A MODEL DRIVEN ARCHITECTURE FOR TELECOMMUNICATIONS SYSTEMS USING DEN-NG

John Strassner
Founder, MDAPCE.com

Keywords: DEN-ng, Model Driven Architecture, Network Management, Telecommunications Software, Web Services, Information Modeling, Data Modeling

Abstract: Current network management approaches rely on stovepipe architectures that can only share data through complicated mediation services that are costly to design and hard to maintain. This causes business rules to be embedded directly in the OSS, which means that every change on them requires direct modifications to the OSS. This paper describes an innovative approach that uses the DEN-ng models to represent the lifecycle of the solution. Extensions to the Model Driven Architecture initiative are presented which enable code to be generated for this approach. A case study is presented that illustrates the power of this approach.

1 INTRODUCTION

Current approaches to network configuration and management are insufficient for achieving important new business and technical objectives (John Strassner, 2002). These approaches lack the ability to define network services as a function of how a business operates. More importantly, they prevent network services from adapting to the varying demands of a changing environment. This adaptation is crucial for Service Providers and Enterprises to run profitable businesses. It is also a necessary foundation for both e-Business and newer approaches, such as autonomic computing (IBM Autonomic Manifesto), (HP Adaptive Enterprise).

For example, TL1, SNMP and CLI are unable to express business rules, policies and processes. This makes it impossible to use these technologies to directly change the configuration of a network element in response to new or altered business requirements. Instead, software must translate the business requirements to a form that can then be translated into TL1, SNMP or CLI (John Strassner and John Reilly, 2003). This disconnects the main system stakeholders – the business people determining how the business is run from the network people who implement network services on behalf of the business.

This problem is exacerbated by the proliferation of important management information that is only found in private MIBs, the many different dialects of CLIs, and the significant differences in the capabilities of a particular network operating system version. Fundamentally, these are all symptoms of a more strategic problem: common management information, defined in a standard representation, is not available. This missing piece prohibits different components from sharing and reusing common data, which leads to "stovepipe" architectures that are so common in the Operational Support Systems (OSSs) and Business Support Systems (BSSs) of today. This also requires a huge integration tax to be paid to integrate best-of-breed applications that were never designed to communicate with each other (John Strassner and John Reilly, 2003).

Worse, current approaches do not provide a complete view of the environment. This is because the information and data models that they use were designed to model the current state of a managed object. These models instead focus on technical concepts, such as how to model a laptop or the performance of a particular type of traffic, and rarely model business concepts.

This paper describes a novel approach, based on the DEN-ng models. It focuses on the ability to model the *lifecycle* of a system. DEN-ng, currently being developed in the TMF (TeleManagement Forum) (TMF SID, 2003), (John Strassner, 2005) addresses the entire *lifecycle* of the managed environment, including how to describe different

Strassner J. (2004).
A MODEL DRIVEN ARCHITECTURE FOR TELECOMMUNICATIONS SYSTEMS USING DEN-NG.
In *Proceedings of the First International Conference on E-Business and Telecommunication Networks*, pages 118-128
DOI: 10.5220/0001391001180128

views of information. This paper also describes initial code generation efforts.

In effect, DEN-ng is used as a *lingua franca* that abstracts the different programming models and functions of different vendor devices into a common form. Significantly, DEN-ng is not just for representing network devices and functions, but can also be used to represent people, Service Level Agreements (SLAs), business policies, and other entities of interest. This enables the entire spectrum of stakeholder interests – from the business analyst, to the product manager, to the system architect, to Network Operations Center technicians (and others) to participate in the definition, design, management, and retirement of the solution.

This paper is organized as follows. Section 1 introduces the problem and briefly describes the proposed solution. Section 2 provides a brief overview of the scope of the DEN-ng model, and its applicability to (tele)communications solutions. Section 3 briefly describes the OMG's Model Driven Architecture (MDA) initiative, and the enhancements made to it in this program. Section 4 describes the lifecycle model used in DEN-ng. Section 5 provides a simple case study to illustrate the practicality and applicability of this approach. Sections 6 and 7 provide conclusions and further work, and references, respectively.

2 DEN-NG OVERVIEW

The DEN-ng object-oriented information model provides a cohesive, comprehensive and extensible means to represent things of interest in a managed environment. Things of interest can include users, policies, processes, routers, services, and anything else that needs to be represented in a common way to facilitate its management. DEN-ng defines the static and dynamic characteristics and behavior of these managed entities as an abstraction of the entities in a managed environment. This is done independent of any specific type of repository, software usage, or access protocol.

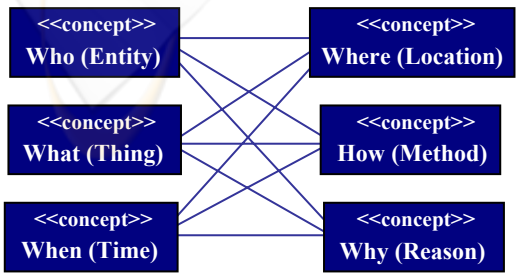

Figure 1: Zachman as Used in DEN-ng

DEN-ng was built using Zachman's perspective (Zachmann Framework) of "Who-What-Where-When-Why-How" for each object, as shown in Figure 1 below.

DEN-ng is based on the Unified Modeling Language (OMG UML – 1.5), the de facto international standard for defining information models. The "ng" denotes its support for the TeleManagement Forum's NGOSS architecture (TMF TNA, 2003), which is a framework that enables the rapid and flexible integration of Operation and Business Support Systems in the communications industry. The TeleManagement Forum is an international consortium of communications service providers and their suppliers (TMF Website).

UML provides a robust metamodel and facilities that enable it to be extended and customized to meet application-specific needs. UML can model the static and dynamic aspects of managed objects. ts metamodel defines the concepts of events and state machines. This is an important point, for as we will see, state machines are used in DEN-ng to model the lifecycle of managed objects. In contrast, the DMTF has its own "metaschema" that is not compliant with UML, and the ITU and IETF don't use UML at all.

DEN-ng is built as a framework of frameworks. This approach has two distinct advantages. First, it enables multiple modelers to work in parallel on different subject domains. Second, it enables information from other standards bodies and fora to be incorporated as is, or have their concepts and ideas mapped into the DEN-ng framework. Mapping is necessary if information is mined from an external standard or other document that isn't compliant with UML. In fact, information has already been mined from several different ITU Recommendations and IETF RFCs; plans include submission to the ITU-T.

The structure of the DEN-ng framework is

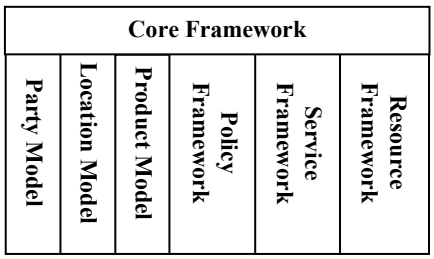

Figure 2: The DEN-ng Framework

shown in Figure 2 below.

DEN-ng is built on three foundational principles: capabilities, constraints, and context. Capabilities represent the set of functions that are available in a managed object. In effect, they normalize the

different commands and variations in functionality between vendor-specific devices.

Figure 3 shows an operator trying to accomplish the same task on two different routers. As can be seen, the CLI is completely different on these two devices. More importantly, the device on the left has different configuration *modes*, which are absent in the device on the right. Thus, the *programming model* for these devices is different. This means that the operator must be aware of these differences, and/or different management tools must be used. This is analogous to requiring the network technician to speak multiple languages fluently. However, even that analogy falls short, because of the significant changes that can be introduced in a new version of an operating system of a device.

Constraints define which capabilities can be used as a function of a particular context; context defines the current environment, objectives, obligations, and policies governing operation, and the desired behavior of the system.

For example, consider a device that can perform different types of encryption. Each different type of encryption is modeled as a capability. Constraints, such as ITAR regulations, restrict certain capabilities from being used in a given environment. The context is the overall environment that is being modeled.

DEN-ng supports the needs of different constituencies through the use of different *views*. This is similar to the RM-ODP concept of *viewpoints* (ODP, 96), but with an important difference. RM-ODP defines a set of disparate viewpoints that were only loosely related to each other. In contrast, DEN-ng defines a set of views that are strongly related to each other. This enables the needs of different constituencies, such as business analysts, architects, developers, operational support personnel, and others, to be associated with each other. Thus, each constituency can use the terms and concepts that they understand to express their needs, yet have those needs clearly communicated to other personnel having different areas of expertise in the organization.

For instance, a business rule can be translated into command changes for a device configuration to ensure that an SLA violation does not occur. This is much more complex than it first appears – the SLA, though technical in nature, is fundamentally a business concept and thus doesn't use networking terminology (e.g., queuing algorithms, drop probabilities, and so forth). Yet, it is these things that the network engineer needs to decide on in order to implement the appropriate traffic conditioning specified by the SLA. Note that this can be exacerbated by the concept of the *extended enterprise* – the set of business partners, external

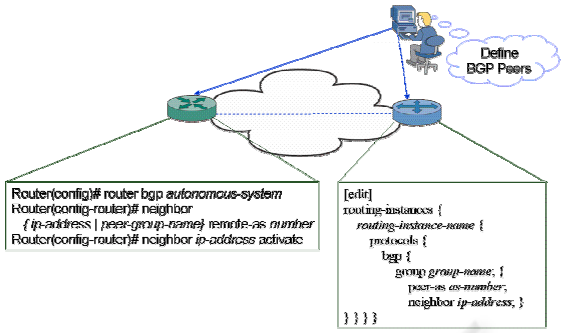

Figure 3: Different Programming Models

organizations, and other actors that take part in the business.

DEN-ng enables these different concepts to be associated with each other, so that other stakeholders can be assured that any two stakeholders are indeed speaking the same language.

DEN-ng is built using patterns (Pattern example, 96). These patterns frequently use other abstraction mechanisms, such as roles (Role Object Pattern, 97), to separate how an object is used from the basic definition of the object. This approach makes the model extensible, and helps model the real-world application of the model by embedding in the model facilities that enable it to be used by practitioners.

DEN-ng uses a finite state machine to enable the characteristics and behavior of elements being modeled to be represented throughout their entire lifecycle. Most other information models are "current state" models – they are limited to defining the behavior of a managed entity at a particular point in time. In contrast, DEN-ng models the static and dynamic characteristics of a managed object using different types of UML diagrams. In effect, static diagrams, such as class diagrams, define a set of building blocks that can be used to represent different features of a managed environment. Dynamic models, such as collaboration and sequence diagrams, use entities defined in these static diagrams to model the behavior of managed objects.

The TMF uses DEN-ng to form the core of its *Shared Information and Data* (SID) model, as well as its policy, service and resource domains. Currently, the business view of the SID consists of over 1,600 pages of documentation, as well as detailed UML models.

In summary, DEN-ng is the only information model that uses patterns and roles, along with abstractions like capabilities and constraints, to shape its design. It is also the only information model that represents the entire lifecycle of managed

entities. Its use of finite state machines enables it to be easily mapped to the OMG's Model Driven Architecture (MDA) initiative (OMG homepage).

Figure 4 is a simplified UML diagram showing some of the conceptual relationships that exist across the Product, Service, and Resource DEN-ng models. This provides a simple example of how different types of objects, and their views, can be combined. Conceptually, this figure shows that a Product, which is a business abstraction, contains Services and Resources. The power of this approach is that changes in these *business* entities (e.g., Product and SLA) can be directly mapped to changes in resource

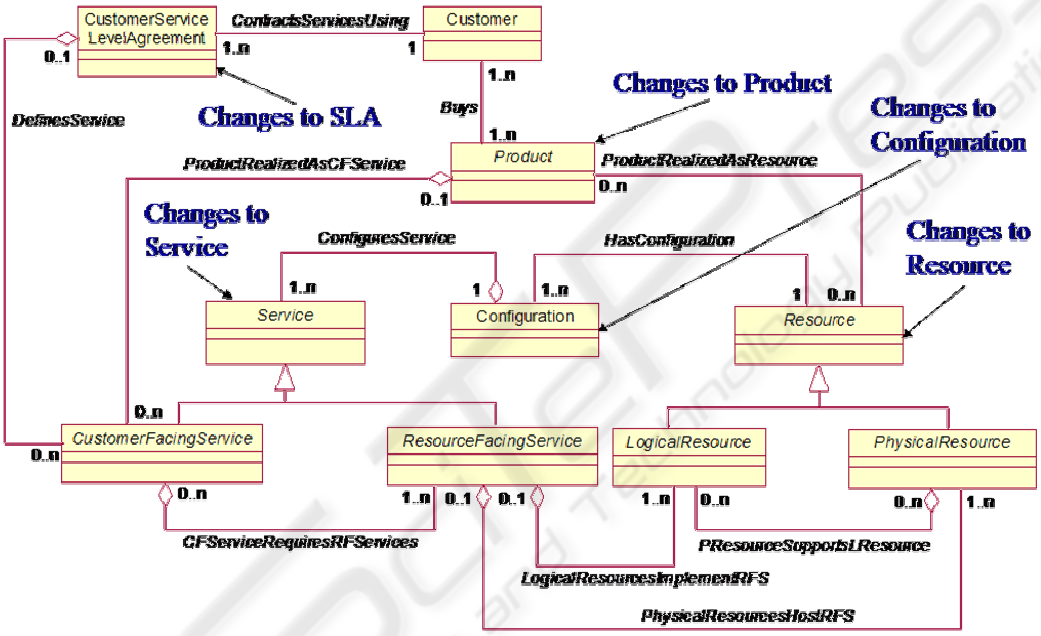

Figure 4: Business Driven Device and Service Management

and service configuration.

Thus, as a Product Offering changes to include new functionality, the underlying model changes to reflect the updated configuration of the services and resources making up the model.

3 MDA ENHANCEMENTS

MDA defines an architecture, expressed as a set of models, that separates the specification of system functionality from the specification of the implementation of that functionality.

In MDA, platform-independent models are initially expressed in UML, which is a platform-independent modeling language. Such models can be subsequently translated to platform-specific models through a set of mapping functions. There are a set of standards that support MDA, but these are beyond the scope of this paper.

Fundamentally, MDA seeks to integrate, through formal modeling, the entire lifecycle of a system: from business modeling to system design, component construction, through implementation and deployment. By providing this traceable evolutionary path, MDA enables the system to further evolve.

DEN-ng builds on and enhances MDA by first building a telecommunications *profile*. A profile is a set of model elements (not just classes!) that have been customized for a specific domain or purpose using the native extension mechanisms of UML (i.e., tagged data, stereotypes, and constraints, as defined in (OMG UML, 1.5)). As such, DEN-ng extends the UML metamodel to define additional constructs required to represent telecommunications concepts. This is further refined to represent concepts for modeling communications systems and networks. A second profile, the *NGOSS* profile, is then built to support the critical concepts defined in the TMF NGOSS program. This is shown in Figure 5 below.

For example, DEN-ng extends the concept of a UML state machine to use coordinated sets of state machines to model behavior. DEN-ng also defines the formal notions of policy (John Strassner, 2003) and

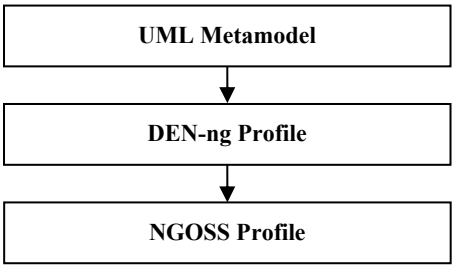

Figure 5: DEN-ng and NGOSS Profiles

contracts. NGOSS further refines contracts (TMF Contract, 2004). By defining formal profiles for DEN-ng and NGOSS, these new constructs can be used, just as standard classes are used, to construct new models.

Figure 6 below shows that different types of management data have varying characteristics. This means that different traffic conditioning mechanisms must be used for different traffic. Since these data have different characteristics, they require different storage mechanisms, and management applications will manipulate these data in fundamentally different ways. Thus, we may require multiple types of repositories, each having different characteristics, for representing management information.

	Voice	Mission-Critical	FTP	Email
Bandwidth	Low to Moderate	Low	Moderate to High	Low
Random Drop Sensitive	Low	Moderate To High	High	Low
Delay Sensitive	High	Low to Moderate	Low	Low
Jitter Sensitive	High	Moderate	Low	Low

Figure 6: Characteristics of Different Traffic Streams

Current approaches tackle this problem by building different stovepipe applications for each different usage. This makes it difficult to coordinate different management applications. In contrast, the formal modeling of DEN-ng enables the fundamental relationships between different types of data to be represented, allowing management data to be more easily manipulated, and consequently, management applications to share and reuse data.

An important point is that the above characteristics point out a need for increased intelligence of the network and its applications. The problem of managing different traffic streams is *not* a bandwidth problem! First, most Service Providers have plenty of bandwidth. Second, increasing the bandwidth for certain types of traffic, such as SNA or voice, doesn't help their performance – rather, they need guarantees on jitter and other metrics.

Management applications of the future present even more difficult challenges. For example, autonomic computing requires applications to change dynamically in response to changing needs. Plus, autonomic services and resources are likely to be dynamically composed from lower-level services and resources (John Strassner, 2004).

Thus, we must enhance MDA to enable it to address these and other advanced features.

Since the DMTF CIM, as well as the IETF and ITU models, are not UML compliant, it is unclear how they can be used with MDA. In contrast, the DEN-ng models are derived from UML, so they are perfect candidates for MDA. Furthermore, DEN-ng models the lifecycle of the solution, which is aligned with the goal of MDA.

As with the MDA approach, modeling behavior enables code to be generated to control how the different managed objects interact. MDA, however, has a number of existing problems that this approach seeks to correct (John Strassner, 2004). Behavior in DEN-ng is described through mechanisms for dynamically defining, configuring, and deploying policies and rules for defining and controlling how the elements in the FSM interact with each other. This formalizes the application of FSMs and codifies how state transitions are implemented.

Figure 7 illustrates the initial code generation process used in the DEN-ng MDA prototype. This process takes a specified UML model file as input – note that this includes static diagrams (e.g., class diagrams) as well as interaction diagrams (required to represent behavior) and other inputs, such as use cases (an integral part of DEN-ng and NGOSS contracts, and necessary to document the scenarios). Since an information model is independent of platform, language, and access protocol, it must be normalized to a common form which enables efficient data model development. This is done by the Schema Normalization process.

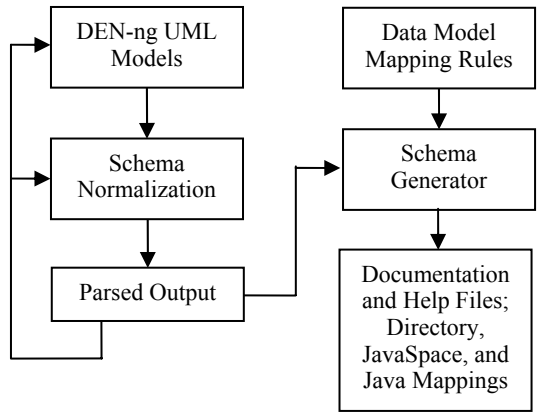

Figure 7: DEN-ng MDA Code Generator

The parsed output is then examined for errors, problems, and other inconsistencies. Significantly, these problems can be fed into either the UML model(s) directly, or to the Schema Normalizer. This flexibility is very important, as it enables systemic problems to be fixed in the model, and the Schema Normalizer to in essence "fine-tune" the results.

This fine-tuning consists mainly of producing output files that are programmer-ready in terms of resolving any ambiguities between the UML model and target platforms. For example, UML defines a "string" datatype, but this in and of itself is insufficient for generating code. What type of string is desired? What are its character limitations, if any? What are its specific syntax rules? Another example is producing import statements to handle different class dependencies.

This normalized result is then fed into the Schema Generator, along with a set of appropriate Mapping Rules. The Mapping Rules enable common concepts and components in the DEN-ng information model to be implemented in different types of repositories. These rules include packaging, persistence, exception handling, querying, and other platform- and language-specific rules to the generated output.

DEN-ng is much more than just a model of current state. DEN-ng uses the notion of a set of Finite State Machines (FSMs). Each FSM models a particular set of behavior, using components that are defined using the static parts of the DEN-ng model. The functionality of the heterogeneous components making up the solution is normalized using the DEN-ng capabilities abstraction. This enables common functions, such as queuing, to be represented in a standard form. The DEN-ng data model is used to translate from this vendor-independent standard format to a vendor-specific form, such as CLI.

Several outputs are produced by the schema generator. Currently, we have developed multiple data mappings (for directory, JavaSpace, and Java using JDO). For example, there are several property files available that control how the generator produces code. Additionally, a log file is output for inspection of informational, warning, and any error messages that were produced during the generation.

We have also automated the extraction of pertinent documentation from the SID addenda and from the UML model itself, and packaged that as programmer-friendly documentation.

It must be realized that many developers are not UML literate. Even if they are UML literate, the DEN-ng model is quite large (1600 pages for the business view alone!). Thus, we have started building programmer-friendly tools to help the developer better understand how to use the DEN-ng model. These include tools to simplify navigation in the model, tools that provide different options for different types of associations (e.g., what is the list of all classes that can be contained by this class, or vice-versa), and other design aids.

Additional work will soon start on building tools that enable other constituencies, such as business analysts, to similarly navigate the model according to their domain-specific needs. Since UML models are not natural vehicles for representing knowledge for several different constituencies, our approach is based on using another tool as the infrastructure, and importing or otherwise displaying UML diagrams when appropriate. This will be the subject of future follow-on work.

Reference (John Strassner, 2004) provides a brief overview of some innovative extensions to the OMG's Model Driven Architecture (OMG Homepage) initiative that show how different UML models can be used as "templates" to generate code. This enables behavior to be represented and implemented through automating the application of knowledge. This is, of course, the first step to implementing behavior autonomously.

4 DEN-NG LIFECYCLE MODEL

DEN-ng was built to model not just the current state of a managed entity, but the entire lifecycle of that managed entity, using an FSM. This concept is extended by coordinating multiple FSMs, which enable the behavior of components and entire solutions to be represented.

More importantly, DEN-ng recognizes that different stakeholders have a different view of a managed object. For example, the business analyst looks at an SLA object and sees an entity that

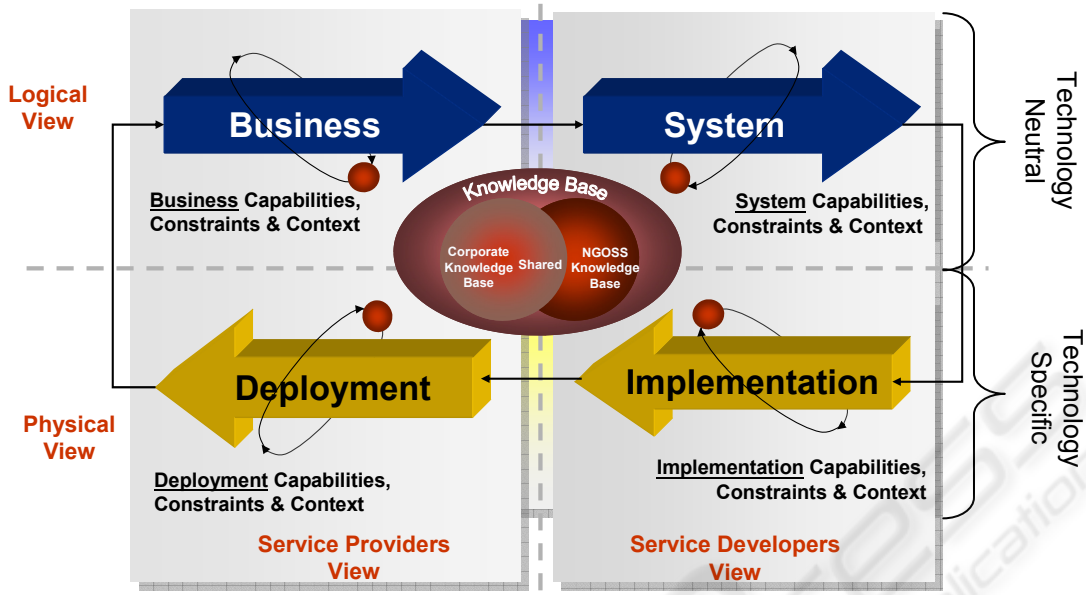

Figure 8: The NGOSS Lifecycle Methodology

represents a contractual agreement, whereas a network administrator looks at the same object and sees the different network services that must be supported using different functions.

This is done through using a novel approach, which consists of a combination of modeling (as described above) and a new methodology for applying the model, called the *lifecycle methodology*. This uses a combination of static and dynamic models to represent the overall behavior of the solution, using a set of FSMs.

The lifecycle methodology is based on integrating a set of different views of the solution (and its components). This is based on a principle called *progressive grounding,* where conversation between the partners is used to form common understanding that develops over time (H. H. Clark, 96).

Fundamentally, this methodology quantifies an important fact: the processes of analyzing business and system requirements, modeling the design, implementing a solution and then deploying the solution make up a lifecycle. The NGOSS Lifecycle effort (TMF Lifecycle, 2003) uses the four views defined in DEN-ng (business, system, implementation and deployment) and formalizes them into a cohesive methodology that can be used to define the characteristics and behavior of NGOSS components and solutions. This methodology is organized to enable flexible modeling of the solution. Sequential progress through the lifecycle is of course supported, but the NGOSS lifecycle methodology also enables organizations to progress through different aspects of the solution in whatever manner models the way their organization works.

The lifecycle provides traceability from the business definition through the deployment of a solution through defining views, as shown in Figure 8 below. These views enable the interests of various stakeholders to be protected by representing the evolution of their interests as the solution progresses from the definition of its business concerns, through its mapping to a particular architecture and implementation, through its deployment.

Specifically, the NGOSS Lifecycle contains two planes of interest: the Logical Plane, which is technology neutral, and the Physical Plane, which is technology specific. The Logical plane describes the conceptual design of the solution from a business and a system perspective. The Physical plane describes the actual implementation of the solution. The Logical and Physical planes intersect two different perspectives, the Service Provider perspective and the Service Developer perspective. The Service Provider perspective is concerned with the operation of the business, while the Service Developer perspective focuses on the underlying architecture and its implementation.

The Business View is used to describe the goals, obligations, and policies that will be used to drive the services offered by the business. This is done using high-level, technology-independent terms. The eTOM (TMF eTOM, 2003) and the SID are used to focus on the concerns of the business: processes, entities and business interactions. DEN-ng models these various tasks and functions as interaction

diagrams in which contracts are used to specify how information is exchanged between collaborating entities.

The System View is primarily concerned with the modeling of processes and information that affect the overall architecture of a component or a solution in a technology neutral manner. The SID, eTOM, and the NGOSS Architecture are used in conjunction to focus on system concerns: managed objects, the specification of behavior, and associated computational interactions, again in a technology neutral manner. Interactions are again used; the difference is that in the system view, contracts also specify how functionality is exchanged between collaborating entities.

The Implementation View puts the focus on *how* to build hardware, software, and firmware that will implement the system being designed. This View uses the particular NGOSS architectural style to map from the technology neutral specifications of the system to a selected target architecture. This drives the customization of an appropriate DEN-ng data model. Again, interactions are used; the difference is that contracts also contain implementation artifacts, such as code, APIs, or Web Service invocations.

Finally, the Deployment View is concerned with operating and actively monitoring the system to ensure that the observed solution behavior is what is expected – if not, then the behavior can be adjusted appropriately by using the NGOSS behavior and control mechanisms of process and policy based management. Once again, interactions are used to specify the details of the desired behavior. Thus, deployment contracts can contain a wide range of data (e.g., statistical performance data to evaluation the performance of the solution, policies to define specific behavior in response to particular stimuli, and so forth).

This approach enables business, technology, and product lifecycles of a given solution to be synchronized. The advantage of this approach is that it provides traceability from the business definition of the solution through its architecture, implementation and deployment. It also enables information and data entities to be further developed as part of the development process.

The DEN-ng lifecycle model uses contracts (John Strassner, 2004) as the "unit of interoperability". Contracts build upon the work done by ANSA as documented in the RM-ODP specifications as well as from current software technologies like Java. However, the previous work has not addressed the issue of different users requiring multiple views of the same object, such as a contract. For example, preceding work has focused on the definition of Application Programming Interface (API) aspects of interactions. This focuses on the implementation and run-time concerns of software interaction, but ignores its business and system aspects.

The NGOSS Architecture uses contracts in each of the four views. This means that contracts are *always* used to provide an interface definition mechanism that links the business, system, implementation and deployment aspects of a solution together.

An important point is that the NGOSS contract *morphs* as necessary to contain new and/or changed information. For example, a contract in one view could contain the specification of a service to be delivered; that same contract in a different view could specify information and code that implement the service. Thus, a contract is more than a software interface specification – it also defines pre- and post-conditions, semantics for using the service, policies affecting the configuration, use, and operation of the service, and much more. In short, the Contract is a way of *reifying* a specification of a service, and implementing the functionality of the service (including obligations to other entities in the managed environment).

5 CASE STUDY

A prototype of the NGOSS approach, including the DEN-ng MDA code generator, has been implemented and is being tested in a live Service Provider environment. Its focus is to prove that key provisioning tasks that result from a service order can be modeled and, hence, automated.

In this test, various product offerings consisting of rate-limited MPLS VPN services are used to provide secure connectivity. The customer can purchase different connectivity options, dictated by the combination of the type of customer premise equipment purchased, along with the number of different services purchased (e.g., VoIP services in addition to Internet connectivity) and their respective QoS.

The MPLS VPN is modeled as a product offering to ensure that the business policies and motivation for the VPN, on a per customer basis, are properly captured in the model. The DEN-ng product model defines a Product as an aggregation of PhysicalResources (e.g., a CPE device) and/or customer-facing services. Thus, a VPN is a customer facing service because the Customer is directly aware of the VPN.

The PhysicalResources and customer-facing services define logical resources (e.g., routing processes) and resource-facing services (e.g., BGP) that are used to ensure the proper operation of what the customer sees. For example, the VPN may

require the use of BGP. The Customer isn't aware of this, so BGP is not a customer facing service. However, since BGP is required for the VPN to work, it is modeled as a resource facing service.

DEN-ng models this product offering through a combination of roles and models. The major roles used are (TMF Service, 2003):

- VPNPhysicalDeviceRole
- VPNLogicalDeviceRole
- DeviceInterfaceRole

The VPNPhysicalDeviceRole represents the physical capabilities that a particular device has. It enables the correlation of physical and logical aspects of components that are used to route traffic. The VPNLogicalDevice Role abstracts the functionality required for the CPE, PE, and P (customer premise edge, provider edge, and provider core, respectively) roles of an MPLS VPN. Finally, the DeviceInterfaceRole enables different types of roles (e.g., edge vs. core functionality) to be associated with a particular DeviceInterface.

The DEN-ng QoS model (TMF QoS, 2003) enables different types of services to be abstracted. A ServicePackage defines the attributes, methods, relationships, and constraints that characterize the behavior of a particular Service as seen by the Customer (e.g., Gold Service provides VoIP access in addition to Internet access, while Bronze Service doesn't; in addition, the user gets faster downloads using Gold Service). A ServiceBundle is a collection of specifications of ResourceFacingServices. A ResourceFacingService is an abstraction that defines network-facing services. This enables different specifications for different types of traffic (e.g., QoS for voice traffic versus QoS for data traffic) to be defined. These are collectively bundled into CustomerFacingServices (e.g., "Gold Service" vs. "Bronze Service").

These abstractions enable the VPN model to be modeled in an extensible fashion, enabling different offerings from the Service Provider to be productized to suit different customer needs while maintaining a common abstraction. For example, Gold Service and Bronze Service may both contain data traffic, but offer different levels of QoS. Gold Service may be further differentiated by providing VoIP traffic. This is shown in Figure 9.

The DEN-ng model enables two different implementations of the same type of traffic (e.g., data or web) to have different traffic conditioning characteristics as defined by different customer service levels (e.g., Gold vs. Bronze).

Each service level is abstracted through roles. Roles are assigned to model the physical as well as the logical characteristics of the service. For example, every VPN service requires a particular physical port of a device to be used. This is abstracted through the VPNPhysicalDeviceRole, enabling all physical ports that play the same role in the network to be provisioned in the same way. It also helps define certain parameters that characterize the solution. For example, the CPE device can be connected to the PE device using a number of different combinations of media and protocols. The VPNPhysicalDeviceRole enables the invariant physical characteristics and behavior of media and protocol to be captured. Thus, certain types of connections can always be provisioned the same way.

Similarly, the VPNLogicalDeviceRole is used to provision CE-, PE-, or P-specific logical functionality. Finally, the DeviceInterfaceRole is used to provision common logical characteristics that this interface must have.

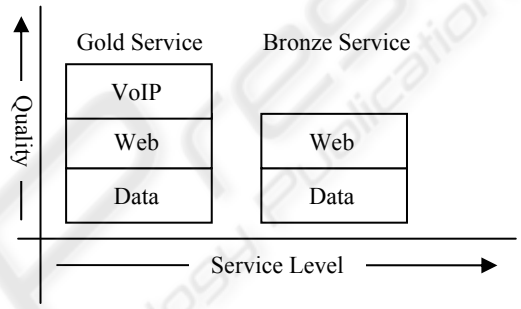

Figure 9: Different Service Packages

This combination enables the following workflow to be established. First, an event such as the insertion or removal of a new Card into a device is detected (e.g., by trapping a SysLog event). Second, the event is parsed to determine the particular action that has occurred. Extensions of the DEN-ng capabilities model are used to model the particular physical and logical characteristics of different vendor devices (Intelliden homepage). This enables specific action to be taken when a certain type of Card is inserted into a device. Finally, the physical and logical characteristics of the Card can be programmed without manual intervention.

For example, assume that a two-port GigE Card is inserted into a router. An event is received that says that a Card was inserted into a specific device. This Card is then identified using vendor-specific commands (e.g., show commands for a Cisco IOS router). The information gleaned from these show commands identify a DEN-ng model of the Card, along with a description of its physical and logical characteristics. A series of UML interaction diagrams define how to provision this type of Card using a set of appropriate roles. These roles in turn

identify parameterized configurations, stored as objects. This enables a physical port to be configured, and an appropriate device interface (e.g., an Ethernet sub-interface) to be instantiated. The generic functionality of the interface is programmed using the DeviceInterfaceRole abstraction; then, the VPN functionality of the interface is programmed using the VPNLogicalDeviceRole abstraction. When these tasks are complete, the relevant portions of the OSS (e.g., the billing component) are informed.

The FSM approach enables the interaction between these and other aspects of the overall solution to be modeled, providing a global view of the behavior of the solution. The Lifecycle is then used to model how this solution is defined, designed, implemented and deployed. Changes to the solution are captured as changes to individual parts of the lifecycle.

6 CONCLUSIONS

Current network management approaches rely on stovepipe architectures that can only share data through complicated mediation services that are costly to design and hard to maintain. Worse, they disconnect the various stakeholders from each other in two important ways. First, the lack of common information inhibits the sharing and reuse of management information. Second, the information models being used only describe the current state of the managed object.

This paper has described an innovative approach to modeling the lifecycle of the components of a solution using the DEN-ng models. DEN-ng is being developed in the TMF, and has several unique aspects that make it well-suited to modeling the lifecycle of managed objects. These include: (1) it is based on UML, and extends the metamodel of UML using UML approved techniques; (2) it has the concept of *views*, which enable different aspects of the same object to be described to best suit the needs of different constituencies; (3) it is built as a framework of frameworks, which enables knowledge from external standard bodies and fora to be incorporated; (4) it uses a set of powerful abstractions – capabilities, constraints, context, and roles, and patterns – to make its models inherently extensible. Most importantly, DEN-ng uses the concept of a finite state machine to model the static and dynamic aspects of a set of collaborating entities. This combination of characteristics is a unique contribution to the industry.

This paper has briefly described some significant enhancements to MDA, which enable the power of the models to be better utilized by various constituencies that aren't necessarily UML literate. A significant enhancement was the generation of tools and documentation to help users take advantage of the power of the DEN-ng models.

However, models are not a natural means of communication amongst most stakeholders. In addition, UML isn't well suited to representing many aspects of a solution, such as policy-based management. Even UML's use cases pale in comparison to the ability to convey information using simple textual templates. We are investigating using other tools as a framework in which UML-based tools plug in to provide specific functionality.

The NGOSS Lifecycle is a formal specification for defining the characteristics and behavior of NGOSS components. It also enables the interests of various stakeholders to be protected by representing the evolution of their interests as the solution progresses from the definition of its business concerns, through a mapping to a particular architecture and implementation, through its deployment.

A case study using the DEN-ng models, the MDA-based code generation, and the NGOSS lifecycle, was briefly described. The use of the DEN-ng model enabled the business, system, implementation and deployment aspects of the overall solution to be represented. FSMs enable the interaction of the various components making up the solution to be modeled; the lifecycle ensures that each stage in the evolution of the solution is properly represented.

REFERENCES

John Strassner – 2002: "A New Paradigm for Network Management – Business Driven Device Management". *In SSGRRs 2002 summer session.*

IBM Autonomic Manifesto: Please see: www.research.ibm.com/autonomic/manifesto

HP Adaptive Enterprise: Please see: http://www.hp.com/large/globalsolutions/ai.html

John Strassner and John Reilly – 2003: "Learning the SID". *Day Tutorial for TMW 2003 Fall Conference*

TMF SID – 2003: "Shared Information and Data (SID) model". *GB922 (and Addenda) and GB926 (and Addenda).* TeleManagement Forum, Dec 2003

John Strassner – 2005: "The Art of Information Modeling". *Book to be published*

Zachmann Framework: Please see: http://www.zifa.com/

OMG UML – 1.5: "Unified Modeling Language Specification", version 1.5, OMG, March 2003

TMF TNA – 2003: "The NGOSS Technology Neutral Architecture". *TMF053 and Addenda.* TeleManagement Forum, Dec 2003

TMF Website: Please see www.tmforum.org
ODP – 96: Open Distributed Processing Reference Model – Foundations, ISO/IEC 10746-2, 1996
Pattern example – 96: See, for example, M. Fowler, "Analysis Patterns – Reusable Object Models", ISBN 0-201-89542-0
Role Object Pattern – 97: In particular, variations of the role object pattern are used – please see http://www.riehle.org/computer-science-research/1997/plop-1997-role-object.pdf
OMG homepage: Please see: www.omg.org/mda
John Strassner – 2003: "Policy-Based Network Management", Morgan Kaufman Publishers, ISBN 1-55860-859-1, Sept. 2003
TMF Contract – 2004: "The NGOSS Technology Neutral Architecture – Contract Description: Business and System Views", TeleManagement Forum, TMF053b, Jan 2004
John Strassner – 2004: "Autonomic Networking – Theory and Practice". Tutorial for NOMS 2004
H. H. Clark – 96: *Using Language*. Cambridge University Press, 1996
TMF Lifecycle – 2003: "The NGOSS Lifecycle Methodology", TeleManagement Forum, GB927, Dec 2003
TMF eTOM – 2003: "Enhanced Telecom Operations Map (eTOM) – The Business Process Framework", GB921, v3.5, June 2003
TMF Service – 2003: "Shared Information and Data (SID) model – Service Overview Addendum", TeleManagement Forum, *GB922 Addendum 4SO,* Dec 2003
TMF QoS – 2003: "Shared Information and Data (SID) model – QoS Addendum", TeleManagement Forum, *GB922 Addendum 4S-QoS,* Dec 2003
Intelliden homepage: Please see www.intelliden.com