MDA-Driven Development of standard-compliant OSS components: the OSS/J Inventory Case-Study

Nektarios Georgalas[1]*, Manooch Azmoodeh**
BT Group, UK
Tony Clark***, Andy Evans***, Paul Sammut***, James Willans***
Xactium Ltd, UK

*georgan@acm.org

*manooch.azmoodeh@bt.com

***[firstName.surname]@xactium.com

Abstract: *The telecommunications-oriented Operational Support Systems (OSS) industry have recognised the value of technology independent modelling of OSS solutions as a way to reduce cost, add agility, validate and verify solution designs against architectural guidelines of an enterprise and most importantly provide traceability in the design methodology process. The challenges faced by the OSS community is how MDA tools can deliver the promise of advanced meta-modelling, model definition and validation and model transformation for both OSS software components and integration logic in the larger OSS landscape. This paper describes how an advanced extensible meta-modelling tool is used to build an OSS component following best practice industry guidelines. Extended MOF, extended executable OCL and a powerful transformation language are used to capture the constraints in the meta-models as well as models followed by complete, 100% code generation from models. Furthermore, meta-models are also developed to capture graphical user interface elements in conjunction with the inventory data models, which are then automatically translated into code. This work is the precursor for defining extensive meta-models for a component-based OSS infrastructure based on industry best practice, for adding high degree of formality to model specifications and for enabling the verification of domain requirements by executing the models through model snapshot creation, way before system implementation takes place.*

Keywords: *OSS, OSS/J, NGOSS, TMF, Component, Contract, OMG MDA, TNA,TSA, meta-modelling executable OCL, Inventory System, IP VPN*

1 Introduction

Developing and operating Operational Support Systems (OSS) for telecommunications companies (telcos) is a very expensive process whose const continuously grows year on year. With the introduction of new products and services, telcos are constantly challenged to reduce the overall costs and improve business agility in terms of faster time-to-market for new services and products. It is recognised that the major proportion of overall costs is in integration and maintenance of OSS solutions. Currently, the OSS infrastructure of a typical telco comprises an order of O(1000) systems all with point-to-point interconnections and using diverse platforms and implementation technologies. The telcoms OSS industry has already established the basic principles for building and operating OSS through the TMF NGOSS programme **[NGOSS]** and the OSS through Java initiative **[OSSJ]**. In summary, the NGOSS applies a top-level approach through the specification of an OSS architecture where:

- Technology Neutral and Technology Specific Architectures are separated
- The more dynamic "business process" logic is separated from the more stable "component" logic

[1] Address: Adastral Park, Orion Building – Ground Floor pp13, Martlesham Heath, Ipswich, IP5 3RE, UK

- Components present their services through well defined “contracts”
- “Policies” are used to provide a flexible control of behaviour in an overall NGOSS system
- The infrastructure services such as naming, invocation, directories, transactions, security, persistence, etc are provided as a common deployment and runtime framework for common use by all OSS components and business processes over a service bus.

Complementary to NGOSS is the low-level approach of the OSS/J, which provides a set of standard Java-based interface specifications with a roadmap for producing APIs covering the entire landscape of the OSS space (Trouble Ticketing, QoS monitoring, Inventory, Billing, SLA management, etc.). These specifications provide a technology specific set of OSS functional capabilities and as such can be a basis for building an implementation view of an OSS using J2EE platform.

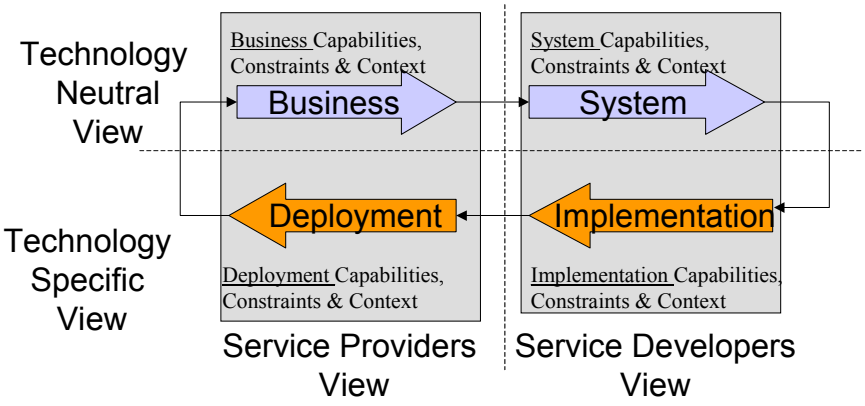

Figure 1 - NGOSS lifecycle methodology

In addition to the above, NGOSS has defined a methodology for developing OSS solutions emphasising separation of concerns so that different actors in the overall design process are freed from polluting their models with details and aspects of other areas. Figure 1 shows four different lifecycle stages or views identified by the NGOSS methodology, namely, business, system, implementation and deployment view. The top row shows the logical views of the systems, which are ***technology neutral***. The business view captures business contracts, business processes, entities and interactions (using the eTOM **[eTOM]** and SID **[SID]** standards) without reference to how they are realised using automated computer systems. The System view provides the computational interactions among automated components, processes and policies. The bottom row shows the physical view of system, which are intrinsically ***technology specific***, where on the right, the implementation view contains the hardware and software to construct the system and on the left the deployment view captures the instance level operating systems and active monitoring of the system.

One of the main goals of the NGOSS lifecycle is to provide traceability from business requirements to systems descriptions to implementation details and finally to deployed systems, thus traversing from the top-level, technology neutral NGOSS specifications to the low-level, Java-based APIs and J2EE architectural principles of

OSS/J. The MDA technology **[MDA]** is identified as the key enabler for providing such automated traceability in an NGOSS environment. The main goal of our research is to use MDA standards and tools to define meta-models and transformation rules around the lifecycle so that various models and views in the lifecycle can be verified for correctness and completeness as well as auto-generating these models and systems. First research results presented in **[Georgalas 2004]** focused upon the applicability of an MDA enabled OSS architecture in a telcoms environment and a generic technology independent NGOSS component specification.

In this paper, we describe how an advanced meta-modelling toolkit, the XMF from XACTIUM **[XACTIUM]** in particular, can be practically used to specify and automatically generate a complete system implementation of a single OSS component. As an example, the component at issue is based on the OSS/J Inventory API specification and can manage models as well as instances of products, services and resources within a telco environment. The paper, also, shows the use of a constraint language, which is a version of OCL extended with imperative constructs making it a powerful formalism for representing complete behavioural specifications at the modelling level. Furthermore, meta-model transformations that automatically derive the component implementation are demonstrated and specified in XMap, a transformation rule language embedded in XMF with a powerful pattern matching capability.

The remainder of the paper is structured as follows. Section 2 will introduce the rationale for choosing the OSS/J Inventory API, present an inventory domain-specific language meta-model with an example inventory PIM that instantiates the meta-model and describe a mapping of the inventory language onto a Java and tool meta-model in order to drive the automatic generation of an inventory application tool. Section 3 will discuss a number of lessons learned from this study. Section 4 finally will complete the paper with concluding remarks on the presented work and a description of further research plans.

2 A Case-Study: the OSS/J Inventory

The merits of MDA have been formally recognised by the TMF as it is signified, on one hand, by the recently announced strategic alliance between TMF and OMG and, on the other, by the similarities encountered between MDA and the NGOSS methodology **[Strassner 2004]**. This leaves little doubt that MDA will play a significant role in the telcoms industry and in particular in the development process of telcoms OSS. Nevertheless, however much appraised at a strategic level, there is no evidence for the use of MDA in practice to develop OSS solutions[2]. With intent on investigating the full power of MDA in the context of OSS solution design, we embarked upon a small scale case-study aiming to generate a fully functional OSS component implementation driven solely by technology neutral model specifications.

The case-study was based upon OSS component APIs specified in Java and J2EE by OSS/J. OSS/J have issued a document with standard J2EE architectural patterns and design guidelines all OSS component specifications **[Gauthier 2001]** should comply

[2] The authors have not found any case-studies reported in the relevant literature. Furthermore, even within the TMF's Catalyst programme, where TMF member companies collaborate to build demonstrators that apply the TMF standards in practice, no project has been setup as of yet with clear focus on the use of MDA to develop OSS solutions.

with. In order to test conformance to the defined specifications and guidelines in practice, OSS/J additionally produce reference component implementations and technology compatibility kits. The case-study was specifically driven by the OSS/J Inventory component API **[Gauthier 2004]** and set as its goal to prove the ability to automatically conduct such compliance tests by means of MDA. This end acquired more value by the fact that this particular API specification lacks, as of yet, a reference implementation and compatibility kit that would permit its practical validation. Given short project budget and timescale limitations, the study made careful assumptions, where necessary, in order to simplify the API's complexity without compromising its results.

The exercise targeted to a twofold outcome, as shown in Figure 2:

- **Automatic generation of PSMs conformant to the eTOM SID standard**: The OSS/J Inventory specification document includes a UML class diagram of an inventory meta-model and some textual, i.e. informal, description of its semantics. The meta-model defines the types of information/content the inventory will manage, such as products, services and resources. These types stem from a bigger model, namely, the Core Business Entities **[Reilley 2004]**, that OSS/J have defined in line with entities and interfaces encountered in eTOM SID for use by the OSS/J component APIs. In the case-study, we will be capturing the meta-model and some of its semantics in an MDA environment and instantiate it with example PIMs. Based on transformation rules, technology-specific representations for entities of an inventory PIM will be automatically generated. These representations collectively form a PSM. The PSM entities in this case will actually be Java classes or EJBs (entity beans) representing, one for one, the inventory PIM entities. While the case-study generated technology specific outputs in Java and EJB, the paper will focus only on the Java ones.

- **Automatic generation of a system implementation conforming to standard OSS/J architectural patterns and design guidelines:** While the PSM entities, i.e. Java classes or entity beans, bear the structure and deliver the behaviour of inventory entities as described in the original inventory PIMs, end-users should not interact directly with these entities. Rather, entities should be accessed through a single interface that exposes a simple set of management methods and hides their complexity. This is a standard OSS/J design guideline, which conforms to the *façade* design pattern and influences the architectural design of OSS/J components. In order to comply with the OSS/J guideline, the case-study aims at implementing an application tool that allows users to manage the inventory content through a simple GUI. Example users of such a tool may be front-desk operators who respond to customer calls and access the inventory to setup a new or change the state of an existing product/service instance. The case-study uses MDA to automatically generate the tool and associated GUI in Java and J2EE (session bean) in order to deliver the required OSS/J pattern and design guideline. Again, this paper only concentrates on the Java outputs.

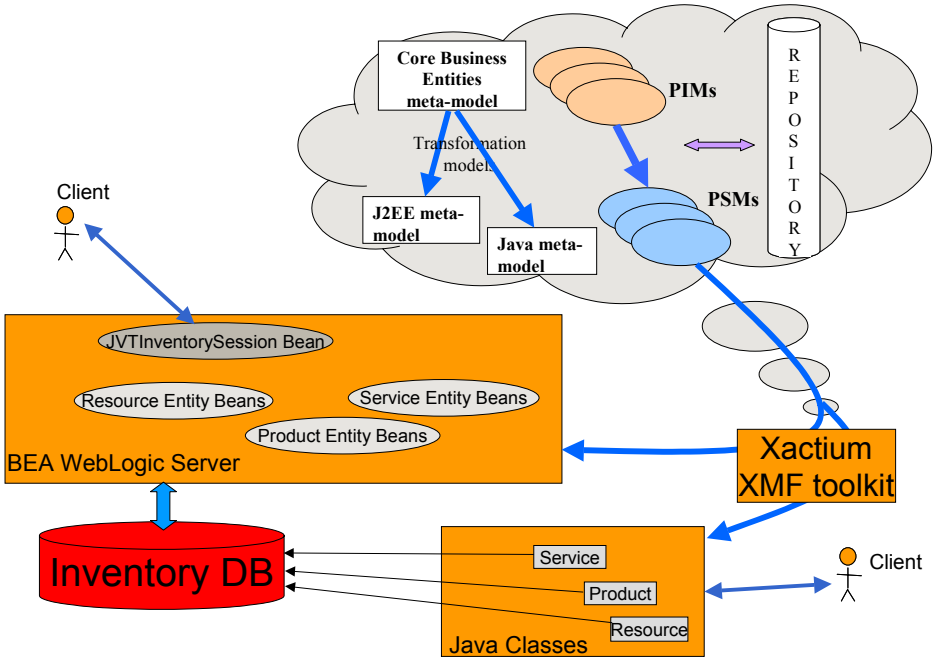

Figure 2 – The OSS/J Inventory case-study

Before embarking on the study, a brief evaluation of available MDA tools was carried out, such as iUML, Arcstyler, OptimalJ, Objecteering as well as XMF. It was found that for this particular application area, XMF offered advanced meta-modelling capabilities for expressing semantic aspects of models and definition of tools for any arbitrary language expressed using the MOF standard.

The XMF toolkit **[Clark 2004]** is a generic meta-programming environment that aims to support a wide variety of MDA development scenarios. To achieve this, XMF provides a variety of rich meta-modelling languages including: a package of OO meta-modelling concepts called EMOF, an executable version of OCL called XOCL, and a mapping language called XMap. Each of these languages has a well-defined executable semantics that is run on the XMF virtual machine.

Figure 3 shows how XMF was used to support the OSS/J Inventory MDA scenario. Firstly, a platform-independent domain specific language for inventories was defined by extending the EMOF meta-model. XOCL was used to specify meta-model constraints so that models written in the inventory language can be checked for correctness. That is, by means of XOCL, the meta-model semantics can be formally captured and automatically enforced, in contrast to the informal, textual description of the semantics presented in the OSS/J Inventory API specification document. Next, mapping rules written in XMap were constructed to transform the inventory meta-model into meta-models of two target platform specific languages: EJB and Java. This enables any model written in the inventory language to be translated into models that corresponded to programs written in EJB and Java. The former generates plain EJB code, which can then be manually deployed onto the BEA Weblogic application server. The latter is more sophisticated in that it generates a fully deployed Java tool for instantiating the generated models and for checking constraints and running operations on the models. The aim is to show that the source language was rich enough to be translated into sophisticated domain specific applications.

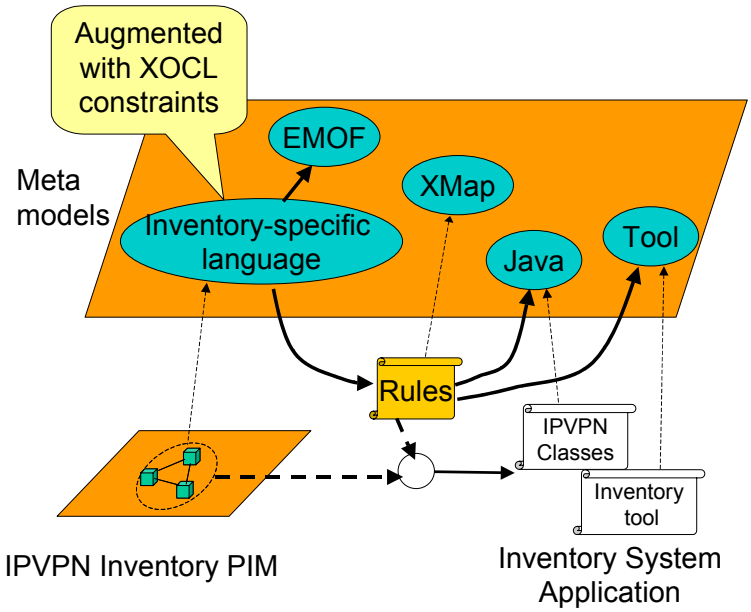

Figure 3 – Using XMF to deliver the Inventory tool

2.1 Domain-specific language

Figure 4 shows the inventory domain specific language meta-model. As mentioned earlier, it includes concepts from the OSS/J Core Business Entities, which are a subset of TMF's SID. The inventory language consists of the following constructs:

- *Entity*, that represents any type of information included in the inventory. According to the specification, three types of inventory content are defined, namely, *Product*, *Service* and *Resource*, which extend type *Entity*.
- *EntitySpecification*, that represents configurations of *Entities*, i.e. constraints, such as range of values or preconfigured setting on features of the *Entity*. Again, the API specification defines three subtypes of *EntitySpecification*, namely, *ProductSpecification, ServiceSpecification* and *ResourceSpecification*, each representing specifications for *Service*, *Product* and *Resource, respectively*.
- *EntityAttribute*, that represents relationships between *Entity* types.

To represent this inventory domain specific language, a meta-model is constructed with classes that specialise classes of the XMF embedded EMOF package, each of which EMOF classes has a well-defined executable semantics. More specifically:

- *Entity* specialises the class *EMOF::Class*, hence it can be instantiated and contain attributes, operations and constraints.
- *EntitySpecification* inherits from *EMOF::Constraint*. It can, therefore, be owned by an *Entity* and contain an evaluate-able XOCL expression. In the Inventory API specification document, *EntitySpecification* is represented as a UML class, which has a simple semantics, and thereby great modelling incapacity to express in full potential the concept semantics as an *Entity* configuration constraint. Therefore, by modelling *EntitySpecification* as a pure constraint, rich expressive power is conveyed to the concept enabling it to represent complex *Entity* configurations.

- *EntityAttribute* specialises the class *EMOF::Attribute* and is used to associate different *Entity* types.

A number of constraints (well-formedness rules) apply to the inventory language. These are expressed in OCL. As an example, the following OCL constraint states that if an *Entity* specialises another *Entity* it must be of the same type as the parent entity. That is, entity *IPStream_S* of Figure 5, for instance, can inherit from *IPStream*, as both are of type *Service*, but cannot inherit from *IPVPN* that is of type *Product.* Here, *of*() is an XOCL operation that returns the meta-class of the entity (i.e. the class that the entity is an instance of).

```
context Entity
  @Constraint SameParentType
    parents->select(p | p.isKindOf(Entities::Entity))->forAll(p |
        p.of() = self.of())
  end
```

Another noteworthy constraint formally delivering an important semantic property of the inventory meta-model, as per the OSS/J Inventory API specification document, involves the association of an *Entity* type with the correct type of *EntitySpecification.* In other words, classes of type *Service*, for instance, can only have specifications of type *ServiceSpecification* and not of type *ProductSpecification* or *ResourceSpecification*. Checking this and other similar constraints on a model that instantiates the inventory language meta-model can quickly and automatically validate the model for semantic correctness. The XOCL for the constraint follows.

```
context Entity
  @Constraint CorrectSpecs
    self.constraints->forAll(c |
     let ctype = c.of()
     in @Case ctype of
        [ IML::Entities::ServiceSpec ] do
          self.isKindOf(IML::Entities::Service)
        end
        [ IML::Entities::ProductSpec ] do
          self.isKindOf(IML::Entities::Product)
        end
        [ IML::Entities::ResourceSpec ] do
          self.isKindOf(IML::Entities::Resource)
        end
      end
   end)
```

Once the inventory language has been defined it is possible to create models that instantiate the language meta-model. An important question at this point is how this model can be visualised. One approach supported by XMF is to create a model of its diagrammatical syntax, which is then used to create a language specific diagram editor for the language. This has the advantage of being able to support very rich diagram types, but requires further modelling work.

A much simpler approach is to make use of a mechanism known as a *metaPackage*. Meta-packages allow a package to be represented as an instance of another package (its meta-package). Because XMF understands that the *metaPackage* represents a package of language definitions, it can provide appropriate stereotypes in the model

package. Note that *metaPackages* represent a stronger variant of profiles **[UML 2003]** because the stereotyped elements are real instances of meta-model elements (as opposed to being virtual instances). This way, NGOSS architectural guidelines, patterns and standards can be captured in a rigorous manner so that designers are capable of continuously validating their models against NGOSS artefacts.

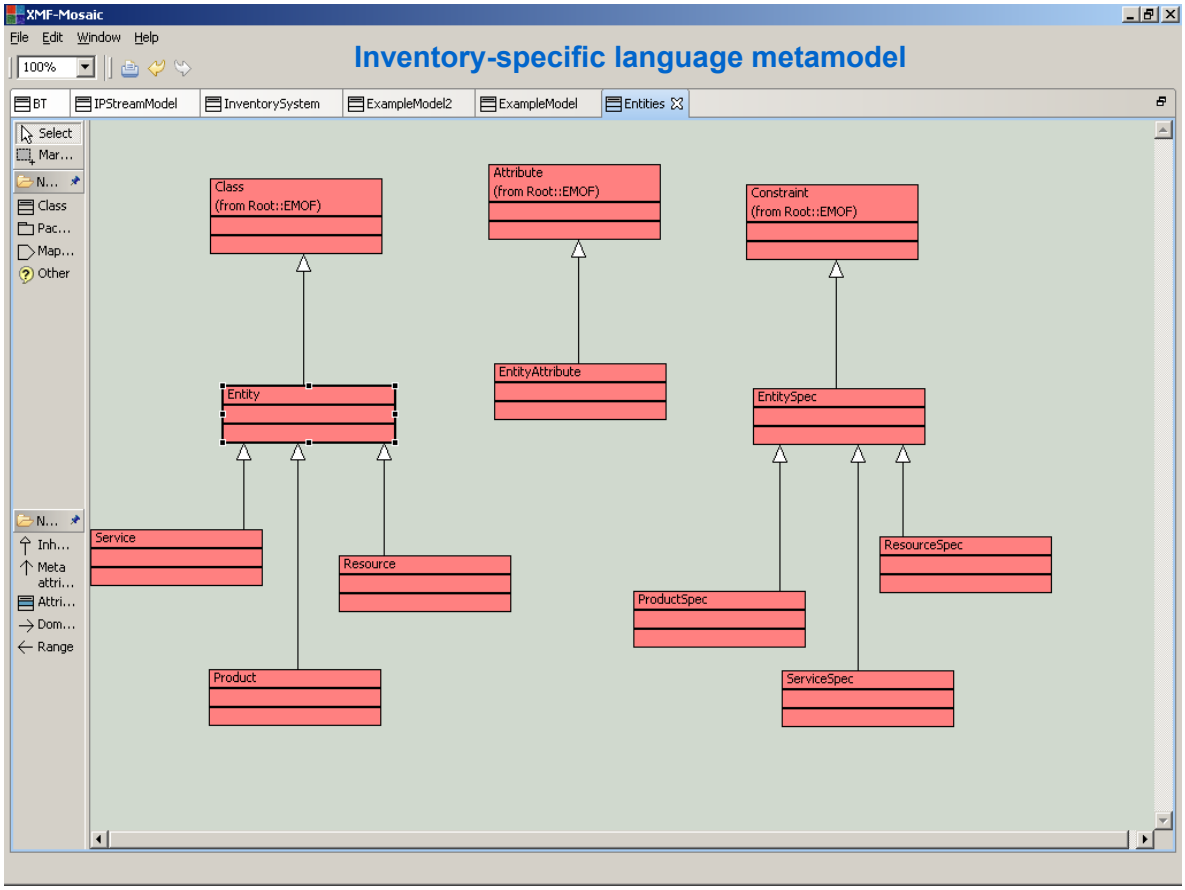

Figure 4 – Inventory-specific language

In Figure 5 a model is presented that is an instance of the inventory meta-model (its meta-package). It is based on an IP Virtual Private Network (IPVPN) product provided by BT and, in favour of simplicity, it only illustrates a subset of entities comprising the product[3]. The example IPVPN product, *inter alia*, would require a broadband link service between the connected customer ends. Hence, the model in Figure 5 shows *IPVPN* containing (*containedServices* attribute) many *IPStream* entities, a BT ADSL service that comes in different offerings for home and for office premises represented by *IPStream_S* and *IPStream_Office*, respectively. *IPStream_S* is further subclassed by *IPStream_S500*, *IPStream_S1000* and *IPStream_S2000*, entities differentiating on the downstream bandwidth of the link that is, respectively, 500, 1000 and 2000 kbps. Individual features of the latter entities are defined in the accompanying *ServiceSpec* constraints, namely, *S500Spec*, *S1000Spec* and *S2000Spec*. Similarly, features of the *IPVPN* product and the *IPStream_S* service are specified in the *IPVPNSpec* and *IPStream_SSpec* specification constraints.

[3] In reality, the IPVPN product at issue could come in different versions packaged with additional features to the broadband link, such as, equipment for the customer premises and/or frontdesk support.

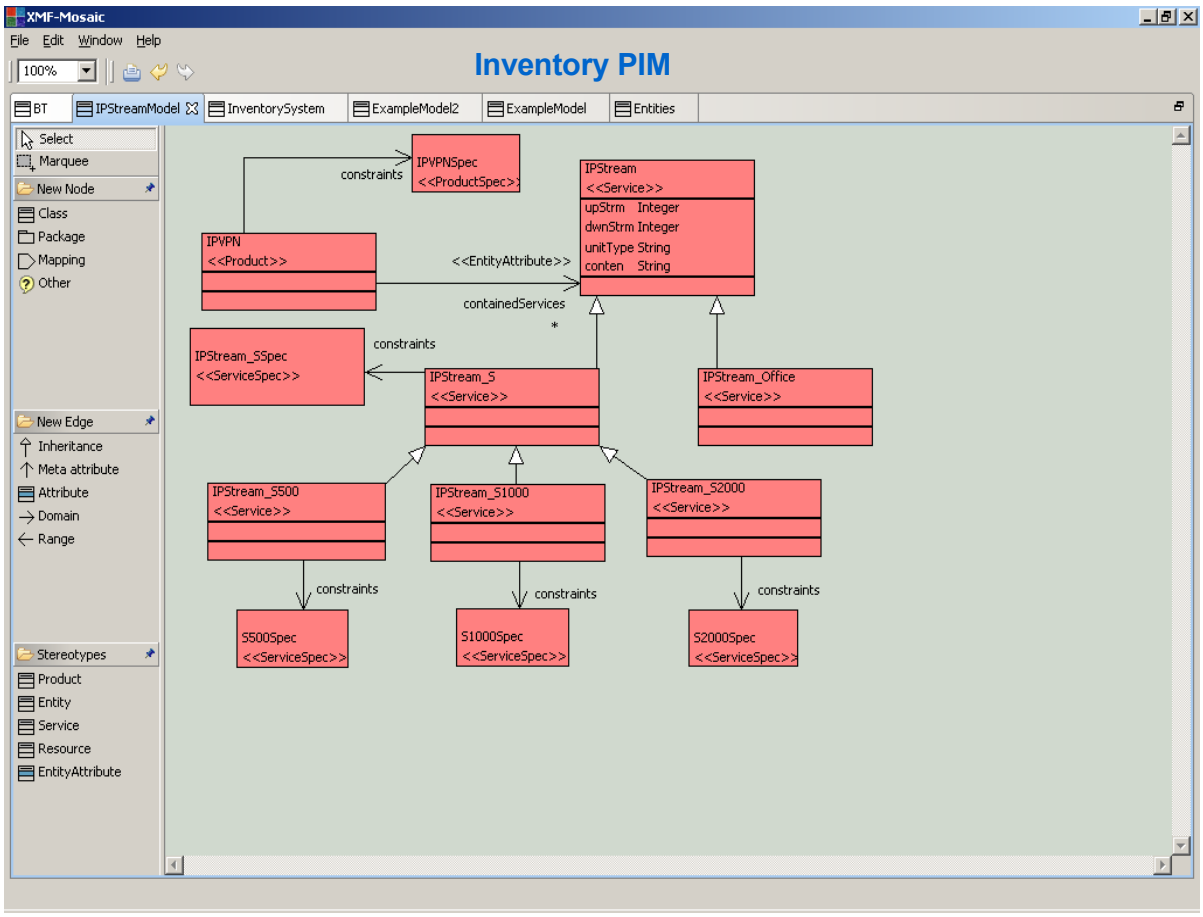

Figure 5 – Inventory PIM

All above model entities have as their types meta-classes defined in the inventory language meta-model of Figure 4. Hence, all entities in the model diagram of Figure 5 are shown as stereotyped classes constituting instances of the inventory domain specific meta-classes, for example, *IPStream_S2000* is an instance of meta-class *Service*. This way a PIM has been designed for the inventory using modelling constructs and semantics customary to the specific domain of interest.

Because all model entities of Figure 5 are instances of inventory meta-classes that specialise *Entity*, which, in turn, extends class *EMOF::Class*, they inherit the ability to have constraints, attributes and operations (and their associated specialisations, namely, *Specifications* and *EntityAttribute*). As an example, the *IPStream_S2000* is associated with *S2000Spec*, which has the following OCL body:

```
self.upStream = 250 and self.downStream = 2000 and self.unitType = "kbps"
```

In addition, XOCL can be used to write operations on the PIM model. XOCL extends OCL with a small number of action primitives, thus turning it into a programming language at the modelling level. As an example, the following operation creates an instance of an *IPStream* and adds it as a *containedServices* attribute to an *IPVPN*:

```
context IPVPN
  @Operation addIPStream(up,dwn,unit,con)
    self.containedServices :=
      self.containedService->including(IPStream(up,dwn,unit,con))
  end
```

Finally, because the entities in the model are themselves instantiable, it is possible to create an instance of the *IPStreamModel* and check that the instance satisfies the constraints that are defined in the PIM model. This is a further level of instantiation that is possible because of the *metaPackage* relationship between the inventory PIM model and the inventory language meta-model. Such a "snapshot" mechanism allows the validity of the model to be established early in the development process without the need to generate a prototype. In many respects it is more powerful than prototyping because it allows the construction and checking of counter-scenarios, that is behaviour that the system should not exhibit at runtime. This gives the designer confidence that the system eventually generated will function in the required manner. An example snapshot is shown in Figure 6.

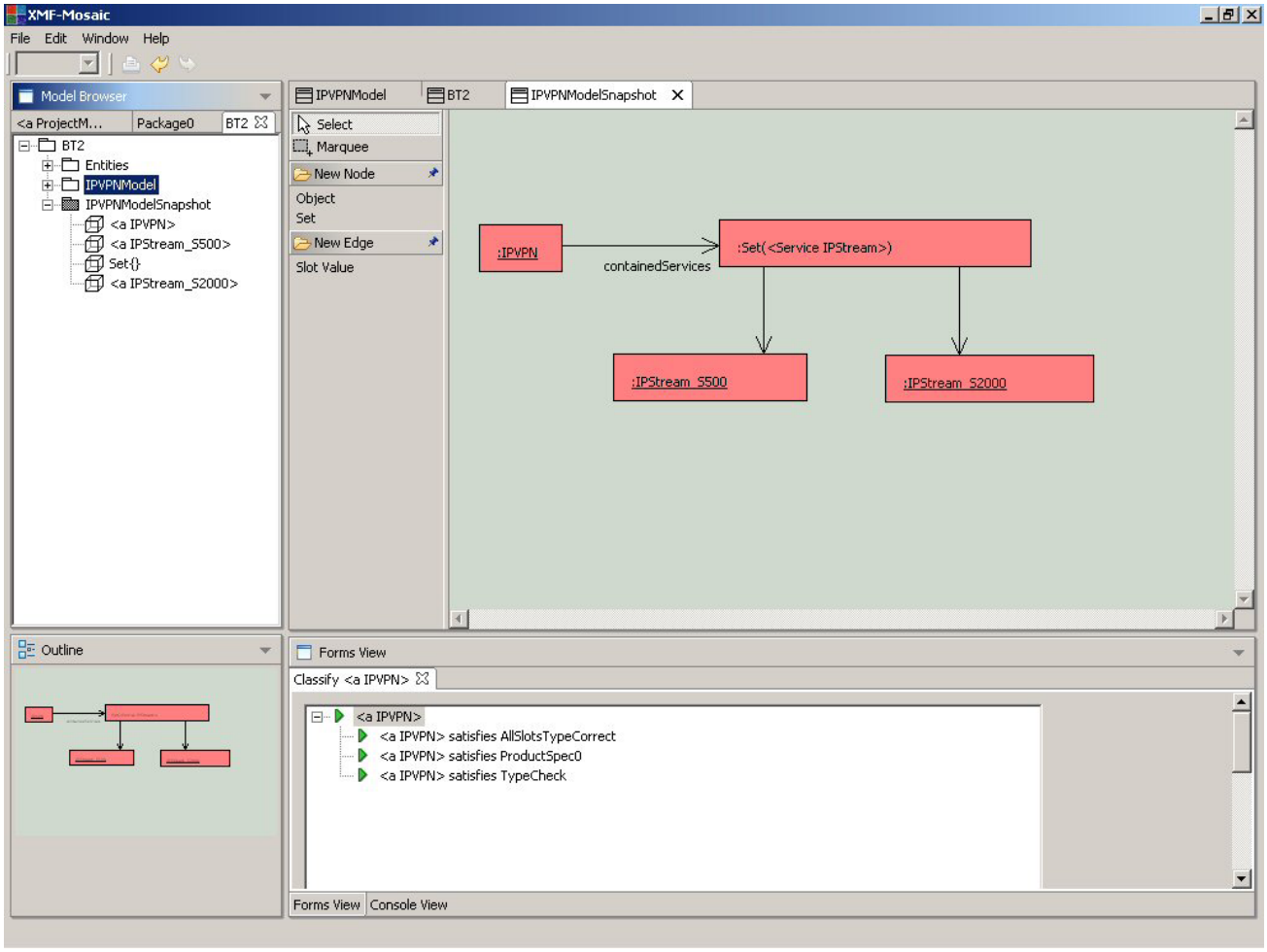

Figure 6 - A snapshot of the IPVPN model

2.2 Transformations of PIMs to PSMs

Using XMap, two mappings were defined from the inventory language. The first was to generate EJBs, whilst the second focused on the generation of Java and a Java class tool. We concentrate on the second one here.

The model of Figure 7 shows the mappings that were used to generate Java. Rather than mapping directly from the inventory language meta-model, a more generic approach was taken in which the mapping was defined from EMOF classes. Because the inventory language extends the EMOF meta-model, they therefore also apply to inventory models (and any other language specialisations defined in the future).

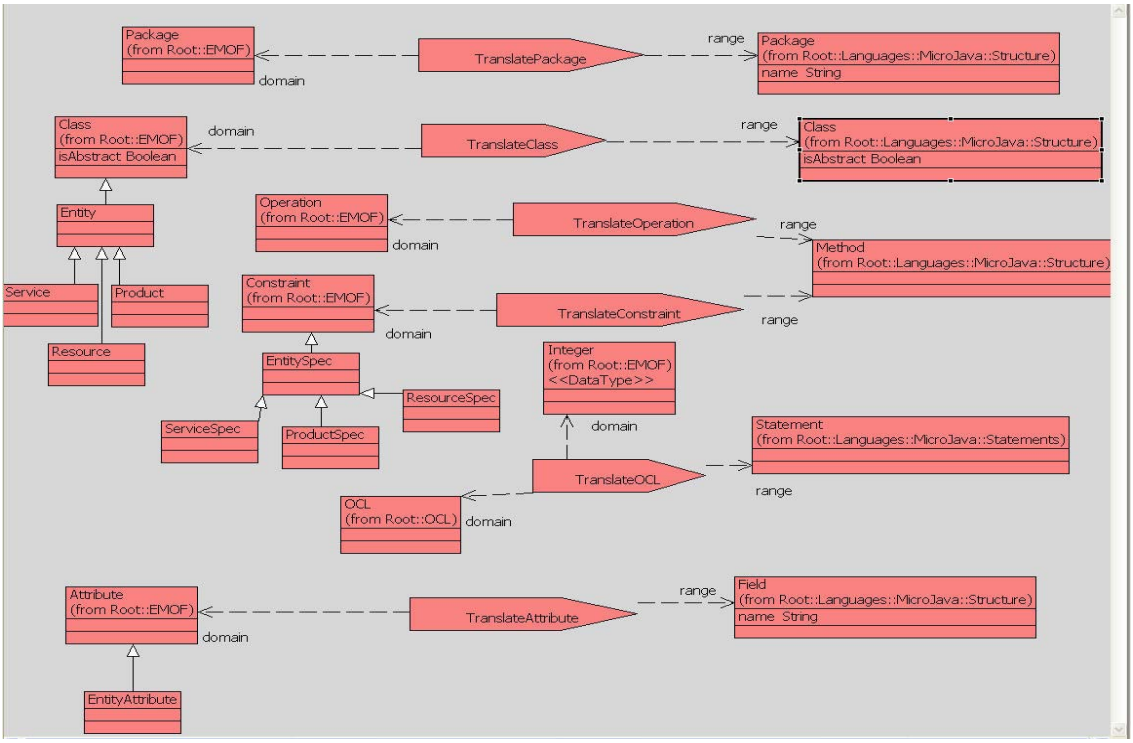

Figure 7 – Mapping of Inventory language to Java

Every element in the EMOF package has a mapping to a corresponding element in the Java meta-model. In XMap, mappings are represented by an arrow from source objects (the domain) to target objects (the range), and contain pattern matches between their values. An example of simple pattern match is described by the following XMap code:

```
context TranslateClass
    @Clause Class2Class
       EMOF::Class[name = N, attributes = A]
       do
       MicroJava::Structure::Class[name = N, features = F]
       where
         F = A->collect(a | TranslateAttribute()(a))
    end
```

Here, a Class is mapped to a Java Class, where the name of the Java Class matches the name of the Class and the attributes of the Class are mapped to fields belonging to the Java Class. Because the bodies of EMOF operations are also mapped, the mapping

results in generating an executable Java program that precisely implements the behaviour of the PIM. This Java code constitutes the PSM representation of the entities in the inventory PIM.

2.3 Tool Generation

Whilst the above mapping generates a standalone Java program corresponding to an inventory model, it would more useful to users of the language if the model it represents could be interacted with via a user interface. To achieve this, a mapping was constructed from EMOF to a meta-model of a class tool interface for managing object models. The meta-model of the class tool interface is shown in Figure 8. A class tool provides an interface that supports a standard collection of operations on objects, such as saving and loading objects and checking constraints on objects. In addition, a class tool defines a number of managers on classes, which enable instances of classes to be created and then checked against their class's constraints or their operations run.

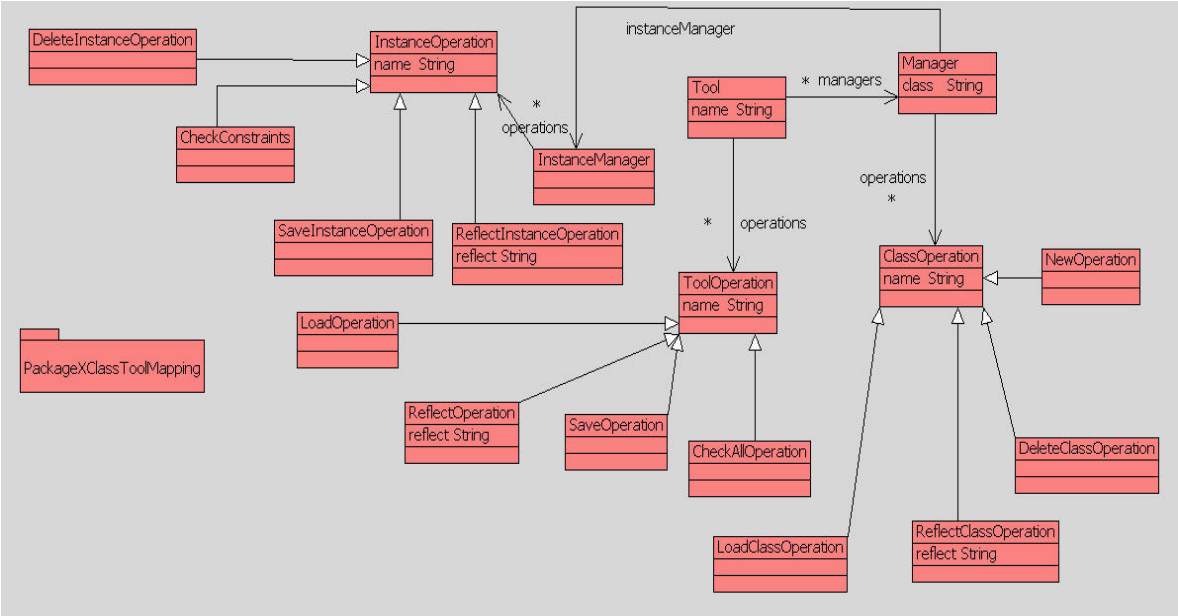

Figure 8 – Tool meta-model

For any EMOF model, a mapping can be defined to the class tool meta-model, which generates a tailored user interface for creating and manipulating instances of a meta-modelling language such as the inventory language. An overview of the mapping is shown in Figure 9. For each class in the source model, a user interface element is created which provides access to operations to create new instances of the class and to manage the operations and constraints provided by the class.

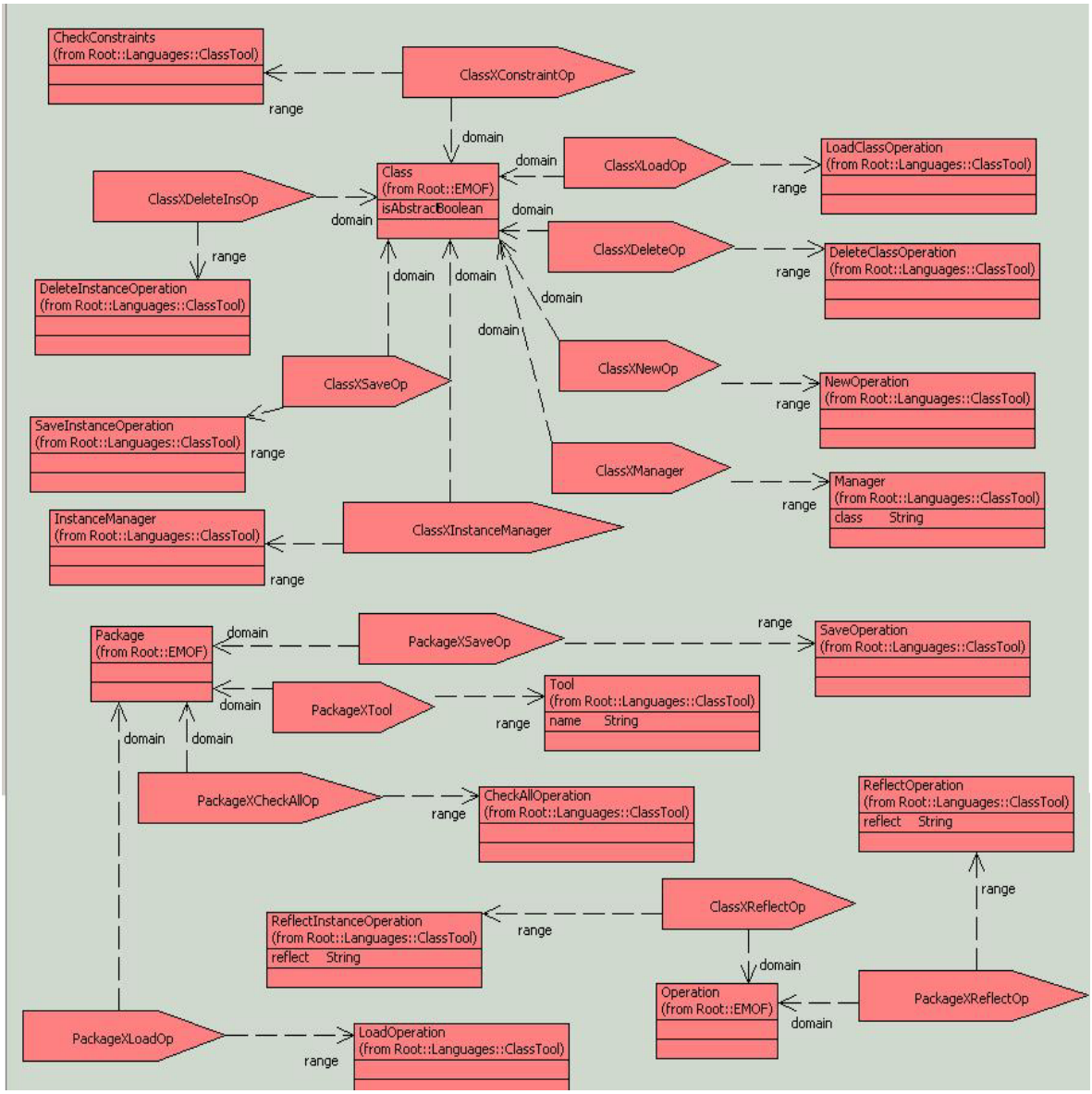

Figure 9 – Mapping of meta-modelling language to class tool meta-model

Applying this mapping to the IPVPN model shown in Figure 5 results in the generation of the class tool in Figure 10. Here, buttons have been generated for each of the entities in the model. These allow the user to create new instances, edit their slot values and delete instances. As the figure shows, a button for invoking the *addIPStream*() method defined earlier has also been added in the GUI executing functionality that implements in Java the method's behaviour described in the model with XOCL.

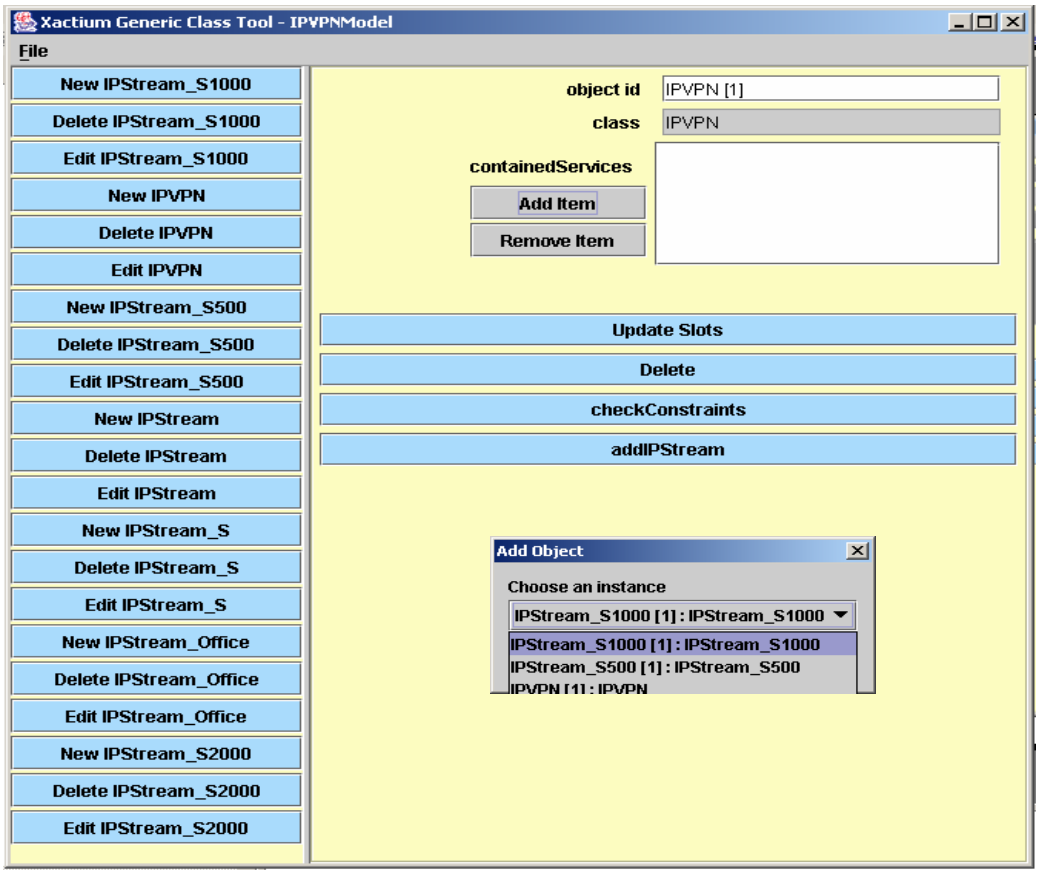

Figure 10 – Inventory tool

3. Lessons learned

A number of very interesting lessons were learned during the conduct of the case-study:

- **Models can be validated against precise meta-models.** The use of MOF and its well-defined, rich semantics for the definition of language meta-models allows for the construction of precise, non error-prone design models. All these models will be checked for validity against rules and constraints captured in the meta-models leaving no room for mistakes and ambiguity. The case-study demonstrated this through the example inventory PIM of the IPVPN product in Figure 5, which when checked against the semantics of the inventory language meta-model it sussessfully passed the validity tests as both meta-model constraints, namely *SameParentType* and *CorrectSpecs* are satisfied. What is more interesting is that tools, like XMF, are already available to provide the necessary automation in support of this process. This becomes more important in a large industrial environment, where solution desigers and developers constantly exchange models involving implementation and integration of complex OSS solutions and a correct understanding of the designs in one go can save in costs (no bouncing for explanations), produce results faster and minimise the possibility of error.

- **Models can be executed.** Models include full specification of structure and behaviour. Given an interpreting environment, a designer can execute the models and test them against different scenarios. The snapshot mechanism and the

interpreting environment of XMF facilitates just-in-time instantiation of models and running/simulating "what-if" situations. This eliminates the requirement of implementing a system prototype first before one can test the durability and robustness of a model. Executing the models is actually a form of rapid prototyping that takes place in the modelling space and very early in the development lifecycle. This capability is very useful in the context of OSS as with a little more effort spend in modelling early on, solutions and ideas can enter a fast-track testing stage before even a single line of code is put together.

- **Automatic generation of platform-specific implementations out of PIMs.** With a bit of more effort invested in the modelling phase, most part of a system's implementation can be automatically generated. This is ideal for the development of tactical solutions since systems are rapidly produced out of PIMs. Moreover, systems can survive through paradigm shifts and technology changes because PIMs remain intact. This is expected to have great effect in the gradual migration of legacy OSS onto new platforms. Additionally, existing systems can evolve as requirements change since every new feature or change introduced in the technology neutral model can be automatically reflected in the implemented system after re-generating the code. In other words, the use of MDA achieves synchronicity between models and system implementations as it is demonstrated, for instance, by reflective changes in the inventory tool GUI as soon as a new operation, e.g. *addIPStream()*, was added in the PIM.

- **Domain-specific languages can be standardised**. With the rigorous definition of appropriate meta-models one can unambiguously specify architectural styles, design patterns and guidelines. This is especially important in the environment of a large enterprise, which needs to apply company-wide and standardise across the business a particular set of system development principles or requires to precisely define a catalogue of reusable system capabilities, without room for interpretation. With special regards to the current NGOSS meta-modelling we could go far beyond its informal description in UML diagrams and specification documents, by capturing its full semantics using XOCL and by completely generating platform specifc models using XMap. This allows the architectural guidelines expressed in the NGOSS Technology Neutral Architecture be specified and enforced by automated tools, like XMF.

- **Specifications and standards can be verified for correctness very early in the lifecycle**. Ambiguities are removed. For instance, in the absence of a reference implementation and compatibility kit for the OSS/J Inventory, using MDA we could achieve fast validation of the specification both by executing the models to check model and meta-model constraint satisfaction and by generating an executable system in a technology of choice that would completely conform to the semantics captured in meta-model and PIM.

- **Standards and specifications can obtain full tool support from the very start of their textual definition.** This is due to the use of rigorous meta-modelling techniques with fully-defined and executable semantics based on OMG standards, such as MOF and OCL. Tools, such as XMF, which are based on these OMG standards, can be easily extended and customised to support new standards and specifications, such as NGOSS and OSS/J. The study, in particular, demonstrated

XMF's support of an important architectural OSS/J design guideline, the *façade* pattern, through the rigorous definition of the inventory language, the generic class tool meta-model and the eventual automatic generation of a front-end system that provides access to and management of inventory entity instances. Furthermore, the complete generation of executable systems out of meta-models and PIMs can rapidly provide prototype technology-specific reference implementations for practical tests of standards on criteria such as performance and scalability.

- **The richer the definition of the platform independent language (including semantics) the richer the mapping can be to platform specific modelling languages**. In particular, it is possible to generate 100% of the code necessary to support the execution of the translated model. In our case-study the inventory domain specific language is fully executable, hence 100% code generation was achievable.

- **MDA has the power to integrate many different types of languages and technologies**. The case-study, for instance, showed clearly the integration of a domain specific language (inventory meta-model) with a platform specific language (Java meta-model) and a user interface tool (class tool meta-model).

- **MDA tools are currently maturing towards constituting a viable and robust solutions used to capture all the complete structural and behavioural aspects of a system in a model**. Despite the small scale of the presented case-study, there was clear evidence that MDA supporting tools are viable, robust and worth be tested to a more extreme extent of an industrial scale.

4 Conclusions and future work

In this paper, we described how XMF, an advanced meta-modelling tool, can be used to develop, verify and generate models, code and GUI interface of an inventory system based on OSS/J standards. We demonstrated that a ***complete*** system description covering structural and behavioural aspects of the system can be captured in an executable model using extended meta-modelling and constraint languages, which are based on OMG standards.

We have demonstrated the power of the XMF meta-modelling tool producing an OSS component completely based on well-defined precise and accurate models. This has raised our confidence in the maturity of the MDA technology in a rich and complex OSS environment. Of course, OSS is more than merely a single component. Often it is made of diverse set of components based on varying platforms interconnected through complex integration hubs, business process, workflow and policy engines. Thus, the ongoing work aims at using MDA tools to provide fully automated support in all stages of an OSS methodology lifecycle.

Initially we intend to extend the model definitions to cover a few OSS/J components (such as trouble ticketing, QoS monitoring and service activation) and aim to capture integration logic of these components in an MDA tool. The integration logic will be captured at business and system view models of the lifecycle together with mappings to implementation and deployment views. An important aspect of the overall methodology is to encourage more reuse of the modelling artefacts in the entire

lifecycle and hence means are required to store model elements in meta-data repositories and enable designers and architects to discover suitable model fragments for use in their designs. This necessitates a method of expressing requirements for OSS components and business processes in a precise language before searching meta-data repositories.

In an OSS environment, code generation is not necessarily a prime driver for adopting MDA. This is due to the fact that the OSS industry is moving towards a plug and play architecture based on available COTS components as a way to reduce the cost of in-house development and reducing vendor lock-in risks. Hence, the greater emphasis is on development of rigorous architectural guidelines and frameworks capturing an enterprise's computing policies together with automated tool support so that the designs of OSS solutions (integration and OSS components, business process definitions and policies governing the behaviour of components and processes) can be validated and verified. This work thus is the first step towards building meta-models that capture the NGOSS lifecycle views, including various forms of components, contracts, process definitions, policies and their inter-relationships.

In conjunction with COTS components, it is recognised that operators tend to customise as much as 80% of COTS software resulting in high costs and use of proprietary tools. As part of the COTS modelling exercise, we intend to capture the customisation of COTS components in high-level models, where any modifications can be done at the model level and then using suitable transformations to apply the necessary changes on the COTS-specific development environment.

References

[Ashford 2004] Ashford C., "OSS through Java as an Implementation of NGOSS", White Paper, April 2004, http://www.ossj.org/learning/docs/wp_technologycomparison1.0.pdf

[Clark 2004] Clark T., Evans A., Sammut P., Willans J., "Applied Metamodelling", book to be published

[eTOM] TeleManagement Forum – enhanced Telecom Operations Map (eTOM), http://www.tmforum.org/browse.asp?catID=1647

[Gauthier 2001] Gauthier P., "OSS/J through Java J2EE Design Guidelines", OSS/J Architecture Board, October 2001, http://www.ossj.org/downloads/design_guidelines.shtml

[Gauthier 2004] Gauthier P., "OSS Inventory API – Overview (Part 1)", Public Draft version 0.9, OSS through Java Initiative, April 2004

[Georgalas 2004] Georgalas N, Azmoodeh M, "Using MDA in Technology-independent Specifications of NGOSS Architectures", First European Workshop on MDA (MDA-IA 2004), Enschede, The Netherlands, March 2004

[MDA] Model Driven Architecture, http://www.omg.org/mda

[NGOSS] TeleManagement Forum - New Generation Operations Systems and Software, http://www.tmforum.org/browse.asp?catID=1911

[OSSJ] OSS through Java Initiative, http://www.ossj.org

[Reilley 2004] Reilley J.P., Gauthier P., "Core Business Entities Concepts and Principles", 2004, http://www.ossj.org/downloads/cbe.shtml

[SID] TeleManagement Forum - SID Overview, http://www.tmforum.org/browse.asp?catID=2008

[Strassner 2002] Strassner J., Fleck J., Huang J., Faurer C., Richardson T., "TMF White Paper on NGOSS and MDA", TeleManagement Forum / Object Management Group, February 2004, http://www.tmforum.org/browse.asp?catID=1875&sNode=1875&Exp=Y&linkID=28972

[UML 2003] Object Management Group – The UML Specification, version 1.5 (final), March 2003, http://www.omg.org/docs/formal/03-03-09.pdf, pp 73-85

[XACTIUM] http://www.xactium.com