

# Using MDA in Technology-independent Specifications of NGOSS Architectures

Nektarios Georgalas\*, Manooch Azmoodeh\*\*

*BT Exact*

*Adastral Park, Orion Building – Ground Floor pp13, Martlesham Heath, Ipswich, IP5 3RE, UK*

*\*georgan@acm.org, \*\*manooch.azmoodeh@bt.com*

## Abstract

*Traditionally, telco's OSS have been developed in-house and were dedicated to the management of individual products and services. With the advent of increasingly vast growth in new product and service offerings and of new business models realised through complex e-commerce partnerships and supply chains, service providers require a more flexible approach to implement the OSS architecture of the future. The use of COTS products reduces maintenance and integration effort and when used in concert with a Service Oriented Architecture (SOA) an environment is created where OSS components, with standardised interfaces, can plug-and-play. The TeleManagement Forum adopts most of these principles in the design of the Next Generation OSS (NGOSS) architecture. Furthermore, it recognises the need for NGOSS architectural components and their designs to be specified in a technology neutral form in order to safeguard OSS solutions from the plethora of technology paradigms that may vary across different business domains or change over the solutions lifecycle. This paper sets out some of the requirements for technology independent specification of OSS architectures and reports on ongoing work in using MDA technology to realise the benefit of such an approach.*

**Keywords:** OSS, NGOSS, TMF, Component, Contract, OMG MDA, TNA, TSA, meta-modelling.

## 1. Introduction

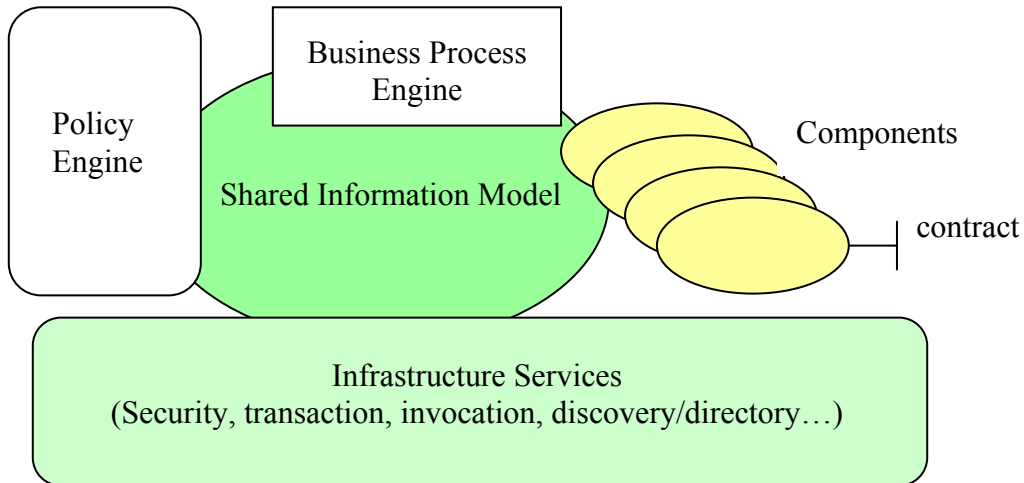
Traditional approaches to telco's OSS development have been around in-house development of OSS capabilities, often dedicated to individual services/products and networking technologies. With proliferation of new networking technologies as well as new services and products and in order to reduce the total cost of ownership and to increase business agility, service providers require a new approach to manage the OSS architecture of the future. There is a trend using COTS products in order to reduce maintenance and integration tax. However, without clear architectural principles in place, such attempts often cause more lengthy and costly operations. The solution to this is thought to be in adopting a Service Oriented Architecture (SOA) to create an environment where granular and loosely coupled OSS components (with some degree of standardised interfaces) can plug and play over a common infrastructure (service bus). Furthermore, as OSS solutions and technologies have different life cycle and thus change over different timescales, to safeguard much of the investment in the skilled activities of designing OSS, and provide longevity of design knowledge and solutions, the OSS architectural components and their design should be specified in a technology neutral form with automated support to map these to technology specific solutions as well as integrating and bridging these into existing OSS infrastructure. This paper sets out some of the requirements for technology independent specification of OSS architectures and reports on ongoing work in using MDA technology to realise the benefits of such an approach.

The remainder of the paper is structured as follows. Section 2 describes the proposed TMF architectural principles for building next generation OSS. Section 3 outlines the MDA principles in realising the TMF's vision. Section 4 describes how the specification of OSS components can be expressed in a technology neutral form. Finally section 5 concludes the paper with future directions of our research work.

## 2. TMF NGOSS Technology Neutral Architecture approach

The TeleManagement Forum [1] have established a working program called New Generation Operations Software and Systems (NGOSS) [2], which promotes specifications and guidelines for the design of the future OSS architecture. Specifically, NGOSS recognises that the OSS of tomorrow will consist of distinct components, with functionality offered as a pick and mix bundle. Its primary aims are to:

*“Enable an open environment where an OSS can be rapidly built/re-engineered using COTS components to achieve flexibility and adaptability to meet the demand of ever-faster changes in service life cycles, networking technologies and user demands as well as 3<sup>rd</sup> party OSS component vendor’s offerings.” [3]*



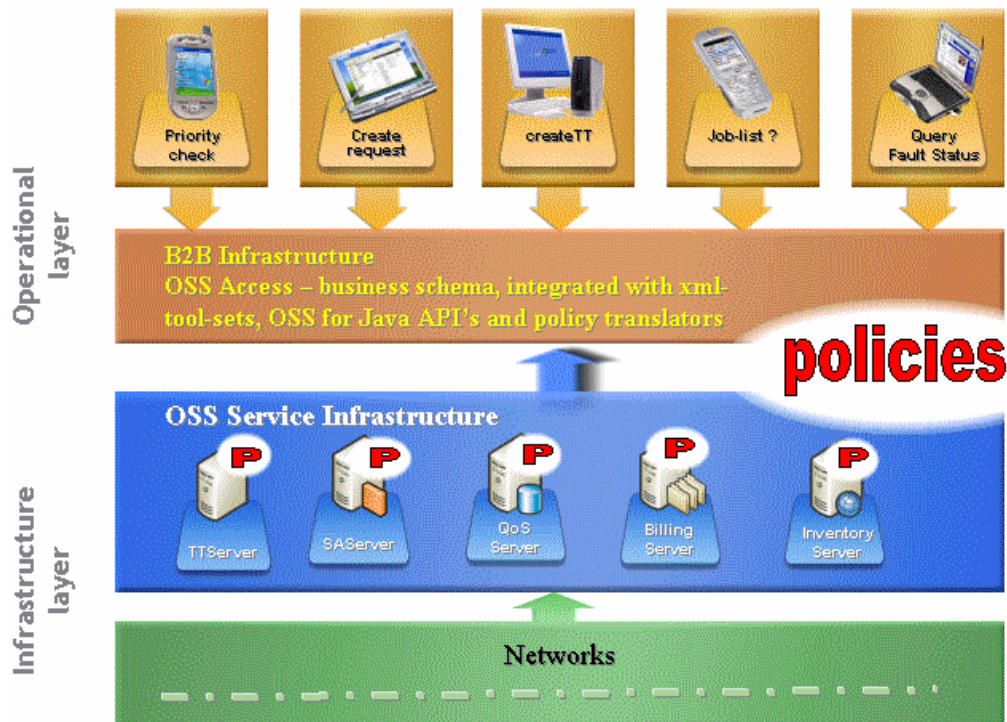
**Figure 1: NGOSS technology-neutral architectural principles**

The fundamental principles driving the NGOSS architecture are depicted in Figure 1 and are explained below:

- **Separation of business processes from components:** so that the business process is defined and managed separately from the more stable components which implement the service contracts used by processes
- **Contract defined component interfaces:** Component services are defined using contracts where, alongside service interfaces, other aspects such as pre-, post- conditions and QoS parameters are specified.
- **Separation of technology neutral (TNA) and technology specific (TSA) architectures:** NGOSS system design and component contracts are documented in specifications that are neutral to any specific implementation technology while separate specifications are produced to describe the mechanism for mapping TNA and contracts to specific technologies for the purposes of implementing and deploying an NGOSS system.
- **Shared Information and Data Model:** provides an information/data reference model and a common information/data language for all NGOSS resources
- **Common Communication Bus:** is used by all software entities to communicate with each other.
- **Policy enabled:** so that using a declarative notation the behaviour of an NGOSS system can be modified/constrained at run time without the need for conventional costly software development life cycles.

Two principal elements underpinning an NGOSS are (i) use of **processes** and **policies** to specify and realize desired **behaviour** and (ii) being able to exert **control** over that behaviour. These two ensure

flexible and adaptable behaviour and control necessary in an NGOSS environment to meet the demands of next generation OSS requirements.



**Figure 2: Technology-specific OSS deployment using OSS/J**

Another interesting OSS-related activity is the OSS through Java Initiative (OSS/J) [4]. OSS/J brings together a number of leading vendors who aim at practically demonstrating the NGOSS principles through real OSS implementations using the J2EE component technology. Secondly, the OSS/J participants define a set of standardized APIs for OSS and deliver reference implementations for them, which are available for free download and testing. Currently, the OSS/J community have implemented APIs for the Quality of Service, Trouble Ticketing, Inventory Control, Billing and Service Activation OSS components. These implementations form an OSS infrastructure (see Figure 2) where each component acts like a “black box” exposing functionality through its API. Above the OSS infrastructure hovers the operational layer which offers the front-end applications e.g. job management application available to engineers, and the business infrastructure for the OSS components e.g. B2B workflow engines and gateway APIs.

Figure 2 additionally shows how the OSS could be extended to include policies. Since policies are declarative behaviour specifications, i.e. they describe “what” the behaviour is and not “how” it is delivered, they are by nature technology-neutral artefacts. In the general case, policies can be executed either by generic policy handling tools at the operational level or by mechanisms inherent to the OSS component at the infrastructure level. However, all OSS developing vendors participating in OSS/J currently implement OSS components as sole providers of APIs that deliver OSS functional capabilities without supplying any inherent component capability for executing policies. It will be a future trend for OSS suppliers to provide a policy-enabled interface on their OSS components in order to exploit the leverage policies offer towards making OSS component behaviour more dynamic and adaptable.

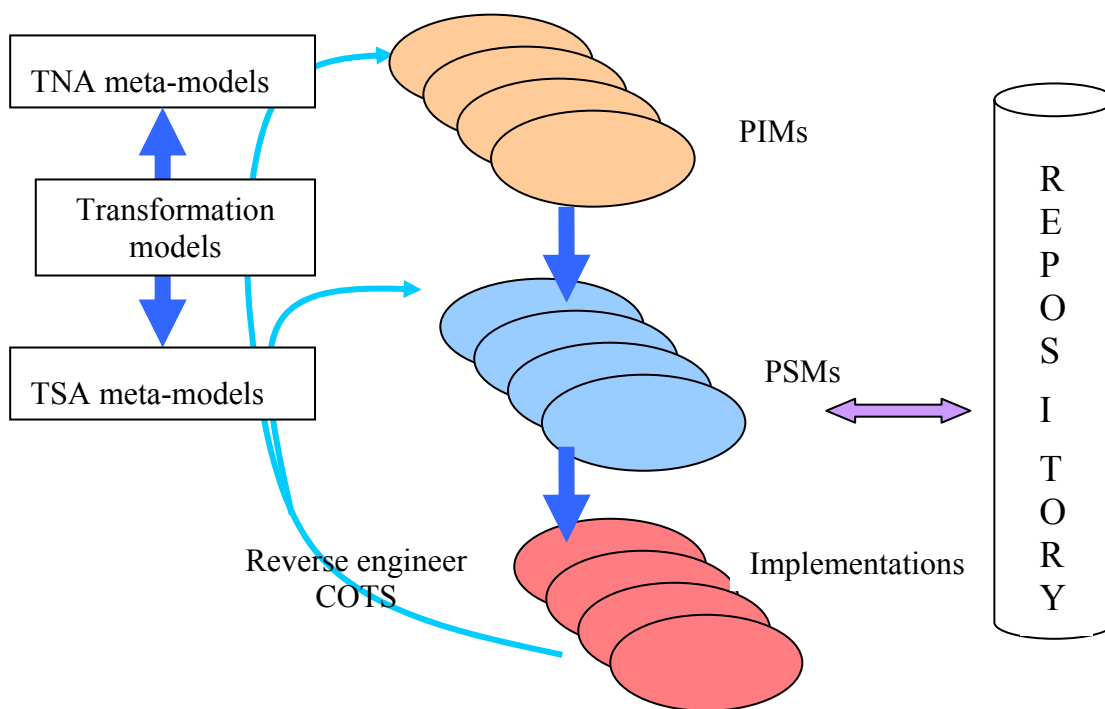
### 3. MDA-enabled OSS infrastructure

Model-based development of software is widely promoted through the established standardisation of OMG’s Model-Driven Architecture (MDA) [6]. The MDA emphasises the generation of technology independent models using the Unified Modelling Language (UML) at different points in the software development lifecycle, e.g. requirements modelling, system analysis modelling, design modelling. This enables the binding to a specific implementation technology to be made as close to the implementation stage as possible. The MDA defines mechanisms for exchanging software models based on a standardised

meta-model, the Meta Object Facility (MOF), and a serialised syntax using the XML Metadata Interchange (XMI) language, which therefore allows technology-independent models to be shared and reused between organisations, irrespective of the different implementation technologies used.

An integrated solution for an MDA-enabled OSS infrastructure is illustrated in Figure 3. A number of platform-independent models (PIMs) formally specify the OSS structure and behaviour abstracting away all technical details. In doing so, designers are assisted by technology-independent meta-models (TIMs), which provide necessary designing guidelines for the generation of PIMs. TIMs here represent the structure of NGOSS architectural components in a technology-neutral form. Next, the platform-specific models (PSMs) are generated to specify the OSS in terms of the target platform. A PSM uses the platform concepts of exception mechanisms, parameter types, component model and so on, in order to capture the technical system view of the OSS. Again, a set of technology specific meta-models (TSMs) comes in to assist the generation of PSMs. The transition from technology- and platform-independent to –specific models is facilitated by a set of transformation rules, which are cut out to provide the mappings between the two kinds of models. These transformations will finally result in the generation of OSS component implementations that should be capable of delivering the required OSS functionality. Additionally, a reverse-engineering process ensures that the legacy OSS, i.e. systems already operational in the business domain, is taken into account in the models of the different abstraction layers to an extent regarded sufficient by IT strategists and designers, considering that progressively legacy systems are destined for substitution by new, more flexible architectures. All models and meta-models are commonly stored in a repository that facilitates their sharing amongst the relevant stakeholders.

Note that in the OSS space, there is a greater emphasis on the mapping of PSMs into COTS components and not to code. Hence, the role of modelling and model transformation is: i) to provide validated specifications for 3rd party component vendors, ii) a technology neutral solution modelling environment for integration of COTS into the overall OSS infrastructure and generation of appropriate bridges, etc for interoperability across technology platforms.

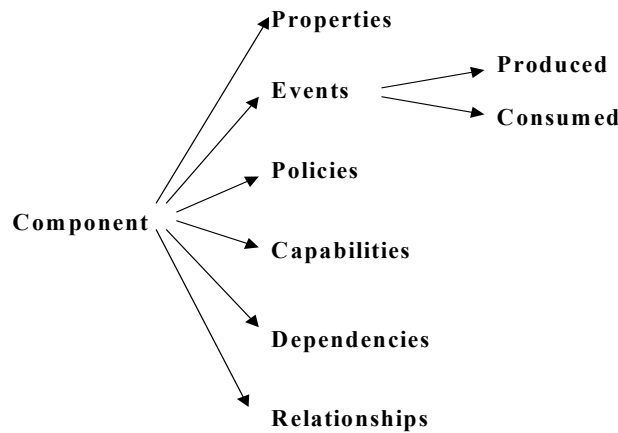


**Figure 3: MDA-enabled OSS infrastructure**

#### 4. Technology-neutral NGOSS component meta-model

According to the TMF NGOSS principles an OSS should be composed of fine-grain off-the-self software components, which are the providers of OSS capabilities. Additionally, the OSS should be documented in a specification that is neutral with respect to any specific implementation technology [3]. Consequently, it also becomes necessary for the components synthesizing the OSS to be represented in a technology neutral way. In recognition of this need, the TMF have introduced the NGOSS component concept as a

UML class within the technology neutral meta-model of the NGOSS architecture [5]. Usually, however, off-the-self components are offered as packaged solutions implemented specifically for a supporting platform or application framework, therefore any component description is technology-bound. In order to provide technology neutral specifications for OSS components a model should be available to explicitly define all required high-level concepts that generically describe a component. Unfortunately, the NGOSS meta-model does not go far enough in providing an analytical and explicit description of what an NGOSS component is, and apart from introducing the *NGOSSComponent* class it falls short of providing a sufficient NGOSS component model proposition. Having identified this gap, we present our proposition of a high-level, technology-independent, model to generically specify NGOSS components, which aligns with most of the general NGOSS principles listed in section 2.



**Figure 4: High-level meta-model for NGOSS components**

The proposed component model is presented in Figure 4. It encompasses a set of high-level constructs that altogether structure a rich technology-neutral modelling template for components. According to this template, a component is an accumulation of properties, capabilities, policies, events, dependencies and relationships.

*Properties* capture component configuration parameters and in general the component state. One very important property common across every component is the component-id, a universal identifier that uniquely characterises a component.

*Capabilities* specify the component behaviour. At the very minimum a capability is described by a component interface. Obviously, a component may support more than one interface, and thus, have more than one capability. A more complex capability version may take the form of a technology-neutral NGOSS contract. Further to the definition of interface methods, a contract contains additional knowledge that more rigorously describes the behaviour delivered by the component interfaces. Such knowledge typically includes pre-conditions, i.e. constrains that should hold before methods are invoked, post-conditions, i.e. conditions that will be true after the end of a method execution, invariants, i.e. constraints that should continuously be held and a set of data that need be shared with other components that use the component contract. The shared data can be captured in the component properties.

*Relationships* represent associations one component establishes with others. They are conceptual modelling constructs that assist the formation of component compositions for the delivery of complex services.

*Dependencies* indicate the indispensable and life-critical association of component A with another component B it entirely depends upon and without which it cannot exist or function. Although dependencies are special forms of relationships, they are treated separately in the component model due to their critical role in ensuring that the component is of a viable state. Dependencies may be static or dynamic. A static dependency is imposed on the component at development time coercing it by definition to operate only if the component it depends upon is also operating. The dynamic dependency emerges at execution time when due to special or changing circumstances component A is instructed, by a newly deployed policy for instance, to start component B in order for component A to sustain its operation seamless.

*Events* are messages generated as a result of the component incurring a change, such as a change in a property value or a capability invocation. The role of an event is to communicate with several destinations (*event consumers*) an incident that happened on a source (*event producer*). In this communication,

information can be exchanged between the interacting parties related to the special circumstances of the incident.

*Policies* are tightly-related to events. As a reaction to events occurred at an event-producing component, certain behaviour may be enacted on an event-consuming component. This behaviour is specified in policies. A policy is a declarative specification of behaviour performed upon the trigger of a received event. A designer may capture in a policy forms of behaviour that are not hard-coded in a component's logic but are rather deployable, manageable and executable elements that come into play at runtime, complementing the mainstream behaviour the component exposes through its functional interfaces.

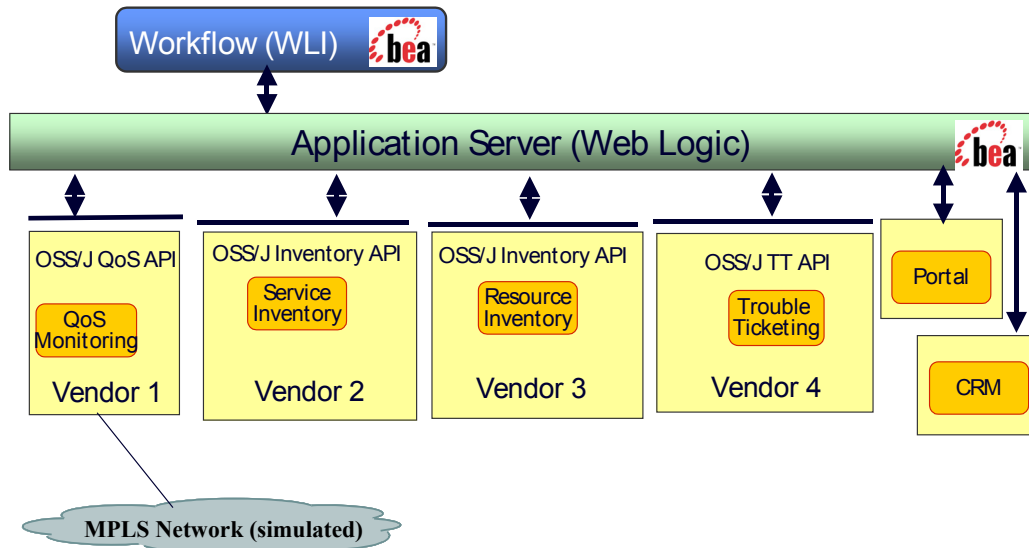


Figure 5: OSS/J testbed

## 5. Conclusions and Future Work

This paper outlined the main requirements underpinning the future OSS architecture as advocated by the TMF. The primary architectural driver lies in the separation of technology-neutral from technology-specific OSS specifications. This allows for OSS designs being developed and exchanged amongst stakeholders with no focus on the technical details, an issue to be tackled at the implementation phase. We introduced MDA as an approach to realise this TMF NGOSS principle. Finally, first results of our work in this direction were presented through the description of a technology-neutral meta-model for NGOSS components.

To solicit our experiments with MDA and NGOSS, we have undertaken the task of developing an OSS testbed (see Figure 5). The testbed consists of OSS components, which are provided by 3<sup>rd</sup> party suppliers and implement the standardised OSS/J APIs. The testbed is deployed on a dedicated internal BT network and uses BEA WebLogic application server as the hosting EJB platform.

The next steps of our work will involve the development of thorough TNM and TSM meta-models based on the OSS/J testbed. We will also concentrate on the translation of PIMs to PSMs in order to investigate what issues are involved and to what degree they impact on the efficiency of the model-driven OSS development process. In this activity we aim at evaluating tools that aid and automate the verification of PIMs against precise meta-models, expressed in MOF and augmented with OCL constraints, as well as the translation of PIMs to PSMs. The overall objective of this endeavour will be, firstly, to verify how Model-Driven OSS development makes a sustainable business case for telcos providing considerable cost reductions and faster marketing of new products and services and, secondly, to make recommendations on OSS-specific meta-models and supporting tools.

## Acknowledgements

The authors would like to especially thank Clare Bagley and Jim Hardwicke for their invaluable contributions to this work.

## References

- [1] The TeleManagement Forum, <http://www.tmforum.org/>
- [2] New Generation Operations Systems and Software, <http://www.tmforum.org/browse.asp?catID=397>
- [3] “NGOSS Architecture Technology Neutral Specification”, TMF 053, Public Evaluation Version 2.5, The TeleManagement Forum, May 2002
- [4] OSS through Java Initiative, <http://java.sun.com/products/oss/adoption/catalyst/catalystNice2002.html>
- [5] “NGOSS Architecture Technology Neutral Specification Metamodel, ANNEX TMF 053F, Member Evaluation, Version 1”, The TeleManagement Forum, February 2003
- [6] Model Driven Architecture, <http://www.omg.org/mda>

