn"/>
            </grounding:wsdlInputMessage>
            <!-- Mapping of OWL-S inputs to WSDL message parts -->
            <grounding:wsdlInputs rdf:parseType="Collection">
                  <grounding:WsdlInputMessageMap>
                        <grounding:owlsParameter rdf:resource="&getTTicketByKey_process;#TTicketID_In"/>
                        <grounding:wsdlMessagePart>
```

```
                    <xsd:anyURI rdf:value="&getTTicketByKeyWSDL;#TTicketID"/>
                </grounding:wsdlMessagePart>
            </grounding:WsdlInputMessageMap>
        </grounding:wsdlInputs>
        <!-- Reference to the WSDL output message -->
        <grounding:wsdlOutputMessage>
            <xsd:anyURI rdf:value="&getTTicketByKeyWSDL;#getTTicketByKeyHttpPostOut"/>
        </grounding:wsdlOutputMessage>
        <!-- Mapping of OWL-S outputs to WSDL message parts -->
        <grounding:wsdlOutputs rdf:parseType="Collection">
            <grounding:WsdlOutputMessageMap>
                <grounding:owlsParameter rdf:resource="&getTTicketByKey_process;#TTicketID_Out"/>
                <grounding:wsdlMessagePart>
                    <xsd:anyURI rdf:value="&getTTicketByKeyWSDL;#TTicketID"/>
                </grounding:wsdlMessagePart>
            </grounding:WsdlOutputMessageMap>
            <grounding:WsdlOutputMessageMap>
                <grounding:owlsParameter rdf:resource="&getTTicketByKey_process;#CustomerID_Out"/>
                <grounding:wsdlMessagePart>
                    <xsd:anyURI rdf:value="&getTTicketByKeyWSDL;#CustomerID"/>
                </grounding:wsdlMessagePart>
            </grounding:WsdlOutputMessageMap>
            <grounding:WsdlOutputMessageMap>
                <grounding:owlsParameter rdf:resource="&getTTicketByKey_process;#ServiceID_Out"/>
                <grounding:wsdlMessagePart>
                    <xsd:anyURI rdf:value="&getTTicketByKeyWSDL;#ServiceID"/>
                </grounding:wsdlMessagePart>
            </grounding:WsdlOutputMessageMap>
            <grounding:WsdlOutputMessageMap>
                <grounding:owlsParameter rdf:resource="&getTTicketByKey_process;#Description_Out"/>
                <grounding:wsdlMessagePart>
                    <xsd:anyURI rdf:value="&getTTicketByKeyWSDL;#Description"/>
                </grounding:wsdlMessagePart>
            </grounding:WsdlOutputMessageMap>
            <grounding:WsdlOutputMessageMap>
                <grounding:owlsParameter rdf:resource="&getTTicketByKey_process;#Location_Out"/>
                <grounding:wsdlMessagePart>
                    <xsd:anyURI rdf:value="&getTTicketByKeyWSDL;#Location"/>
                </grounding:wsdlMessagePart>
            </grounding:WsdlOutputMessageMap>
            <grounding:WsdlOutputMessageMap>
                <grounding:owlsParameter rdf:resource="&getTTicketByKey_process;#State_Out"/>
                <grounding:wsdlMessagePart>
                    <xsd:anyURI rdf:value="&getTTicketByKeyWSDL;#State"/>
                </grounding:wsdlMessagePart>
            </grounding:WsdlOutputMessageMap>
            <grounding:WsdlOutputMessageMap>
                <grounding:owlsParameter rdf:resource="&getTTicketByKey_process;#Status_Out"/>
                <grounding:wsdlMessagePart>
                    <xsd:anyURI rdf:value="&getTTicketByKeyWSDL;#Status"/>
                </grounding:wsdlMessagePart>
            </grounding:WsdlOutputMessageMap>
        </grounding:wsdlOutputs>
        <grounding:wsdlReference>
            <xsd:anyURI rdf:value="http://www.w3.org/TR/2001/NOTE-wsdl-20010315"/>
        </grounding:wsdlReference>
    </grounding:WsdlAtomicProcessGrounding>
    <grounding:WsdlOperationRef rdf:ID="getTTicketByKey_operation">
        <rdfs:comment>
        getTTicketByKey_operation......
  </rdfs:comment>
        <!-- locate port type to be used -->
        <grounding:portType>
            <xsd:anyURI rdf:value="&getTTicketByKeyWSDL;#TicketSoap"/>
        </grounding:portType>
        <!-- locate operation to be used -->
        <grounding:operation>
            <xsd:anyURI rdf:value="&getTTicketByKeyWSDL;#getTTicketByKey"/>
        </grounding:operation>
    </grounding:WsdlOperationRef>
</rdf:RDF>
```

Service Model for getTroubleTicketBykey

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE uridef [
      <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns">
      <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema">
      <!ENTITY owl "http://www.w3.org/2002/07/owl">
      <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
      <!ENTITY service "http://localhost:8080/esrOntologies/OWL-S_1.0/Service.owl">
      <!ENTITY profile "http://localhost:8080/esrOntologies/OWL-S_1.0/Profile.owl">
      <!ENTITY process "http://localhost:8080/esrOntologies/OWL-S_1.0/Process.owl">
      <!ENTITY getTTicketByKey_concepts
"http://localhost:8080/esrExampleServices/getTTicketByKey/getTTicketByKey-Concepts.owl">
      <!ENTITY getTTicketByKey_service "http://localhost:8080/esrExampleServices/getTTicketByKey/getTTicketByKey-
Service.owl">
      <!ENTITY DEFAULT "http://localhost:8080/esrExampleServices/getTTicketByKey/getTTicketByKey-Process.owl">
]>
<rdf:RDF xmlns:rdf="&rdf;#" xmlns:rdfs="&rdfs;#" xmlns:owl="&owl;#" xmlns:xsd="&xsd;#" xmlns:service="&service;#"
xmlns:process="&process;#" xmlns:profile="&profile;#" xmlns="&DEFAULT;#">
      <owl:Ontology rdf:about="">
            <owl:versionInfo>
       $Id: getTTicketByKey-Process.owl, v 1.0 2004/Feb/25 15:10:14 darko Exp $
    </owl:versionInfo>
            <rdfs:comment>
      This ontology represents the OWL-S process description for the
      getTTicketByKey web service.

    </rdfs:comment>
            <owl:imports rdf:resource="&service;"/>
            <owl:imports rdf:resource="&process;"/>
            <owl:imports rdf:resource="&profile;"/>
            <owl:imports rdf:resource="&getTTicketByKey_concepts;"/>
      </owl:Ontology>
      <!-- ############################################################################ -->
      <!-- Instance Definition of getTTicketByKey Process Model -->
      <process:ProcessModel rdf:ID="getTTicketByKey_ProcessModel">
            <process:hasProcess rdf:resource="#getTTicketByKey_Process"/>
            <service:describes rdf:resource="&getTTicketByKey_service;#getTTicketByKey"/>
      </process:ProcessModel>
      <!--
getTTicketByKey is not a composite process.

If getTTicketByKey is composed of a sequence whose components atomic
processes, thay sholud be defined here.

-->
      <!-- Conditional outputs shoud be defined here -->
      <!-- #################################################################################### -->
      <!-- getTTicketByKey (ATOMIC)
     Get details....
  -->
      <process:AtomicProcess rdf:ID="getTTicketByKey_Process">
            <process:hasInput rdf:resource="#TTicketID_In"/>
            <process:hasOutput rdf:resource="#TTicketID_Out"/>
            <process:hasOutput rdf:resource="#CustomerID_Out"/>
            <process:hasOutput rdf:resource="#ServiceID_Out"/>
            <process:hasOutput rdf:resource="#Description_Out"/>
            <process:hasOutput rdf:resource="#Location_Out"/>
            <process:hasOutput rdf:resource="#State_Out"/>
            <process:hasOutput rdf:resource="#Status_Out"/>
      </process:AtomicProcess>
      <process:Input rdf:ID="TTicketID_In">
            <rdfs:subPropertyOf rdf:resource="&process;#input"/>
```

```
        <process:parameterType rdf:resource="&getTTicketByKey_concepts;#tTicketID"/>
    </process:Input>
    <process:Output rdf:ID="TTicketID_Out">
        <rdfs:subPropertyOf rdf:resource="&process;#output"/>
        <process:parameterType rdf:resource="&getTTicketByKey_concepts;#tTicketID"/>
    </process:Output>
    <process:Output rdf:ID="CustomerID_Out">
        <rdfs:subPropertyOf rdf:resource="&process;#output"/>
        <process:parameterType rdf:resource="&getTTicketByKey_concepts;#customerID"/>
    </process:Output>
    <process:Output rdf:ID="ServiceID_Out">
        <rdfs:subPropertyOf rdf:resource="&process;#output"/>
        <process:parameterType rdf:resource="&getTTicketByKey_concepts;#serviceID"/>
    </process:Output>
    <process:Output rdf:ID="Description_Out">
        <rdfs:subPropertyOf rdf:resource="&process;#output"/>
        <process:parameterType rdf:resource="&getTTicketByKey_concepts;#description"/>
    </process:Output>
    <process:Output rdf:ID="Location_Out">
        <rdfs:subPropertyOf rdf:resource="&process;#output"/>
        <process:parameterType rdf:resource="&getTTicketByKey_concepts;#location"/>
    </process:Output>
    <process:Output rdf:ID="State_Out">
        <rdfs:subPropertyOf rdf:resource="&process;#output"/>
        <process:parameterType rdf:resource="&getTTicketByKey_concepts;#state"/>
    </process:Output>
    <process:Output rdf:ID="Status_Out">
        <rdfs:subPropertyOf rdf:resource="&process;#output"/>
        <process:parameterType rdf:resource="&getTTicketByKey_concepts;#status"/>
    </process:Output>
</rdf:RDF>
```

Sevice Profile for getTroubleTicketBykey

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE uridef [
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns">
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema">
    <!ENTITY owl "http://www.w3.org/2002/07/owl">
    <!ENTITY service "http://localhost:8080/esrOntologies/OWL-S_1.0/Service.owl">
    <!ENTITY profile "http://localhost:8080/esrOntologies/OWL-S_1.0/Profile.owl">
    <!ENTITY profileHierarchy "http://localhost:8080/esrOntologies/OWL-S_1.0/ProfileHierarchy.owl">
    <!ENTITY process "http://localhost:8080/esrOntologies/OWL-S_1.0/Process.owl">
    <!ENTITY getTTicketByKey_concepts
"http://localhost:8080/esrExampleServices/getTTicketByKey/getTTicketByKey-Concepts.owl">
    <!ENTITY getTTicketByKey_service "http://localhost:8080/esrExampleServices/getTTicketByKey/getTTicketByKey-
Service.owl">
    <!ENTITY getTTicketByKey_process "http://localhost:8080/esrExampleServices/getTTicketByKey/getTTicketByKey-
Process.owl">
    <!ENTITY getTTicketByKey_grounding
"http://localhost:8080/esrExampleServices/getTTicketByKey/getTTicketByKey-Grounding.owl">
    <!ENTITY DEFAULT "http://localhost:8080/esrExampleServices/getTTicketByKey/getTTicketByKey-Profile.owl">
    <!ENTITY etom "http://localhost:8080/esrBrowser/eTOM.owl">
]>
<!--
This document uses entity types as a shorthand for URIs.
Download the source for a version with unexpanded entities.
  -->
<rdf:RDF xmlns:rdf="&rdf;#" xmlns:rdfs="&rdfs;#" xmlns:owl="&owl;#" xmlns:service="&service;#"
xmlns:process="&process;#" xmlns:profile="&profile;#" xmlns:etom="&etom;#"
xmlns:profileHierarchy="&profileHierarchy;#" xmlns="&DEFAULT;#">
    <owl:Ontology rdf:about="">
        <owl:versionInfo>
    $Id: getTTicketByKey-Profile.owl, v 1.0 2004/Feb/25 15:10:14 darko Exp $
  </owl:versionInfo>
        <rdfs:comment>
    This ontology represents the OWL-S profile description for the
    getTTicketByKey web service.
  </rdfs:comment>
        <owl:imports rdf:resource="&service;"/>
        <owl:imports rdf:resource="&profile;"/>
        <owl:imports rdf:resource="&process;"/>
        <owl:imports rdf:resource="&profileHierarchy;"/>
        <owl:imports rdf:resource="&getTTicketByKey_concepts;"/>
        <owl:imports rdf:resource="&getTTicketByKey_service;"/>
        <owl:imports rdf:resource="&getTTicketByKey_process;"/>
        <owl:imports rdf:resource="&getTTicketByKey_grounding;"/>
    </owl:Ontology>
    <!-- define template class getTTicketByKey that subclasses Track_and_Manage_Problem in the eTom Ontology -->
    <owl:class rdf:ID="getTTicketByKey">
        <rdfs:subClassOf rdf:resource="&etom;#Track_and_Manage_Problem"/>
    </owl:class>
    <!-- ###################################################################### -->
    <!-- # Instance Definition of getTTicketByKey Agent                     # -->
    <!-- ###################################################################### -->
    <getTTicketByKey rdf:ID="getTTicketByKey_Profile">
        <!-- reference to the service specification -->
        <service:presentedBy rdf:resource="&getTTicketByKey_service;#getTTicketByKey"/>
        <!-- reference to the process model specification -->
        <profile:has_process rdf:resource="&getTTicketByKey_process;#getTTicketByKey_Process"/>
        <profile:serviceName>getTTicketByKey</profile:serviceName>
        <profile:textDescription>
        Returns Trouble Ticket information for a specific Trouble Ticket key
  </profile:textDescription>
        <!--
     specification of contact information.
```

```
            There are two contacts specified here:
            1. to a company
            2. to pearson that is in charge...

            The two conctacs are related to the profile through different
            instances of the contactInfo relation

  -->
    <profile:contactInformation>
        <profile:Actor rdf:ID="BTexact-Next Generation Web Resreach">
            <profile:name>BTexact Technologies</profile:name>
            <profile:title>Resreach Representative</profile:title>
            <profile:phone>412 268 8789 </profile:phone>
            <profile:fax>412 268 5569 </profile:fax>
            <profile:email>http://www.bt.com/index.jsp</profile:email>
            <profile:physicalAddress>
         Adastral Park
            Martlesham
            Ipswich
            IP5 3RD
            UK
    </profile:physicalAddress>
            <profile:webURL>
       http://132.146.233.36:7001/TTWSWeb/Ticket.jws?.EXPLORE=.TEST
   </profile:webURL>
        </profile:Actor>
    </profile:contactInformation>
    <profile:contactInformation>
        <profile:Actor rdf:ID="BTexact-Next Generation Web Resreach">
            <profile:name>Nick Kings</profile:name>
            <profile:title>Knowledge Communities Specialist</profile:title>
            <profile:phone>412 268 8789 </profile:phone>
            <profile:fax>412 268 5569 </profile:fax>
            <profile:email>nick.kings@bt.com</profile:email>
            <profile:physicalAddress>
         Adastral Park
            Martlesham
            Ipswich
            IP5 3RD
            UK
    </profile:physicalAddress>
            <profile:webURL>
       http://132.146.233.36:7001/TTWSWeb/Ticket.jws?.EXPLORE=.TEST
   </profile:webURL>
        </profile:Actor>
    </profile:contactInformation>
    <!-- description of Geographic radius as a service parameter.
rather than a direct property of profile as in version 0.6
-->
    <profile:serviceParameter>
        <profile:GeographicRadius rdf:ID="getTTicketByKey-geographicRadius">
            <profile:serviceParameterName>
 getTTicketByKey Geographic Radius
</profile:serviceParameterName>
            <profile:sParameter rdf:resource="&getTTicketByKey_concepts;#UNITED-KINGDOM"/>
        </profile:GeographicRadius>
    </profile:serviceParameter>
    <!-- specification of quality rating for profile -->
    <profile:qualityRating>
        <profile:QualityRating rdf:ID="getTTicketByKey-goodRating">
            <profile:ratingName>
 SomeRating
</profile:ratingName>
            <profile:rating rdf:resource="&getTTicketByKey_concepts;#GoodRating"/>
        </profile:QualityRating>
    </profile:qualityRating>
    <!-- Specification of the service category using NAICS -->
    <profile:serviceCategory>
        <profile:NAICS rdf:ID="NAICS-category">
            <profile:value>
  ?????
</profile:value>
            <profile:code>
 561599
</profile:code>
        </profile:NAICS>
    </profile:serviceCategory>
    <!-- Specification of the service category using UN-SPSC -->
    <profile:serviceCategory>
        <profile:UNSPSC rdf:ID="UNSPSC-category">
            <profile:value>
 Travel Agent   ????
</profile:value>
            <profile:code>
 90121500
</profile:code>
        </profile:UNSPSC>
    </profile:serviceCategory>
    <!-- Descriptions of IOPEs -->
    <!-- Descriptions of the parameters that will be used by IOPEs -->
    <profile:input>
        <profile:ParameterDescription rdf:ID="TTicketID">
            <profile:parameterName>TTicketID</profile:parameterName>
            <profile:restrictedTo rdf:resource="&getTTicketByKey_concepts;#tTicketID"/>
            <profile:refersTo rdf:resource="&getTTicketByKey_process;#TTicketID_In"/>
        </profile:ParameterDescription>
    </profile:input>
    <profile:output>
        <profile:ParameterDescription rdf:ID="TTicketID">
            <profile:parameterName>TTicketID</profile:parameterName>
            <profile:restrictedTo rdf:resource="&getTTicketByKey_concepts;#tTicketID"/>
            <profile:refersTo rdf:resource="&getTTicketByKey_process;#TTicketID_Out"/>
        </profile:ParameterDescription>
    </profile:output>
    <profile:output>
        <profile:ParameterDescription rdf:ID="CustomerID">
            <profile:parameterName>CustomerID</profile:parameterName>
            <profile:restrictedTo rdf:resource="&getTTicketByKey_concepts;#customerID"/>
            <profile:refersTo rdf:resource="&getTTicketByKey_process;#CustomerID_Out"/>
        </profile:ParameterDescription>
    </profile:output>
    <profile:output>
        <profile:ParameterDescription rdf:ID="ServiceID">
            <profile:parameterName>ServiceID</profile:parameterName>
            <profile:restrictedTo rdf:resource="&getTTicketByKey_concepts;#serviceID"/>
            <profile:refersTo rdf:resource="&getTTicketByKey_process;#ServiceID_Out"/>
        </profile:ParameterDescription>
    </profile:output>
    <profile:output>
        <profile:ParameterDescription rdf:ID="Description">
            <profile:parameterName>Description</profile:parameterName>
            <profile:restrictedTo rdf:resource="&getTTicketByKey_concepts;#description"/>
            <profile:refersTo rdf:resource="&getTTicketByKey_process;#Description_Out"/>
        </profile:ParameterDescription>
    </profile:output>
    <profile:output>
        <profile:ParameterDescription rdf:ID="Location">
            <profile:parameterName>Location</profile:parameterName>
```

```
                <profile:restrictedTo rdf:resource="&getTTicketByKey_concepts;#location"/>
                <profile:refersTo rdf:resource="&getTTicketByKey_process;#Location_Out"/>
            </profile:ParameterDescription>
        </profile:output>
        <profile:output>
            <profile:ParameterDescription rdf:ID="State">
                <profile:parameterName>State</profile:parameterName>
                <profile:restrictedTo rdf:resource="&getTTicketByKey_concepts;#state"/>
                <profile:refersTo rdf:resource="&getTTicketByKey_process;#State_Out"/>
            </profile:ParameterDescription>
        </profile:output>
        <profile:output>
            <profile:ParameterDescription rdf:ID="Status">
                <profile:parameterName>Status</profile:parameterName>
                <profile:restrictedTo rdf:resource="&getTTicketByKey_concepts;#status"/>
                <profile:refersTo rdf:resource="&getTTicketByKey_process;#Status_Out"/>
            </profile:ParameterDescription>
        </profile:output>
        <!-- The consequence of getTTicketByKey is that......
        <profile:effect>
            <profile:ParameterDescription rdf:ID="getTTicketByKeyEffect">
                <profile:parameterName>PopulateTTicketEffect</profile:parameterName>
                <profile:restrictedTo rdf:resource="&getTTicketByKey_concepts;#GetTTicketByKeyEffect"/>
                <profile:refersTo rdf:resource="&getTTicketByKey_process;#getTTicketByKey_Effect"/>
            </profile:ParameterDescription>
        </profile:effect>

        -->
    </getTTicketByKey>
</rdf:RDF>
```