The NGOSS Technology-Neutral Architecture

TMF 053

Public Evaluation

Version 3.0

April, 2003

NOTICE

The TeleManagement Forum ("TM Forum") has made every effort to ensure that the contents of this document are accurate. However, no liability is accepted for any errors or omissions or for consequences of any use made of this document.

This document is a draft working document of TM Forum and is provided to the public solely for comments and evaluation. It is not a Forum Approved Document and is solely circulated for the purposes of assisting TM Forum in the preparation of a final document in furtherance of the aims and mission of TM Forum. Any use of this document by the recipient is at its own risk. Under no circumstances will TM Forum be liable for direct or indirect damages or any costs or losses resulting from the use of this document by the recipient.

This document may involve a claim of patent rights by one or more TM Forum members, pursuant to the agreement on Intellectual Property Rights between TM Forum and its members, and by non-members of TM Forum.

Direct inquiries to the TM Forum office:

1201 Mt. Kemble Avenue
Morristown, NJ 07960 USA
Tel No: +1 973-425-1900
Fax No: +1 973-425-1515
TM Forum Web Page: www.tmforum.org

ACKNOWLEDGMENTS

The TeleManagement Forum would like to acknowledge the contribution made to this effort by the following people:

Steve Afrin,	*ComInsights*
Francis Anderson,	*Tivoli*
Francesco Caruso,	*Telcordia Technologies*
Geoff Coleman,	*Briar Creek Limited*
Giuseppe Covino,	*Telecom Italia Group (current editor)*
Martin Creaner,	*TeleManagement Forum*
Cindy Cullen,	*Telcordia Technologies*
Tony Dean,	*TeleManagement Forum*
Petre Dini,	*Cisco Systems*
Paul Doyle,	*ADC Telecommunications*
Cliff C. Faurer,	*TeleManagement Forum*
Joel Fleck,	*Hewlett-Packard*
Jane Fox,	*Telcordia Technologies*
Wedge Greene,	*Worldcom (sponsor)*
Carlton Hall,	*CH2M Hill*
Martin Huddleston,	*Defense Evaluation and Research Agency (UK)*
Matt Izzo,	*Agilent*
John Kaplan	*Connexn Technologies*
Alain Lemoine,	*Eftia OSS Solutions*
Paul Levine,	*Telcordia Technologies*
Dan Matheson,	*Hewlett-Packard*
Bruce Murrill,	*TeleManagement Forum*
Christopher Prowse,	*Defense Evaluation and Research Agency (UK)*
Jonas Rabin,	*Lucent Technologies*
Dave Raymer,	*Motorola (team lead)*
Jean-Luc Richard,	*Evidian*
Tony Richardson,	*TeleManagement Forum*
Jeff Risberg,	*TIBCO*
Ken Roberts,	*Cisco Systems*
David Robinson	*Qinetiq*
Edward Scott,	*PrismTech*
John Strassner,	*Intelliden*
Joe Sventek	*University of Glasgow*
Al Vincent,	*TeleManagement Forum*
Stuart Ward,	*Orange*
Jim Willits,	*Hewlett-Packard*

ABOUT THIS DOCUMENT

TM Forum Documents

The NGOSS Technology Neutral Architecture Specification is being issued as Public Evaluation Version 3.0.

The purpose of an Evaluation Version is to encourage input based on experience of members and the public as they begin to use the document. Following the Evaluation Period, documents that are seen to deliver value are candidates for formal approval by the TM Forum. All documents approved by the TM Forum undergo a formal review and approval process.

This document will continue under formal change control. Further work will be reflected in revisions to this document.

Revision History

Version	Date	Who	Purpose
1.0	2001.01.01	TMF	Release for member evaluation (reformatted)
1.01	2001.01.13	D. Raymer, R. Trivedi	Updates based on TMF membership evaluation
1.02	2001.01.16	A. Vincent	Dublin Meeting Rough Changes
1.03	2001.01.24	A Vincent	Post Dublin Review
1.04	2001.01.29	A. Vincent	Post Dublin Review 2
1.08	2001.02.09	D. Lewis	Dublin input edited for consistency
1.09	2001.03.30	D. Raymer, R. Trivedi	Rework from Dublin, Paris Inclusion of ARCs and Comments from BAC/SDM
1.5	2001.04.27	TMF	Updated for Member Evaluation
1.60	2001.08.21	D. Raymer	Updates based on Long Beach TAW review
1.99	2001.09.07	D. Raymer	Updates based on TSA input. Release for working team evaluation, prior to general member evaluation.
2.1	2002.05.02	J. Strassner	Edits per 053_series_update_team, concentrating on policy, the shared information/data model, and process. Minor editing and clarification updates also included throughout the document.
2.5	2002.05.27	TMF	Formatted for Public Evaluation
2.6	2002.12.23	J. Sventek	Major edits reflecting red team architectural discussions with regards to RM-ODP, binding, contracts, etc.

2.7	2003.02.21	G. Covino	major rearrangements of contents in section 2 and 4 to clear distinguish between requirements and NGOSS principles; other minor editing like reducing details on Policy and Process models
3.0	2003.04.07	G. Covino	tuning of definitions, general formatting

Time Stamp

This version of the NGOSS Architecture Technology Neutral Specification, TMF 053 Version 3.0, can be considered valid until May 2003.

How to obtain a copy

An electronic copy of this document can be downloaded at the TM Forum Web Site (www.tmforum.org), Publications or through a link to Publications from a specific team's public project area.

If the document version has only been released to members for review and comment, the document can be downloaded from the Members Only area of the TM Forum Web Site.

Paper versions can be made available for a small fee to cover copying, mailing and handling. If you would like a paper version of this document, please order via the TM Forum Web Site or contact the TM Forum office. If you are not a member, please contact the TM Forum office on +1 973-425-1900.

How to comment on the document

Comments must be in written form and addressed to the team editor for review with the project team. Please send your comments to the editor using the editor's e-mail shown below.

Giuseppe Covino, Telecom Italia Lab *giuseppe.covino@tilab.com*
Dave Raymer, Motorola *david.raymer@motorola.com* *(team lead)*

TABLE OF CONTENTS

LIST OF FIGURES

PREFACE

About TeleManagement Forum

TeleManagement Forum is an international consortium of communications service providers and their suppliers. Its mission is to help service providers and network operators automate their business processes in a cost- and time-effective way. Specifically, the work of the TM Forum includes:

- Establishing operational guidance on the shape of business processes
- Agreeing on information that needs to flow from one process activity to another
- Identifying a realistic systems environment to support the interconnection of operational support systems
- Enabling the development of a market and real products for integrating and automating telecom operations processes

The members of TM Forum include service providers, network operators and suppliers of equipment and software to the communications industry. With that combination of buyers and suppliers of operational support systems, TM Forum is able to achieve results in a pragmatic way that leads to product offerings (from member companies) as well as paper specifications.

TeleManagement Forum supports several kinds of projects, including Process Automation and Technology Integration projects, as well as Catalyst Projects that support both Process Automation and Technology Integration projects through real-product integration.

1 Introduction

1.1 Background

With the gap between the worlds of data communications and telecommunications rapidly disappearing, the complexity and size of the networks supporting the emerging information and communications industry is increasing at a corresponding rate. The ability of existing OSS (Operation Support System) solutions to manage information and communication networks is being rapidly outpaced by the growth of the networks, the complexity of applications being run on the network, the increasing number of users of the network, and the sophistication of services run on the network. A new generation of OSS is needed to manage the new generation of networks and services.

The Operational and Administration Management system of today has become a roadblock to innovation, instead of a business tool for competitive success. The pace of change in the industry is such that incremental improvements will not work effectively. Service providers must both increase their operational efficiency by an order of magnitude and reduce the time to market of new services to compete, while software developers and systems integrators need to find completely new ways of quickly producing profitable solutions.

This calls for a re-thinking on the part of information and communications service providers on how they manage their business. It also calls for software developers to embrace a new way of developing management software. This is the impetus for the TM Forum's New Generation Operations Systems and Software (NGOSS) initiative. The NGOSS initiative aims to deliver a framework for rapid and flexible integration of Operation and Business Support Systems in telecommunications and throughout the wider communications industry. Key areas of the NGOSS initiative are:

- Definition of the Next Generation **Business Process Framework**
- Definition of a standard set of Business Processes for the Information and Communications Services Industry
- Definition of the **Systems Framework** upon which these business processes may be built
- **Practical implementations** and live demonstrations of these solutions
- Development of an industry compliance program to certify solutions and products for compliance to the NGOSS Specifications.
- Creation of a **resource base** of documentation, models, code and training materials to support the TM Forum members in their own development of NGOSS-compliant components

1.2 Goals and Objectives

This document will describe the major concepts and architectural details of the NGOSS architecture in a technologically neutral manner. This document does not prescribe a single new technology – rather, it allows for a federation of different technologies, each of which offers particular advantages at the business and system levels. In particular, it enables business concepts and principles to drive system design and architectures. This may be implemented using currently available distributed systems information technologies. Critical to this process is the sharing and reusing of common data and information.

The goal is to define a component-based, distributed systems architecture and an associated critical set of system services that this architecture requires.

Stakeholders will make different implementation decisions about the use of specific information technologies in the management products they build or procure. Therefore, this document describes the principles and system services that should be adopted in these new systems in a technology neutral form.

This document enables the mapping of specific information technologies onto the NGOSS technology neutral architecture. The specification of a technology neutral architecture enables different technologies to be used to build coordinated management systems that control one or more OSS components. It also enables the fast adoption of new information technologies as they emerge.

This initiative will provide an industry roadmap of common implementation principles to be adopted by all stakeholders in the value chain. The end objective is for the management systems integration of 'off the shelf' software to be accelerated, leading to reduced time to market. It is equally important to provide the ability to change and update installed NGOSS systems in accelerated timescales.

1.3 The Road to NGOSS

NGOSS is a set of guidelines and specifications for the industry to build software in a more structured and complementary way, based on industry experience and previous and ongoing TM Forum activities. The TM Forum projects, described below, have established a good basis for the NGOSS architecture. This enables it to be well defined and yet sufficiently flexible to cope with emerging and as yet undetermined technologies:

- The Application Component Team [ACT] project developed the concept of OSS building blocks that communicate via well-defined contracts[1]. It identified separate tiers for Human Interaction, Process Automation, and Enterprise Information Building Blocks. The ACT project has completed its work, having delivered TM Forum document GB909 Part 1, which defines a set of generic requirements for information building blocks. Its findings are being incorporated into the NGOSS architecture framework as appropriate. In particular, the architectural requirements for building blocks (which correspond to NGOSS Components) and contracts as well as requirements specific to Process Automation and Enterprise Information building blocks are being incorporated.

- The Application, Specialization and Integration of System Technologies (ASIST) team has been identifying candidate distributed systems technologies to support the architectural concepts presented by the ACT team in GB909. The ASIST work has been incorporated into the technology-specific work of NGOSS.

The following TM Forum projects are currently active, and are contributing to the specification of the technology-neutral architecture.

- The enhanced Telecom Operations Map™ [eTOM] presents a high-level overview of Telecommunications OSS business processes. This is described in TM Forum document GB921, and provides the analytical basis for business process integration across the OSS space. The eTOM builds on, and supercedes, the popular TM Forum Telecom Operations Map (TOM) document (GB910).

- The Shared Information/Data (SID) model can be viewed as a companion model to the eTOM, in that it provides an information/data reference model and a common information/data vocabulary from a business entity perspective. Teamed with the eTOM, the SID model provides enterprises with not only a process view of their business but also an entity view. That is to say, the SID provides the definition of the 'things' that are to be affected by the business processes defined in the eTOM. The SID and eTOM in combination offer a way to explain 'how' things are intended to fit together to meet a given business need.

[1] Contract as defined in [SZY99]: Specification attached to an interface that mutually binds the clients and providers (implementers) of that interface. Contracts can cover functional and non-functional aspects. Functional aspects include the syntax and semantics of the interface. Non-functional aspects include quality –of-service guarantees.

- The SID team also is chartered to work on a systems view. This would extend the current business definitions of the SID to include system aspects. This view will form the basis for technology-specific implementations. Viewed in this way, the SID provides an extensible information model for representing the business and system views of an NGOSS system, and provides a single model from which application-specific data models can be derived.

- The Catalyst programs produce working demonstrations of technology integration using the NGOSS specification according to TM Forum principles. These projects, based on leading OSS products, validate and demonstrate existing TM Forum work, as well as identify areas of particular difficulty for ongoing integration work. Previous Catalyst projects have developed practical solutions to common OSS integration problems, most notably in the areas of business process abstraction, the use of some common information models, and the use of common communications and directory services. The Catalyst program is complementary to, and migrating towards, NGOSS style development, with the findings of one activity incorporated into the other on an ongoing basis.

- The Red Team was chartered in May 2002 to further complete the overall NGOSS concept. The goals for this activity are to make the technology-neutral architecture consistent, coherent, complete and implementable. Through a number of subteams, the Red Team has worked to achieve these goals. This document, together with its collected annexes, captures the results of the Red Team efforts.

Each of these projects (other than the Red Team) is described in detail elsewhere (see references).

1.4 Program Implementation

The NGOSS program is a significantly complex set of activities. It has consequently been planned as a set of phased deliveries. The outputs of each phase are planned to be implemented in Catalyst projects and fed into subsequent phases, to ensure lessons learned from implementation are incorporated into future guidelines and specifications.

As the program progresses, it is expected that an increasingly comprehensive library of documentation, models, contracts and reference implementations will be built within the TM Forum and made available to the industry for implementation and incorporation into products. This will enable the faster delivery of NGOSS-compliant products, thus enabling the stakeholders to gain the benefits of a more homogeneous, yet flexible, OSS environment.

1.5 Phases of Delivery and Roadmap

NGOSS will be delivered to Industry through a number of Phased Releases. An overview of what is to be contained in each Phase of Release will be maintained in the NGOSS Roadmap. This is kept up to date as a living document on www.tmforum.org.

1.6 Stakeholders in this Document

This document is intended for use by a number of stakeholders, within which there are likely to be a number of different roles, or users of, the document. The stakeholders presented below are not intended to be all inclusive, and it is accepted that other uses for this document will be found.

These usage scenarios are based on the individual stakeholders who may use various parts of the NGOSS specification(s) for their own goals. These stakeholders are as follows:

- Service Providers (SP)
- System Integrators (SI)
- Independent Software Vendors (ISV)

- Network Equipment Providers (NEP)

Note: a detailed analysis of the business case benefits for and the requirements of each of the individual stakeholders shown above is contained in the following TM Forum documents

- TMF 051 (Business Case) [BIZ]
- TMF 052 (Requirements) [REQ]

A further discussion of the roles that workers within each of the NGOSS Stakeholders may perform can be found in section 3.0 of this document. Workers within these roles will find a number of different uses for this document, depending on the internal organization and processes of a stakeholder.

1.6.1 Document Use by a Service Provider

The NGOSS architecture will provide service providers with the basis for a flexible OSS solution that will be able to rapidly evolve to meet the future requirements and be able to manage multi-vendor, multi-technology networks and related services. The NGOSS architecture will enable service providers to choose the 'best fit' management components for their business and enable management capability to grow along with the business, while protecting its OSS investment.

With the emphasis moving towards managing the timeliness and quality of the services which the network delivers rather than blindly managing the elements of the network, the need to be able to aggregate information from across all of the management functions will become paramount. The NGOSS architecture will enable easier communication across multiple management components, thus enabling the easier sharing of data from the multiple sources. This document will provide the Service Provider with a fundamental understanding of the architecture upon which their OSS needs to be built. By understanding that architecture, the Service Provider can take full advantage of all the features and services that NGOSS based systems can offer, ultimately reducing operational expenditure to run its business and capital expenditure to enhance the offering to its customers.

1.6.2 Document Use by a Systems Integrator

This architecture document is of key interest to system integrators that want to build a new generation of operational and business systems. Integrating applications from the existing and previous generations of support systems typically characterized by closed, frequently monolithic architectures with multiple private copies of data and differing (often proprietary) technologies, meant forcing inter-application communications. The NGOSS architecture with its support of a Shared Information and Data (SID) model, open interfaces, contracts and a common systems framework will enable system integrators to concentrate on the delivery of a completely integrated business solution rather than the narrower focus of trying to get a few applications to communicate.

Additionally, the basic concepts of the NGOSS architecture will mean a change in the methodology used for integration. Being able to assume that the management components will share common data formats and framework services as defined in this architecture means that the system integrator can concentrate on the flow of the business process and the policies affecting the flow. This flow enables a better utilization of the network to meet the business objectives of the organization using it.

This document will provide system integrators with information needed to plan for the skills and knowledge needed to work with both ISVs and NEPs delivering NGOSS based solutions.

1.6.3 Document Use by an Independent Software Vendor

The NGOSS Architecture document will be of critical importance to ISVs that are interested in competing in the OSS marketplace of the new generation networks. Regardless of whether the ISV is interested in providing an implementation of basic platform or middleware services, operational management applications, or both - this

document will provide the blueprint for success. As the service providers move towards the deployment of the new generation networks, their demand for component-oriented management systems software, based on the principles and architecture of the NGOSS, will be the rule not the exception.

In the case of an ISV focused on the basic platform or middleware services (termed Framework Services in this document) the NGOSS Architecture document and the NGOSS Requirements document will provide the information that is typically developed over months, if not years. In the case of an ISV working in the management applications market, this document will provide the key to understanding and leveraging the systems services available to support them.

1.6.4 Document Use by a Network Equipment Provider

For the network equipment provider this document will provide insight and understanding of the management environment into which their equipment will be deployed. Having that insight will enable them to structure and build their own equipment management systems in a way that more easily integrates with the other stakeholders in the value chain. As ease of integration of operations management systems is increasingly a significant network equipment procurement factor, meeting the industry need in this area will become critical to equipment sales success.

It will also enable more ambitious NEPs to change the architecture of their network elements to enable them to be more easily managed. Finally, it provides insight into how a next generation network element can better provide next generation services[2].

1.7 Organization of this Document

This document is organized into multiple sections. Each section is briefly described below

- *Section 1, Introduction*

  This section provides an executive overview of NGOSS, the work leading up to this document and the roadmap for progressing NGOSS. It also summarizes the utility of the document to each of the four major stakeholders: service providers, system integrators, independent software vendors and network equipment providers.

- *Section 2, Architectural Requirements*

  This section, besides a general introduction on distributed architecture concepts, explains which architectural requirements guided and pushed the definition of the NGOSS Architecture and principles. It is these principles that are supported and realized by the architectural definitions described in later sections.

- *Section 3, Roles, Responsibilities and the Use of this Document*

  This section provides an overview of the technical roles that users of this document have when using the NGOSS architecture. It lists their needs in relation to this document, and others in the NGOSS document family.

- *Section 4, The NGOSS Technology-Neutral Architecture*

  This section defines in details the NGOSS Technology-Neutral Architecture.

- *Section 5, Realization of Principles and Requirements*

[2] This document defines the environment in which these next generation network elements will be deployed, and therefore indirectly serves as requirements for the design of next generation network elements.

This section describes how the architecture defined in the previous section realizes the principles and architectural requirements enumerated in section 2.0.

- *Section 6, References*

  This section contains a list of references used in the formulation of this document.

- *Section 7, Open Issues*

  This section contains a list of Issues and Assumptions that are seen as critical to the understanding of the contents of this document.

Additional information will be covered, relative to this document, through the creation and issuance of addenda. The current addenda are:

- TMF053a: Glossary
- TMF053b: Contracts
- TMF053c: Behavior and Control
  - TMF053m: Process Management
  - TMF053p: Policy Principles and Architecture
- TMF053d: The NGOSS Metamodel
- TMF053f: Distribution Transparency Services
- TMF053s: Security Principles and Architecture

2 Architectural Requirements

2.1 Basic Architectural Introduction

2.1.1 Distributed, Interface-Oriented Architecture

The basic architecture underlying NGOSS is that of a distributed, interface-oriented architecture (DIOA). Several of these DIOA's have been defined in the IT and Communications industries [RM-ODP, CORBA, DCOM, Java RMI, TINA-C, JINI]. NGOSS is an OSS-specific profile of the concepts defined in many of these DIOA's.

The fundamental technology-neutral entity in any DIOA is an interface. A running system constructed using such a DIOA consists of a number of runtime entities that together offer the functionality represented by a set of interfaces. Clients wishing to exploit this functionality bind to required interfaces and invoke operations over those bindings.

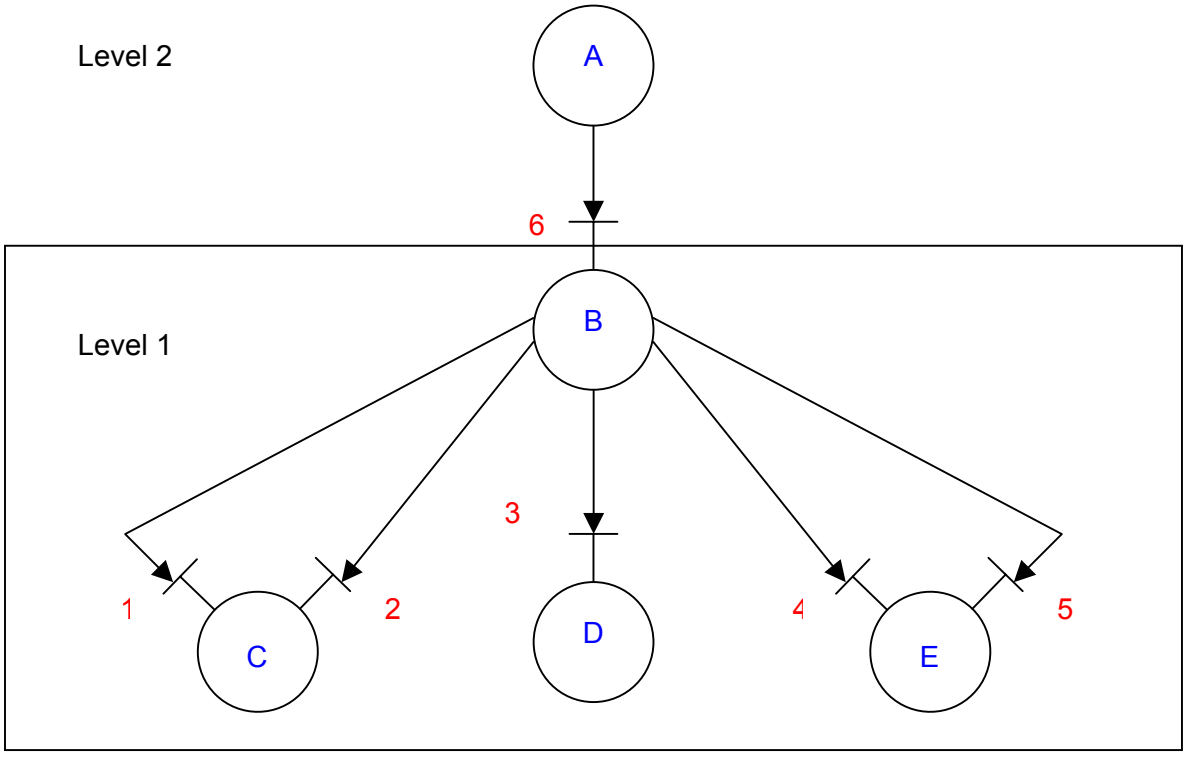

Figure 1: Example hierarchy of runtime entities conforming to a DIOA

A client of such functionality can itself offer one or more interfaces for other clients to bind to and exploit. This hierarchical nesting can go on *ad infinitum*. Figure 1 shows a two-level example of such a structure. In the figure, each circle represents a runtime entity that supports one or more interfaces. The interfaces are labeled 1 through 6. The client at Level 2 (A) uses interface 6, which is provided by the client at Level 1 (B), which uses interfaces 1 through 5 provided by C, D, and E.

A client must be "programmed" using some type of "programming language"; entities that support interfaces (such as C, D, and E in Figure 1) must also be "programmed" using some type of "programming language". System

management/OSS uses of a DIOA usually mandate that clients must be programmable using scripting languages (e.g. Perl, Python). In order for the hierarchical structure shown in Figure 1 to be achieved, clients like B must be programmed in a language that not only permits binding to interfaces 1 through 5, but must also permit the advertisement of interface 6 for consumption by other clients. NGOSS is a technology-neutral specification of a DIOA.

2.1.2 Components

The previous section referred to 'runtime entities'. Current best practice in the exploitation of DIOA's is to introduce a component model for the modeling and implementation of these 'runtime entities', and to use Component-Based Software Engineering techniques for the design, implementation, and deployment of solutions constructed using a DIOA [SEI00]. The remainder of this section (2.1.2) introduces the technical concepts that form the foundations for the Software Engineering Institute's Component-Based Software Engineering activities [SEI00].

Component-based software engineering is concerned with the **rapid assembly** (note that rapid assembly is just another way of saying plug-and play construction) of systems from components where

- components and frameworks have **certified properties**; and
- these certified properties provide the basis for **predicting the properties of systems** built from components.

A software **component** merges two distinct perspectives: component as an implementation and component as an architectural abstraction; components are therefore architectural implementations. Viewed as implementations, components can be deployed and assembled into larger (sub)systems. Viewed as architectural abstractions, components express design rules that impose a standard coordination model on all components. These design rules take the form of a **component model**, or a set of standards and conventions to which components must conform.

A component is:

- an opaque implementation of functionality
- a unit of deployment that is subject to third-party composition
- conformant with a component model

The ability to integrate components and to develop a market of components depends fundamentally on the notion of component **interface**. An interface is a coherent set of functional capabilities (both attributes and operations) of a component that users/clients of those capabilities can rely on.

The pattern of interaction among different roles, and the reciprocal obligations of components that fill these roles must be specified some way (see later the concept of "contract"). A **role** is an expected behavior pattern of a actor[3] in an interaction; examples include constraints, associations, state transition diagrams, etc.

A **component model** specifies the design rules that must be obeyed by components. These design rules improve composability by removing a variety of sources of interface mismatch. The rules ensure that system-wide attributes and behavior are achieved, and that components may be easily deployed into development and runtime environments. A **component framework** provides the services and mechanisms to support and enforce a component model.

[3] the precise meaning of "actor" here is the "participant" as it is defined in the SID

A component model defines not only how components can be composed into larger assemblies (which are also components); it can also define the external visibility of the component interfaces within an assembly outside of the assembly. In order to enable the maximum scope for use, the component model should enable the component composer to tune the visibility of such internal interfaces.

2.2 Separation of Technology-Neutral and Technology-Specific Architectures

One of the major guidelines in the definition of the NGOSS Architecture is the clear separation between the concepts valid independently of any specific implementation technology, namely "Technology-Neutral Architecture – TNA", and the concepts specific to one or more particular technologies, namely "Technology-Specific Architecture(s) – TSA".

Therefore all functionalities provided in a TNA Architecture, whether they concern Business Services or Framework Services, are documented in a specification that is neutral with respect to any specific implementation technology. There will be a separate set of documents produced, in the Technology-Specific part of the NGOSS program, which will describe the mapping of the Technology-Neutral Architecture and specifications of services to specific technologies for the purposes of implementing and deploying an NGOSS system [CORBAAppNote, SOAPAppNote].

These technology-specific mappings are expected to leverage industry standard frameworks (e.g., frameworks that support distributed computing, component-based architectures, etc.). Hence NGOSS deployments will benefit from the efforts to develop said frameworks.

2.3 NGOSS Technology-Neutral Requirements

There are a number of requirements that have driven the conception of the NGOSS architecture and represent industry best practice. The NGOSS integrates these principles into a coherent architecture for the development of management systems. This section of the document serves to introduce these fundamental architectural drivers; discussion of how these requirements are matched by NGOSS principles are described in Section 5 of this document.

2.3.1 Interface definition

An NGOSS system must be characterized by the fact that each software entity that provide Services to other entities does so through an interface defined in a way that the corresponding Service is specified with the following characteristics:

- *A description of the Service to be provided in terms of:*
  - The metadata used to describe the interface
  - The metadata used to describe the operations that may be invoked on the Service
  - For each operation, the set of terminations[4] that may be returned by the Service after invocation of the operation
- *The behavior of the Service: some of the behavior that **may** be specified are:*
  - The pre-conditions under which an operation may be invoked (i.e., the set of conditions that must be satisfied in order to invoke the operation)

[4] An operation invocation can return one of a set of results (each potential result is called a termination).

  - The post-conditions, which define the state that the system is left in for each termination that can be returned when an operation is invoked

- *The service must be independently manageable.*

Services are generally divided into three types:

- Basic Framework Services: these are Services needed to support any application domain built upon the chosen NGOSS Component model over the chosen distributed interface oriented architecture (e.g. Registration, Naming and Service Location).

- OSS Framework Services: these are Services needed to support any type of distributed OSS on the chosen NGOSS Component model/DIOA (e.g. Process Management).

- OSS Application Services: these are Services specific to a particular ISV or service provider's environment (e.g. Fault Management, Billing).

Note that the automation of a service provider's business process(es) may require the orchestration of all three types of Services. The NGOSS architecture is prescriptive about the basic framework Services. The TMF may over time choose to be prescriptive about one or more OSS framework Services. OSS ISV's will most certainly define vendor-specific OSS application Services in their product offerings.

2.3.2 Technology-Neutral Component Model[5]

The NGOSS architecture must be a component-based architecture. From the minimalist point of view, components offer service capabilities. However, to fully understand the implications of component-based systems, the minimalist point of view is insufficient.

The Carnegie-Melon Software Engineering Institute (CM SEI) defines a component as

- A binary[6] implementation of functionality,
- Contractually specified,
- A unit of independent deployment,
- Subject to 3rd party composition,
- Conformant to a component model

As a binary implementation of functionality, a component is an artifact. To be more specific, it is one of the artifacts of the software development process. In order to be useful, the services provided by a component must be clearly specified.

An NGOSS Component must:

- have all of its external (contextual) dependencies explicitly defined;
- be implemented in compliance to a technology-specific component model, such that a technology-specific component execution environment is capable of making the NGOSS Component services available at runtime to the remainder of the system[7].

[5] Throughout the remainder of the document, the term **component** (with lower-case 'c') is used for components consistent with the SEI definition, while the term **Component** (with upper-case 'C') is used for an NGOSS technology-neutral Component.
[6] i.e. "atomic"

- either not have persistent state [SZY99] or must manage its persistency according to the business requirements of the system (e.g. high availability and/or hot failover). This applies to an NGOSS Component, not to an interface or service that will of course create, manipulate, and rely upon information that is stored in a persistent fashion within the system. The reason for this requirement is that in a high availability environment, the possibility of redundant Component installations must be accounted for. An NGOSS *Component instance* is the installation of a particular Component within the system in such a manner that this particular Component can be instantiated by the component execution environment. If a Component instance has persistent state associated with it that is required by service instances, and that state information becomes unreachable, for whatever reason, the ability to use a redundant Component instance is compromised. A Component instance may have local configuration information, and information related to the management of the Component instance (license key, local directory server name, etc), but may not maintain information at the Component instance level to be used by service instances.

Given the defined services and the explicitly defined external (contextual) dependencies, an NGOSS Component can be combined in a system with other NGOSS Components to deliver an aggregated service that is then available to all other NGOSS Components (and client applications) within the system.

For instance, in a Java™ based system, the component is embodied by the .JAR[8] file. The .JAR file contains other files (e.g., .class files) that are the compiled (i.e., Java™ virtual machine byte-code) binary implementation of the technology-specific components, each of which contain contractually specified functionality. The .JAR file may be installed into a system and used, provided that any external contextual dependencies are met. The Java™ 2 Standard Edition platform does not, in and of itself, provide support for the specification of these external contextual dependencies, other than as textual documentation. Installation-time support for the specification of external context dependencies is provided through the chosen Java™ 2 Component Model. One implementation of this is Enterprise Java™ Beans (EJB), specifically via the EJB deployment descriptor.

The NGOSS Technology Neutral Architecture makes the following presumptions in regards to NGOSS Components:

- The NGOSS architecture must not define which interfaces and services are contained in a particular Component.

- The NGOSS architecture, however, must define how Components operate in general, as well as how a Component installs and makes its services available.

The reason behind the choice of a Component-based development relies in the advantages over the existing, traditional approach of monolithic system development. These benefits include, but are not limited to:

1. Supply of components from multiple sources.

   This supports best value for money in procurement of components and enables financial considerations to be taken into account for make / buy decisions. It also helps prevent vendor lockout as well as lock in.

2. Re-use of components.

   Business components may be re-used across multiple business scenarios. In addition, new business scenarios may be viewed as 'delta' changes to existing solutions. The financial implications for this are

---

[7] The issue of conformance to a component model is seen as being a technology-specific issue and will not be discussed further within this document.
[8] Java™ Archive File

centered upon the need for only marginal additional costs when components have to be extended or replaced to support the new business scenarios.

3. Legacy / heritage systems.

   It may be necessary to bring legacy non-NGOSS systems to an NGOSS environment for business /financial reasons. This can be achieved by creating *mediation interfaces* between legacy systems and the NGOSS components.

4. Flexibility of the systems environment and associated speed to market for new service products.

   Since new service products will be delivered through linking together of components, this more flexible environment will allow a much more diverse range of service products to be provided to customers. In addition, the NGOSS approach shall enable them to be provided in a much faster time.

2.3.3 Separation of Business Process from Component Implementation

An NGOSS system must be characterized by the separation of the hard coded behavior of Components from the software that automates business processes across the Components – i.e. an NGOSS system should be composed of defined services that can be orchestrated using scripting/process management technologies. An NGOSS system is further characterized by externalized descriptions of behavior expressed in a technology-neutral manner.

Examples of this are:

- *The external description of a Service is separated from one or more specific implementations through a representation called a Contract. Contracts enable different aspects of the business and system views of an application to be integrated through the use of business and system models.*
- *A business process is represented by an executable meta-language description, which describes the flow of execution between Component instances.*
- *System behavior is controlled through the use of different types of management paradigms*

A business process model may invoke lower-level business process models. This means that a business process step that a service provider desires to automate (e.g., verifying that the network can support the provisioning of a desired service) may be made up of one or more lower-level interactions with different system runtime entities that provide the necessary services; lower-level business process models used in this way must be able to provide one or more Contract instances to which the higher-level business process model can bind. That means that *multi-level* process management can be supported.

Process management is the application of modern business management techniques to business processes. This includes techniques for defining, measuring, analyzing, testing and deploying business processes as well as executing them. All of these activities form a part of business management, and so must contribute to the goal of improving business results for telecommunications service providers.

In pre-NGOSS systems, parts of business process functionality are embedded in separate business Components that lack the control interfaces to allow proper management of the business processes. This fragmentation of the business process renders many process management techniques impossible or impractical. The separation of the business process from the Components allows process management techniques to be applied; for example, most business process models are in fact sequenced calls to business service Contracts.

2.3.4 Security-Enabled Architecture

The customers of telephony services (voice and data) are demanding innovative solutions that can be deployed quickly, are highly configurable and provide heterogeneous support for multiple vendors and multiple technologies. An NGOSS system is required to manage these customer demands and the new generation of networks. The TMF has taken up the challenge to design an infrastructure that will meet the industry need. As part of that initiative, it is important that the solutions provide a security infrastructure to meet the demands for security.

The heterogeneous nature of the current generation of telephony services means that customers will have their service provided by a mix of companies and departments. In this environment companies must automate the processes that provide those services. Inadequate security in this environment could quickly escalate into major revenue loss as vulnerabilities are exploited by automatic systems.

An NGOSS system should be designed according to an overarching security model. An implementation of an NGOSS system will require the setup and operation of one or more security mechanisms and policies in order to operate the NGOSS system in a secure manner. To this end we have structured the security provisions using the structure of the ISO 17799 Information Security Management standards. This provides a framework to manage and operate an NGOSS system to meet the security objectives of an operating company.

2.3.5 Policy-Enabled Architecture

The NGOSS system architecture can optionally use policy management. The term policy-enabled is defined as a system that operates using policies to make present and future decisions. In other words, if a system is policy-enabled, then the operation and management of that system is dependent on the execution of policies. Stated more generally, policies provide rules that govern behavior within a system. Policy rules are defined to meet business, system, or other objectives; consequently, policies may link one or more of the business, system (operational), and implementation views of the NGOSS system to each other. Please see [053C] for more information.

A Policy Service has three main uses. First, it may be called upon to validate something using a specified policy (such as a user's access privileges, as specified by a security policy). Second, it may be used to determine what policies apply to a given set of conditions. Finally, it may be called upon to determine if there is a conflict between two or more policies that are to be executed.

The policy-enabled requirement mandates that the basic architecture indicate those points in the execution stream at which policy information will be consulted if policy is enabled in a particular OSS, and how those policies will be executed.

2.3.6 A Shared Information and Data Environment

An NGOSS system must be characterized by the usage of a common information model for enabling integration and interoperability. The information model is designed to be more than just a standard representation of data – it also defines semantics and behavior of, and interaction between, managed entities. This set of information, all provided in a standard representation using standard data types, is used to describe domain information (e.g., orders, network service and configuration definitions) in an NGOSS system.

2.3.6.1 Data Stewardship

An NGOSS system, to be distributed, must also be characterized by the stewardship of distributed enterprise data. Stewarded data has two fundamental characteristics that differentiate it from non-stewarded data.

- *There is a clearly identified "steward", that is an official repository of record, which is responsible for maintaining the data. This is not meant to imply that NGOSS must use a single and/or centralized repository.*
- *There is a clearly identified "owner", who controls who can make changes to the data when and how. This does not preclude the existence of shared data that are stewarded by multiple "owners" (e.g., a multi-master directory).*

A data steward also supports a control mechanism, so that a service can be informed of changes to critical, shared data.

The architectural term for providing stewarded data is *information provider*.

2.3.6.2 Information Adaptation and Mediation

While the NGOSS architecture describes a neutral and uniform framework of Services, Components, and a Shared Information and Data model, real-world deployments may not represent or be able to implement this ideal. For this reason, the NGOSS architecture must include the concepts of "adaptation" and "mediation".

Mediation is the process of translating from one level of representation and/or abstraction to another level of representation and/or abstraction. Often, mediation of two entities is achieved by translating the two entities to a common third form (e.g., the SID defined in GB922). Mediation enables entities to interoperate in a loosely coupled way.

Adaptation is a mechanism to convert one entity that has a particular syntactic representation to another entity that has a different syntactic representation, in order to enable interoperation of the two. Effective interoperation may still require the application of a mediation layer, since adaptation only addresses syntactic differences, not semantic differences.

2.3.7 Distribution Transparency

In the following sections 2.3.7.x requirements to support distribution transparency are presented.

2.3.7.1 A Repository of Runtime Information

An NGOSS system must be characterized by the existence of a *repository* that records information used during system execution, such as the following:

- *The Naming Service enables logical names and/or aliases to be assigned to a manageable entity that renders a Service*
- *The Naming Service then enables the subsequent access to that manageable entity by its names and/or aliases*
- *The Naming Service provides the ability to bind attributes to a named entity*
- *The Service Location Services enables access, selection and location without knowing the precise name of the object that provides the service (i.e. the attributes are used as selection parameters)*
- *The location and state of all management entities in the environment (e.g., people and/or processes that can manage certain devices and services)*
- *The location and state of "models", which are externalized descriptions of content and/or behavior, as described below. Two important types of models are Contracts and Policies, each of which is defined further in sections 5.3 (Contracts) and 5.8 (Policies).*
- *Information that supports the use of any service or policies, or the set of user instance and role instance information that is used by any defined Policies.*

The existence of a single logical repository is assumed, regardless of whether it is physically implemented as a single or a distributed system.

Note that the repository must store technology-neutral artifacts as well as technology-specific artifacts. This document is only concerned with its use of technology-neutral concepts.

2.3.7.2 Common Communication Mechanism

An NGOSS system must be characterized by the existence of a communication mechanism (e.g., a messaging bus) or some other form of common communication. All software entities will use one or more communication mechanisms to communicate with each other. Each communication mechanism offers one or more different transport mechanisms. There may be more than one such communication mechanism within a given system implementation, and these mechanisms may represent different technology-specific mappings. The common communication mechanism must provide transport mechanisms consistent with the basic interaction styles defined in the architecture and any security policies that a service provider has defined for his/her network.

More importantly, the use of a common communications mechanism enables the standardization of system-wide operations, messages, or events that can be distributed to interested components. This is an important point, as a transport "just" carries information, and by itself doesn't provide interoperability.

2.3.7.3 Invocation Mechanism

There are several steps associated with invoking an operation on a service instance: location of the instance providing the service, proper usage of the communication mechanism, results handling, etc. Security and other policies must be applied at each step in the process. For this reason, Invocation mechanisms must be defined as a convenient way to perform all of these steps and ensure the integrity of the operation in the context of the overall system and the client invoking the service. This mechanisms are usually technology specific and therefore not covered in this document.

2.3.8 Compliance and Certification

While not strictly core principles of the NGOSS architecture, compliance and certification ensure that an application claiming compliance to NGOSS implements and complies with the fundamental principles of NGOSS. For the NGOSS architecture to be ubiquitously deployed, it must be possible to measure the degree of compliance that any given implementation of the NGOSS architecture exhibits. The particular services that are deployed within an NGOSS based system are not the measure of its compliance. Instead, compliance is measured by verifying if an implementation has implemented the core principles of an NGOSS architecture, as described in this document. The following is a listing of the most important of these principles.

- *Architecture is inherently distributed (2.1.1)*
- *Architecture uses interfaces to communicate (2.1.1)*
- *Architecture is componentized (2.1.2)*
- *Business processes are separated from Component Implementation (2.3.3)*
- *Architecture is security-enabled (2.3.4)*
- *Architecture may be policy-enabled (2.3.5)*
- *Architecture uses shared information and data (2.3.6)*
- *Architecture uses a common repository (2.3.7.1)*

One of the most important mechanisms for realizing the above is the use of Contracts, as defined in section 4.1 and in [TMF053B]. The specific details of this are beyond the scope of this document, please refer to [TMF053B].

Briefly, a given system's compliance to NGOSS architectural principles is measured by its compliance to the above eight bullet points, not of the system functionality as a whole. Furthermore, information is accessed and exchanged using contracts.

A deployment of an NGOSS system need have only those component that contain implementations of the Services that are required to meet the needs of the system operator. Such a deployment represents a complete deployment of an NGOSS system. However, it is important to note that what appears to be a complete deployment for one system operator may in fact be a partial deployment (relative to needed Services) for a second system operator.

Certification requirements will be established by the TeleManagement Forum "NGOSS-Powered™" industry-wide Compliance Program steering team. These requirements are considered to be out of scope for this document; instead, they are specified in the TMF 050 document series [COMP].

2.4 Categorizations of NGOSS Systems

To cover all the architecture lifecycle aspects, an NGOSS system must be categorized along several different dimensions. No one categorization is best suited for all uses. The following sections describe three different categorizations that have been found useful.

2.4.1 TMF Views

The following text has been extracted from [DIM].

NGOSS consists of a framework of principles and procedures that can be used to guide the development of distributed computing solutions through the creation and re-use of artifacts retained in the Knowledge Base. The purpose of doing so is to improve business process automation through the integration and interoperation of commercial off the shelf software. The concept of the knowledge base is central to the methodology because it provides a way to "link" work efforts and activities as the progressive views of the system solution are developed (see Figure 2).

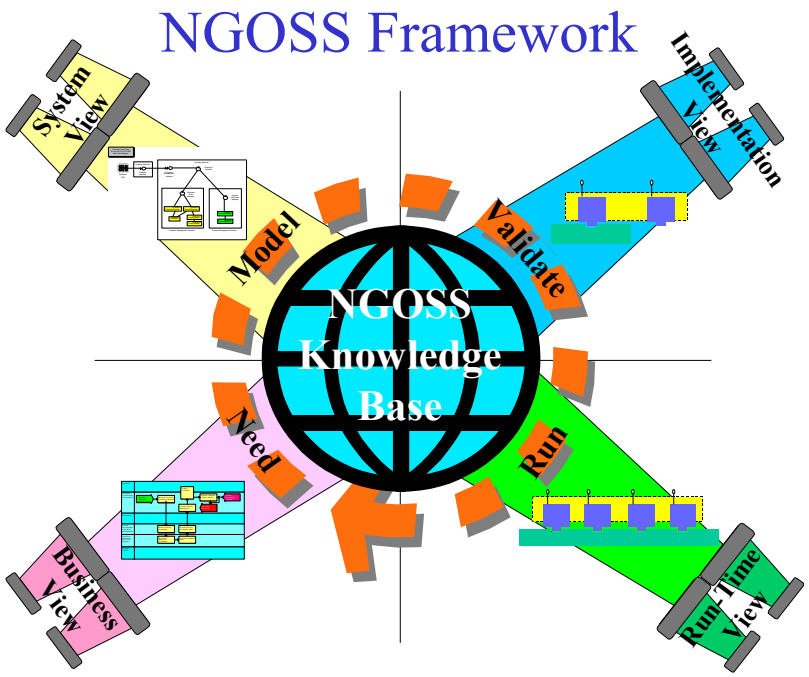

Figure 2: NGOSS Views

Moving in a clockwise direction, an operator's deployment teams apply the procedures defined in the selected method to identify the business need, model the solution, validate the model and perhaps build the run-time system solution. Each team will be staffed to fulfill the roles necessary to allow their understanding of the solution to be properly documented. Each team's view of the solution space will be rendered as a set of specifications created by drawing on the artifacts available from the Knowledge Base.

The artifacts retrieved from the Knowledge Base support building a model (problem, constraints and answer) of the proposed solution. The model is used as the basis for reconciling implementation and realization decisions once construction on the actual distributed system solution is underway.

The types of artifacts available for use in constructing the solution model are varied, but fall into three general categories: process, information and infrastructure:

- *Process Context – business process flows, system process plans and process realization scripts; contracts and reference code*
- *Information Context – business entities, shared data models, and realization data models*
- *Infrastructure – technology neutral and technology specific architectural frameworks*

The process and information contexts provide a way to focus on a particular dimension of the solution space. As can be seen from the artifacts listed above, the process context emphasizes the behavioral aspects of the solution space while the information context describes specific details regarding the factual aspects (i.e., the static data) of the solution space.

Thus, the purpose of this view is to relate the business, system, implementation, and runtime views of systems and Components to each other.

2.4.2 ODP Viewpoints

There are many different views that different stakeholders have of a distributed system. These have been canonized using RM-ODP viewpoint within ISO [RM-ODP]. IEEE 1471 defines "viewpoint" as a specification of the conventions for constructing and using a view. The primary viewpoints described therein are Enterprise, Information, Computational, Engineering, and Technology.

The RM-ODP standard has the following to say about viewpoints:

> This reference model defines a framework comprising:
>
> - *Five viewpoints, called enterprise, information, computational, engineering, and technology which provide a basis for the specification of ODP systems;*
> - *A viewpoint language for each viewpoint, defining concepts and rules for specifying ODP systems from the corresponding view-points;*
> - *Specifications of the functions required to support ODP systems; and*
> - *Transparency prescriptions showing how to use the ODP functions to achieve distribution transparency.*
>
> The architecture for ODP systems and the composition of functions is determined by the combination of the computational language, the engineering language, and the transparency prescriptions.

The RM-ODP standard succinctly describes each viewpoint as follows:

> Enterprise – a viewpoint on an ODP system and its environment that focuses on the purpose, scope and policies for that system.
>
> Information – a viewpoint on an ODP system and its environment that focuses on the semantics of information and information processing.
>
> Computational – a viewpoint on an ODP system and its environment which enables distribution through functional decomposition of the system into objects which interact at interfaces.
>
> Engineering – a viewpoint on an ODP system and its environment that focuses on the mechanisms and functions required to support distributed interaction between objects in the system.
>
> Technology – a viewpoint on an ODP system and its environment that focuses on the choice of technology in that system.

One should note that:

- *There is an almost 1-1 mapping of the Business View and the Enterprise viewpoint*
- *There is an almost 1-1 mapping of the Shared Information and Data aspect of NGOSS and the Information viewpoint*
- *The technology-neutral architecture encompasses the Computational and Engineering viewpoints*
- *Each technology-specific architecture is a mapping of the TNA onto a particular Technology viewpoint*

2.4.3 Stages of System Development

This section specifies requirements specifically related to the different stages of the system development lifecycle, dealing with the artifacts that are important during each stage and which of those artifacts are architectural. The section will focus on a product that satisfies a hypothetical TMF definition of an OSS billing system; it is necessary to develop and test the product components that together yield the billing system, to provide a way for a customer to install and deploy the product, and to give the customer a means by which the deployed product is activated (e.g. start it running). The product may also come with various tools that can be used while the product is running. The following subsections explore each of these more closely.

2.4.3.1 Design, Development & Integration

Presumably, the TMF has defined what it means to be a standard billing system; in particular, it has defined the shared information for billing records, the contracts that the product must support for obtaining, distributing, processing, etc. billing records, how policy can affect the generation/use of billing records, how this service must be secured (subject to the operator's security policy), and other important definition and usage information. In order to build the product, the developer must have access to the service specifications that are mandated by the TMF, as well as to the standard shared information specifications mandated by the TMF and exchanged over instances of these contract types. The developer must also know how these concepts are mapped onto a particular technology-specific architecture so that the Components that will provide the service can be developed. As part of the development process, the designer will decide how to group contract instances into Components in that particular technology-specific architecture; in particular, it is important to note that TMF will not mandate this grouping.

From an architectural perspective, at design and development time the developer must have access to service specifications and shared information definitions in the repository. Therefore these particular services must be available at design/development/integration time for all other Components.

Since type specifications must be available at all stages, the repository and the services used to access, manage, and modify shared information are particularly unique services in the architecture.

The above example assumed that the result of the designer’s and developer’s efforts was a deployable product. In many cases, however, a subsequent stage is introduced; that of the system integrator. The system integrator integrates applications and services produced by a number of design/development teams (including both new and legacy), includes value add or integration software as needed and delivers a deployable solution to service providers.

What is the result of a successful design and development stage? A deployable product that supports all requirements of the service that it was designed for and is available for use by service providers.

2.4.3.2 Deployment

The deployable product enables the deployment of technology-specific, and vendor-specific, Components that when activated, provide the service that the customer purchased. During deployment, a database will be populated with information about the product; this information will be very vendor and technology specific. This does not negate the fact that this information is important to the operator that has deployed this product.

The Components that make up the product conform to the contract specifications and shared information specifications to which it was built. Since these are architectural artifacts, the act of deployment should cause this information to be added to the repository. These specifications are the ONLY technology-neutral architectural artifacts that are involved in the deployment stage.

2.4.3.3 Activation

This stage encompasses activating the Components that make up the service, and deactivating the Components that make up the service for graceful termination. The net effect of activation is to reify the Components into technology-specific entities that are operational. As part of the activation of a Component, the Component will activate contract instances; when all Components in the product have completed activation, all contract instances that are mandated by the TMF technology-neutral specification exist and are accessible to perform the actions for which they were created.

Section 2.3.2 mandates that all contract instances must be registered. Furthermore, the service location service (SLS) must be used to locate contract instances. Note that the SLS only needs to exist during the activation stage of other services, since it has not been required in earlier stages. Registration of contract instances requires some amount of information to be provided - minimally, some indication of the contract type that it is an instance of, and a contract instance handle that can be used by a client to bind to that instance.

A very effective tool that architects often use when relating different views is the tool of abstraction; in this case, an informational view artifact (contract specification) can be represented by a unique name. For example, the actual representation of a contract specification is an information view artifact that needs to be stored in the repository. For runtime negotiations with regards to contract specification, one could pass around the actual representation; alternatively, more sanely, the contract specification is represented by an unique identifier. Therefore, the advertisement signature to the service location service probably has an identifier argument for the contract specification, rather than a BLOB for the full representation.

2.4.3.4 Operation

This stage is just the steady-state situation that the operator's services are running and providing the services for which they were purchased. Applications that want to use this service must be able to retrieve binding information for instances from the Service Location Service, be able to bind (subject to policy and security considerations) to these services, and then invoke operations on those instances subject to the signatures and invocation semantics.

One should not infer that stages are somehow directly related to ODP viewpoints. From the above discussion one can see that artifacts from the information, computational, and engineering viewpoints all find their ways into all of the stages. It is only in the OPERATION stage that the illusion of computational viewpoint, to the exclusion of all others, holds.

2.5 Models

The term “model” here refers to the external specification of the characteristics and behavior of managed entities. Two types of models must be specified in an NGOSS solution:

- An *information model* that is an abstraction and representation of the entities in a managed environment. This includes definition of their attributes, operations and relationships. It is independent of any specific type of repository, software usage, or access protocol.
- A *data model* that is a concrete implementation of an information model in terms appropriate to a specific type of repository that uses a specific access protocol or protocols. It includes data structures, operations, and rules that define how the data is stored, accessed and manipulated.

Models can take many forms, including the specification of class diagrams, metadata, business objects, contracts, processes, deployment information, and policy information. It is important to note that models are used to provide an extensible system, rather than hard-coding information explicitly into a set of software modules. The NGOSS architecture must be built around the principle of externalizing this information, and expressing it in a technology-neutral fashion ([MDA] provides a reference to Model Driven Architectures). Managed information defined in these models can be retrieved by using the browsing functionality provided by the Registration Services.

For consistency across an NGOSS solution, the representation of information within an NGOSS deployment will be accomplished using the NGOSS metamodel [053d] and SID [GB922].

UML will be used to model shared information and data. There are many different ways to express behavior. However, since the SID is based on UML, OCL (the Object Constraint Language) will be used for expressing computable expressions that affect managed entities. Implementing the OCL expressions is outside the scope of this document.

3 Roles, Responsibilities and the Use of this Document

3.1 Use of this Architecture by Architects, Designers and Implementers

This section identifies a set[9] of technical roles that users of this document have, and their associated responsibilities. This can be used in defining the *lifecycle of architectural concepts* as seen by these roles. Such a role/lifecycle model provides the reader with further insight into the underlying rationale of these TNA concepts, why these concepts are really needed and how they all fit together.

The NGOSS architecture represents a facility for constructing distributed systems that support business processes associated with managing information and communications services domains. It provides a technology-neutral way of describing the structure of these systems, composed from a series of Components making use of technology-specific implementations of basic services. This is similar to the organization of other distributed Component implementation models, such as CORBA or Enterprise Java Beans(EJB), and thus the responsibilities involved in the design and development of NGOSS systems is similar to the responsibilities described for other similar types of systems.

There are distinct roles associated with the design of the infrastructure, design of the Components, and design of the system, each with their own set of responsibilities. These are described in more detail below:

- A Business Process Planner: This responsibility focuses on the definition of a business process flow.
- A Business Process Analyst: This responsibility acts as an intermediary between the Business Process Planner and the Domain Analyst.[10]
- A Domain Analyst: This responsibility focuses on defining an Analysis Model for a scoped area of management related problems for which NGOSS business Components are sought.
- A Service Designer: This responsibility focuses on the design and publishing of Service Implementation that may be shared and reused by others
- A Component Builder: This responsibility focuses on the analysis, design, implementation, testing and release of component-based software. The output of this role is a set of Components that can be used by any NGOSS system.
- A Systems Builder/Integrator: This responsibility focuses on analyzing the business needs of a system and designing, implementing and testing a solution that meets those needs.[11]
- An Infrastructure Builder: This responsibility focuses on delivering platform systems that provide the Framework Services needed by Business Components.
- A System Administrator: This responsibility focuses on monitoring and managing a system containing Component software in order to ensure it operates within required operational parameters.

---

[9] Note that this set of roles is not exhaustive.

[10] The Business Process Analyst has an in-depth knowledge of the existing NGOSS deployment as well as understanding of the overall business operation into which the new process will fit.

[11] This role will aim to make maximum use of existing contract specification and component software. The system builder/integrator selects from the available set of components, not necessarily aware of the internal structure and implementation of said components. To the extent possible, the relevant characteristics of the component are described through the contract specifications associated with the component.

- A System Security Administrator: This responsibility administers the security policy and how that system grants and denies access and usage limits.

- A System User: This responsibility focuses on a set of business tasks that involve using services offered by a system containing component-based software. A system user is primarily concerned with the operation of the system, rather than how the system is constructed. [12]

- An NGOSS Component: This responsibility focuses on providing services via its contracts, for which it may need to interact with other Components.

Figure 3 graphically illustrates the static relationships between roles (depicted as UML actors) in the NGOSS Methodology:

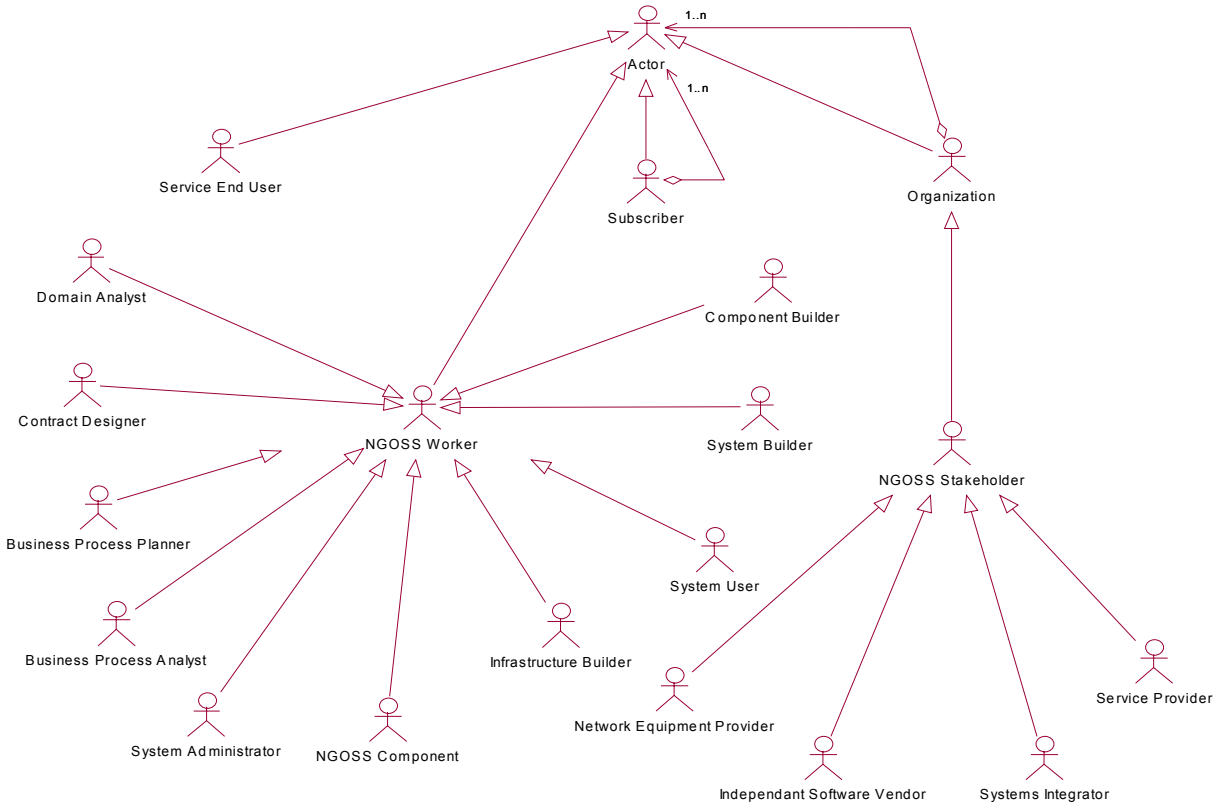

Figure 3: Relationships between Actors

3.2 Usage and Methodology

A *business process planner* defines a business process that is given to a *business process analyst* to develop requirements. These requirements are then given to a *domain analyst*, who with help from the business process

[12] There are two types of system users: end users and management users. The *end users* are concerned with getting the business results from the system, and the *management users* are concerned with maintaining the reliability and availability of the system to the end users. Note that a system user can be both an end user and a management user.

analyst, develops a domain analysis model. The domain analysis model is used to create the specifications for contracts and shared information and data objects. The activity of creating contract specifications and shared information and data specifications, is depicted in Figure 4.

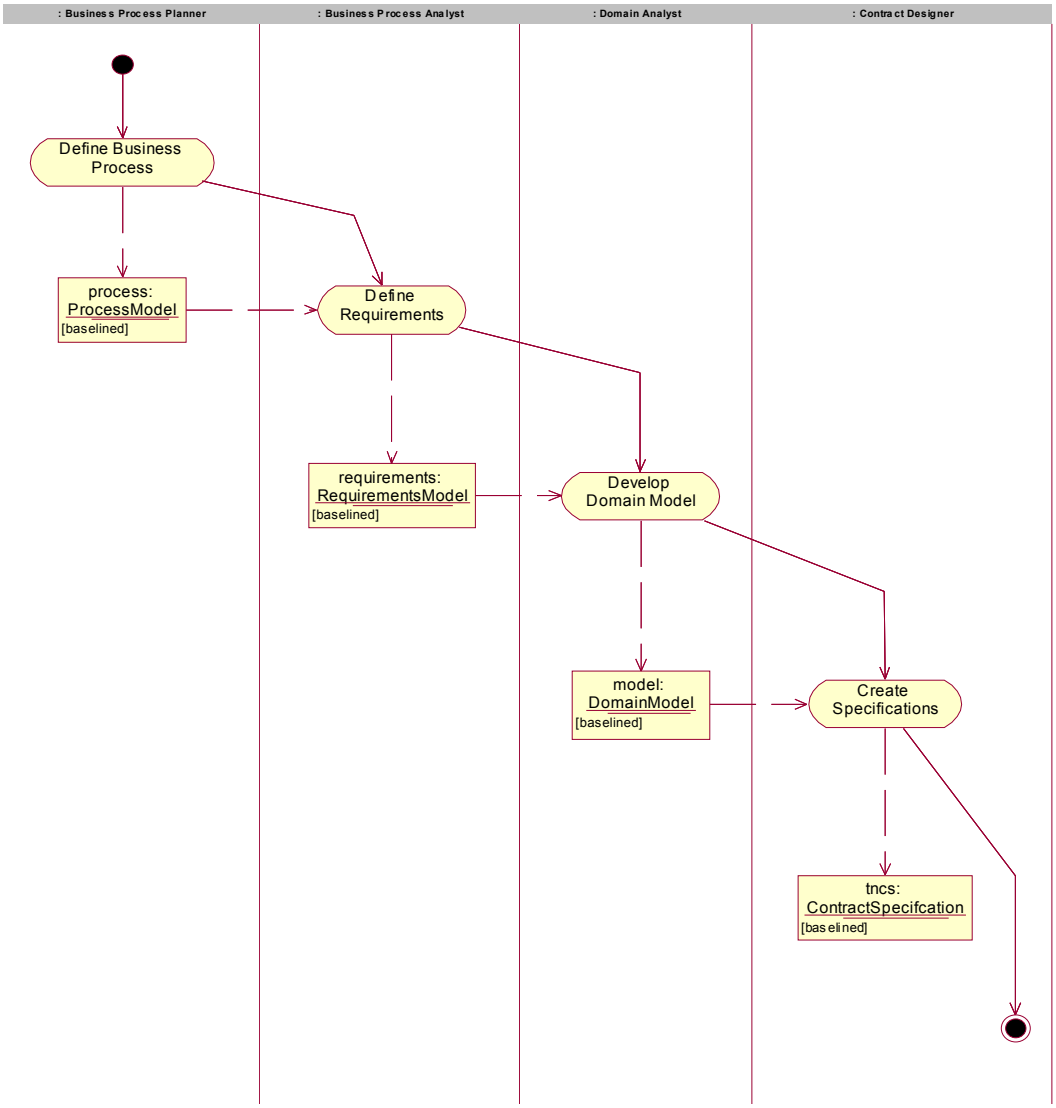

Figure 4: Creation of a Technology Neutral Contract Specification

The deliverable output of the *Create Specifications* activity is a *technology neutral contract specification* (other outputs, such as refinement of the information and data model specifications, are not shown in the above figure in order to keep it as conceptually simple as possible). The creation of the technology neutral contract specification is further decomposed in the following figure.

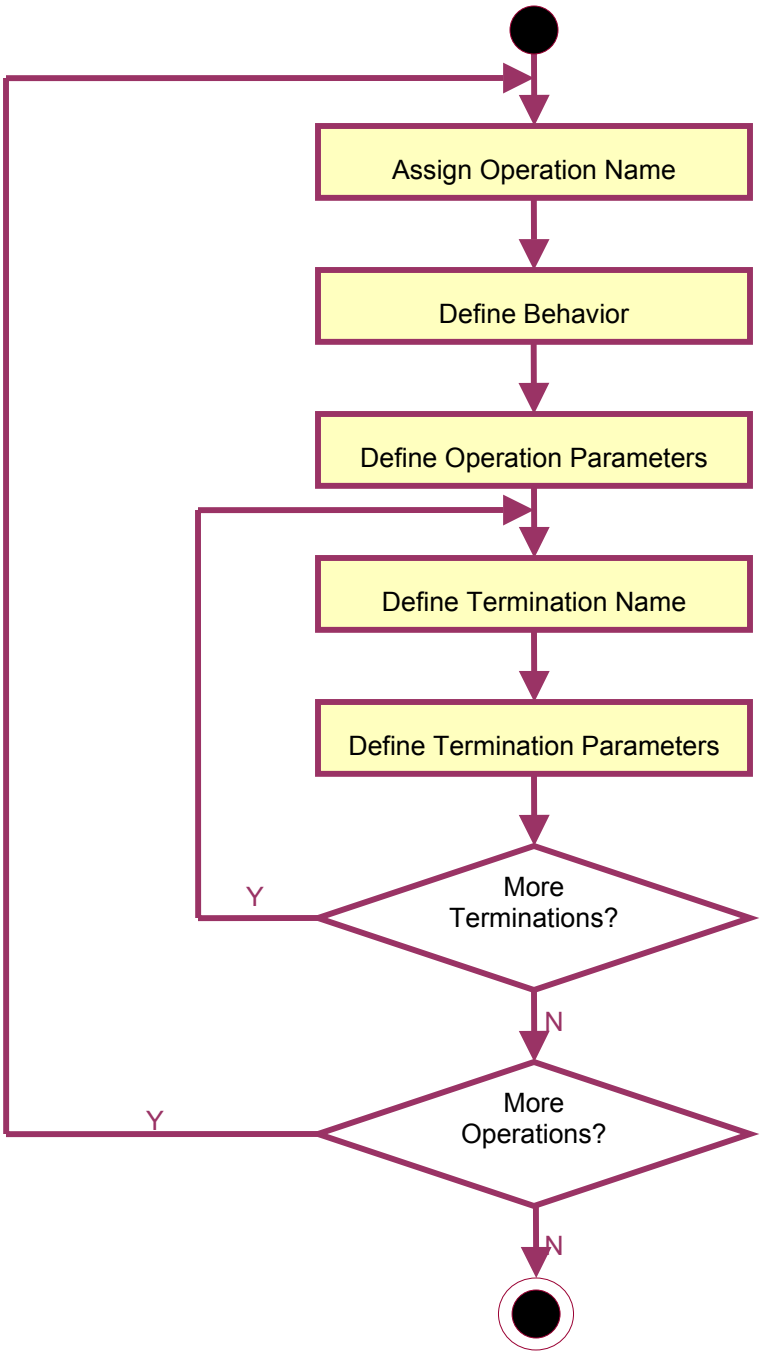

Figure 5: Decomposition of the creation of a contract-defined interface

Instances of shared information are termed *information objects*. Information objects are used to represent different aspects of a managed entity. These include definition of the basic characteristics of the managed entity (in the form of attributes), basic operations on that entity (in the form of methods), and how the entity interacts with other managed entities (in the form of relationships). In addition, the behavior of a managed entity can be modeled. In NGOSS, we abstract this behavior to a contract. Thus, the parameters (both request and response)

as well as any exceptions that may be generated during the execution of the contract implementation are all key elements that are modeled. This is done as part of developing the system analysis views utilizing the SID.

The definition of the shared information needed by a specific contract may (and is in fact encouraged to) reuse existing shared information definitions. Shared information definitions are contained within the SID. It is assumed that the reuse of shared information definitions will be accomplished by one of two mechanisms. These mechanisms are *by-reference* and *by-copy-and-extend.* In the former case, the definition of information contained within a specific shared information model is referenced by *name.* In the latter case, the definition of shared information is copied from an external shared information model, into a local shared information model, and then referenced by *name*.

Note that the SID was built to be inherently extensible. Each GB922 Addendum contains a section that explains how the model is developed, what assumptions were made in its development, use cases, and other information that enable the user to understand how the model was built. This facilitates extending the shared information by, for example, subclassing.

The actual definition of the Contract and the Shared Information associated with it is ultimately dependent on the existence of a number of models. The following figure depicts the static structure of the relationships between these models.

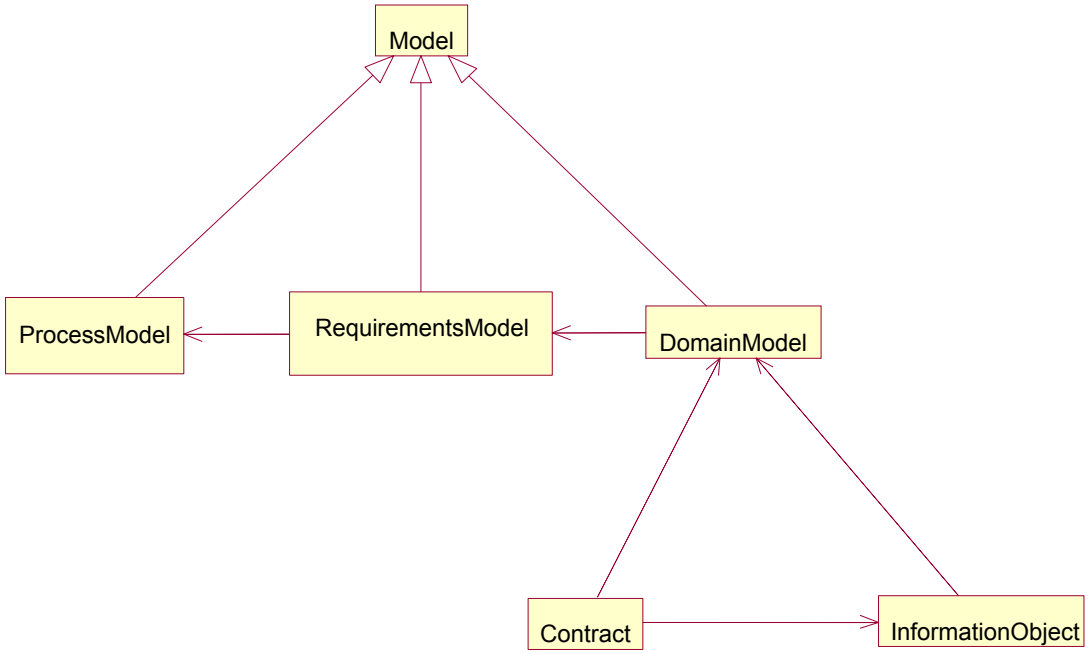

Figure 6: Static structure of relationships between Models and Contracts

The contract specification will be used in various manners, once the specification has been baselined. However, from an architectural perspective, discussion of such uses is considered to be out of scope, and therefore is not pursued within this document. Please see TMF053b for more information.

4 The NGOSS Technology-Neutral Architecture

This section describes the NGOSS technology-neutral architecture how it is has been designed from the requirements enumerated in Section 2. The text restricts itself to the computational viewpoint, based on the RM-ODP standard [RM-ODP] with one exception – Components (which are most similar to clusters in the RM-ODP engineering viewpoint) have computational aspects that are highlighted below.

The computational model comprises concepts, rules and structures for the specification of an NGOSS system from the computational viewpoint.

A computational specification defines the functional decomposition of an NGOSS system into Components that interact at interfaces.

In the computational viewpoint, applications consist of interacting computational Components.

In the remainder of the document, the term "Service" may be used to denote the functionality made available by a Component through an aggregation of Contracts. This is to be distinguished from the more colloquial use of the term service, like for example what is offered by a Service or Network Provider to a subscriber.

4.1 Concepts

This section provides unambiguous definitions for terms used throughout the remainder of this document (and its annexes) in the definition of the architecture. **The reader should beware of attributing colloquial semantics to these terms when reading the remainder of the document**; architectures require formal and unambiguous definitions of terms in order to remain complete and correct; attributing inappropriate semantics to one or more defined terms will usually destroy the correctness of the architecture. These definitions are also collected in TMF053a.

Interface: a coherent set of operations on a Component that clients of those operations can depend on using

Contract: the rules and constraints description for interaction via the Interface associated with the Contract; a Contract represents the unit of binding in the technology-neutral architecture;

Contract instance: a runtime manifestation of a contract-defined interface that deliver services to other runtime entities.

Service: a collection of functionality described by one or more contracts. The service represents the unit of manageability.

Component: an architectural element that represents the delivery mechanism for services. A Component represents the unit of deployment in the technology-neutral architecture and offers one or more services.

Operation: an interaction, through a contract invocation, between a client Component and a server Component which is either an interrogation or an announcement.

Announcement[13]: an interaction – the **invocation** – initiated by a client Component instance, resulting in the conveyance of information from that client Component instance to a server Component instance, requesting a function to be performed by that server Component instance.

Interrogation[14]: an interaction consisting of:

[13] Announcements have similar semantics to Notifications in other DIOAs.

[14] An interrogation is similar to an operation, in which the client requests that something be done, and the server responds with information that results from performing the request.

- The initial interaction – the **invocation** – initiated by a client Component instance, resulting in the conveyance of information from that client Component instance to a server Component instance, requesting a function to be performed by that server Component instance
- A second interaction – the **termination** – initiated by the server Component instance, resulting in the conveyance of information from the server Component instance to the client Component instance in response to the invocation

Interface signature: an interface signature comprises a set of announcement and interrogation signatures, as appropriate, one for each operation type in the interface.

Announcement signature: a definition of the name of the invocation and the number, names and types of its parameters.

Interrogation signature: a definition of the following elements:

- The name of the invocation;
- The number, names and types of the invocation parameters;
- A finite, non-empty set of definitions, one for each possible termination type of the invocation; each definition contains the name of the termination and the number, names and types of its parameters.

Binding mechanism: an infrastructure mechanism that enables a client Component to establish a binding to a server Component

4.1.1 NGOSS Component

The NGOSS Component[15] is an architectural element that supports one or more Contracts; a Component represents the unit of deployment in the technology-neutral architecture. NGOSS Components are therefore, from the minimalist point of view, containers of contract implementations. There is a component execution environment that, using contract implementations, creates contract instances.

The NGOSS focuses on the Contract Specification as the fundamental unit of functionality providing interfaces to NGOSS Components. The NGOSS Component is a container for at least two types of contracts:

- Contracts that represent the value-added service of the component
- Contract used to manage the service itself;

This distinction avoids dictating which contracts the developer of an individual component is required to implement. The individual component developer may implement those contracts that represent the functionality of their product, be it a self-contained service quality management component (which may be represented by single contract), or an end-to-end customer care system (which may be represented by multiple contracts).

An NGOSS Component must support at least one Contract; an NGOSS Component must also be manageable. Some NGOSS Components may achieve this manageability by supporting a standard management Contract in addition to any supported functional contracts.

An NGOSS Component can be further categorized into an NGOSS Framework Service Component and/or an NGOSS Business Service Component.

- An NGOSS Framework Service Components (FSC) implements one or more fundamental infrastructure services within the NGOSS that enable the features of the NGOSS architecture (such as plug and play and contract-trading) to be realized.

[15] a.k.a. Component

- An NGOSS Business Service Component (BSC) provides services that support the business-related functionality of the NGOSS architecture, such as billing, rating and discounting, network data management, and others.

4.1.2 TNA Concepts at Different Stages of System Development

Each concept defined in the previous section has a manifestation during different stages of system development. These are as follows:

- <concept> is the architectural element to be used in modeling
- <concept> specification is the formal description of the <concept>
- <concept> implementation is a technology-specific implementation of the <concept>
- <concept> instance is a runtime instance of the <concept> (if the <concept> manifests itself at runtime)

4.1.2.1 Architectural Elements

Each Contract is associated with exactly one Interface. Each Component deliveries one or more functional Contracts.

4.1.2.2 Specifications

Each Contract specification is associated with exactly one Interface specification. Each Component specification is associated with one or more Contract specifications.

4.1.2.3 Implementations

Each Contract implementation contains exactly one Interface implementation. Each Component implementation contains one or more Contract implementations, being functional and management ones.

4.1.2.4 Instances

Each Contract instance contains exactly one Interface instance. Each Component instance contains one or more Contract instances, being functional and management ones.

4.1.3 Architectural Views

The NGOSS architecture consists of elements and their relationships, and take the form of various diagrams. Control diagrams describe the interaction between NGOSS Components with respect to time.

The NGOSS architecture is model-based. That is, information in the NGOSS is represented via well-defined models that can interact and be shared with each other. Different aspects of a model can be shown using different diagrams.

There is more to NGOSS than an abstract architecture, however. A high degree of technology-independence is essential if NGOSS is to withstand the rapid pace of technical innovation in distributed systems technology. As well as placing constraints on the core architecture, this requires ongoing NGOSS activities to identify and map the technology neutral architecture onto existing and emerging technologies, so that NGOSS can continually leverage the best technologies over time.

The following sections describe the core architectural elements of the technology neutral architecture. These include NGOSS Contract Specifications, NGOSS Components, NGOSS Distribution Transparency, NGOSS Process Management Services, NGOSS Shared Information and Data Management Services, NGOSS Policy Management Service, and the NGOSS Adaptation and Mediation Services.

4.1.3.1 Structural View

Figure 7 shows the structure of the NGOSS architecture and the layering of services.

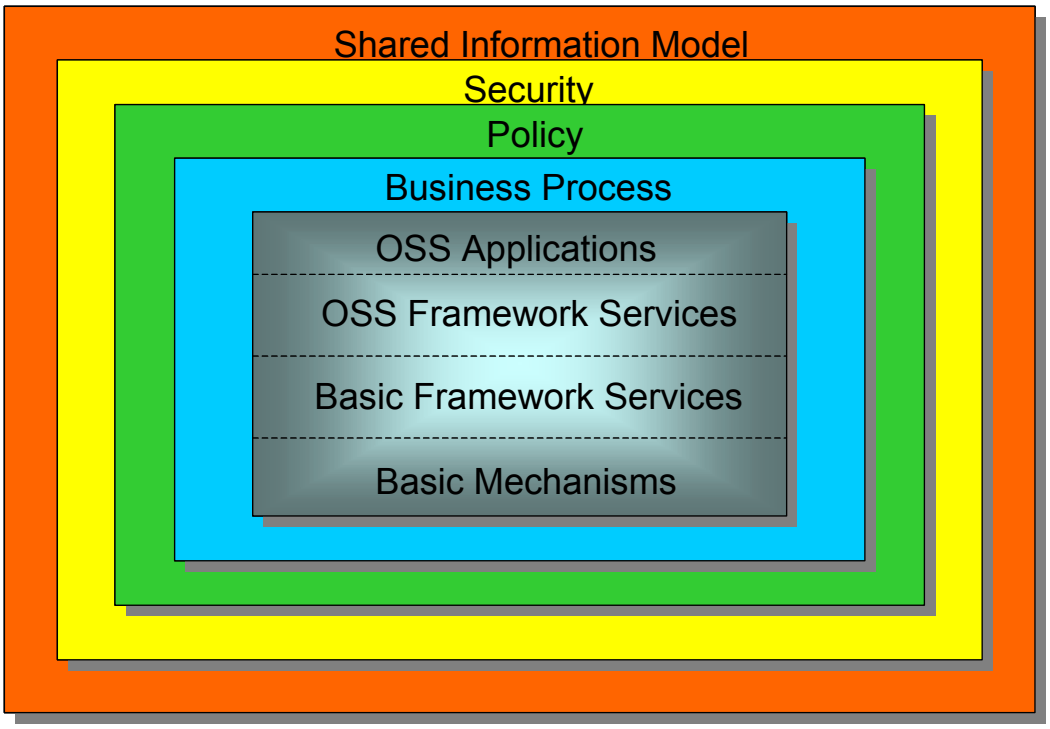

Figure 7: NGOSS architectural layering of services

The NGOSS architecture specifies two fundamental classes of Services:

- OSS Business Services – these provide elements of business functionality which are presented through Contract Specifications. Examples: Provisioning, Billing.
- Framework Services – these provide functionality to all the framework for correctly operating, are provided through appropriate contract specifications and can be further split into two sub classes:
  - OSS Framework Services – these provide standard OSS capabilities that can be orchestrated by Business Services. Example: logging.
  - Basic Framework Services – these provide the capabilities needed to support the patterns of interaction between Components implementing Business Services. Examples: Naming, Registration, Service Location.

Note the use of the Process Management, that *externalizes* process control, to integrate the Business Services, though Business Services can communicate without using a Process Manager. The actual Process Definition is being used by the Process Management Service, just as the Security Policy Definition and the Management Policy Definition are being used by their particular Services.

The Basic Framework Services may also be used by the other Services. For example, the Process Definition for Billing would actually be found by fetching it from the Repository.

The Framework Mechanisms and Services include communications, registration, lookup and service location, invocation, shared data management and policy management, among others. These have been drawn from the analysis of the support requirements of typical Business Services, and from the services typically provided in distributed system frameworks.

4.1.3.1.1 Artifact Aspect

From the TNA perspective, there are five tangible artifacts of interest. These are

- The Contract Specification: the technology neutral definition of functionality.
- The Contract Implementation: a technology specific manifestation of a contract (not covered in this document).
- The Contract Implementation Specification: basically a place holder for technology specific mapping definition of a contract (i.e. not covered in this document).
- The Contract Instance: the runtime manifestation of a Contract defined interface.
- The NGOSS Component: the unit of management for contract implementations.

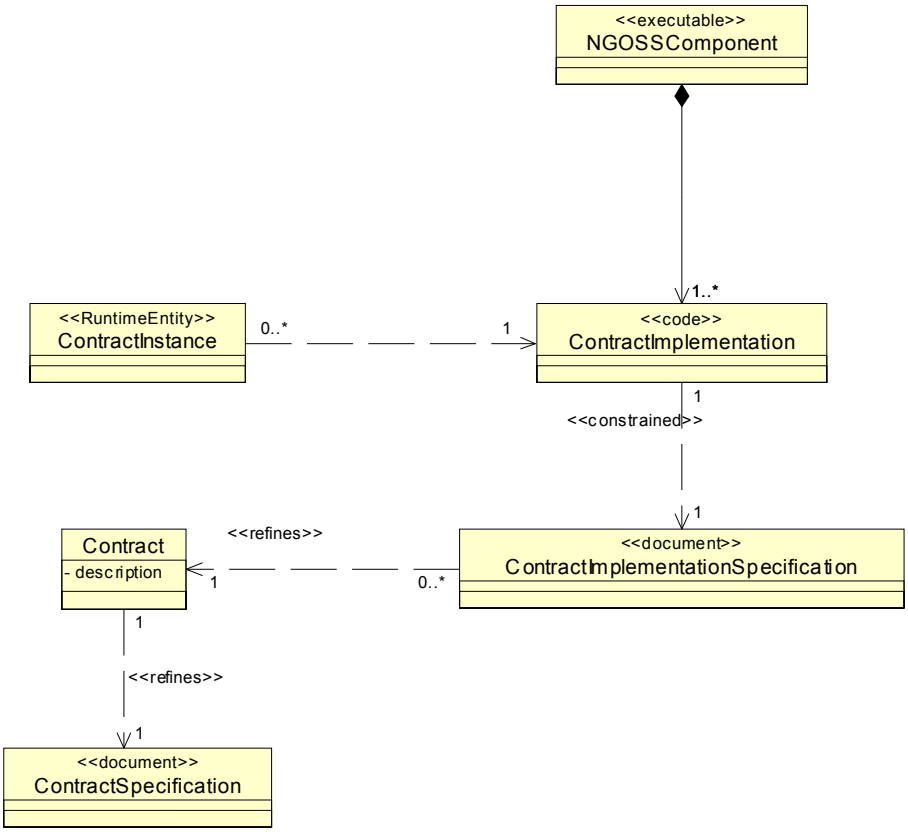

Figure 8: Tangible Artifacts with the NGOSS Technology Neutral Architecture

4.1.3.1.2 Registration and Repository Aspect

The NGOSS Registration Service provides for services and functionality needed to support location transparency in the system. The entities within an NGOSS System that are considered to be of interest from an architectural view are:

- The NGOSS System Repository,
- Entries that identify important Shared Information, which includes Contracts, Processes and Policies,
- Contract Registrations,
- Component Registrations, and
- Contract Instance Registrations

The NGOSS System Registration Service provides a maintenance interface (for the addition, modification, deletion and browsing of Components, Contracts and Contract Instances) to the NGOSS System Repository (assumed to be a single logical repository, possibly comprised of a federation of separate physical repositories). The information contained in the Repository includes instructions on where the entities specified above are located and how they can be accessed and/or invoked. The access and invocation of this information is done

through a Service Location Framework service; the direct access to the Repository browse functionality can be used by designer or administrators.

An important means of accessing shared NGOSS information uses the following principles:

- Technology neutral information is found by referencing the appropriate contracts
- Technology specific information is found by referencing the appropriate technology-specific contracts
- Contracts are a generic mechanism to find and/or invoke specific Components and to monitor and/or control Components.

The Repository also stores access information to managed entities in the shared information and data models that need to be used by multiple components. For example, common policy information that is shared across systems may be found and accessed using the Repository.

NGOSS Components will re-use the basic functionality of the Repository in an ad-hoc fashion. This ad-hoc reuse includes such operations as storing proprietary information.

The Repository is assumed to be of importance primarily for discovery operations. That is to say, once the definition of a piece of shared data has been found, and a binding to a contract implementation has occurred, the Repository should be bypassed mainly for performance reasons. This holds until such a time that the binding and/or information definition is discarded by the Registration Service or Service Location Service client and must be re-established.

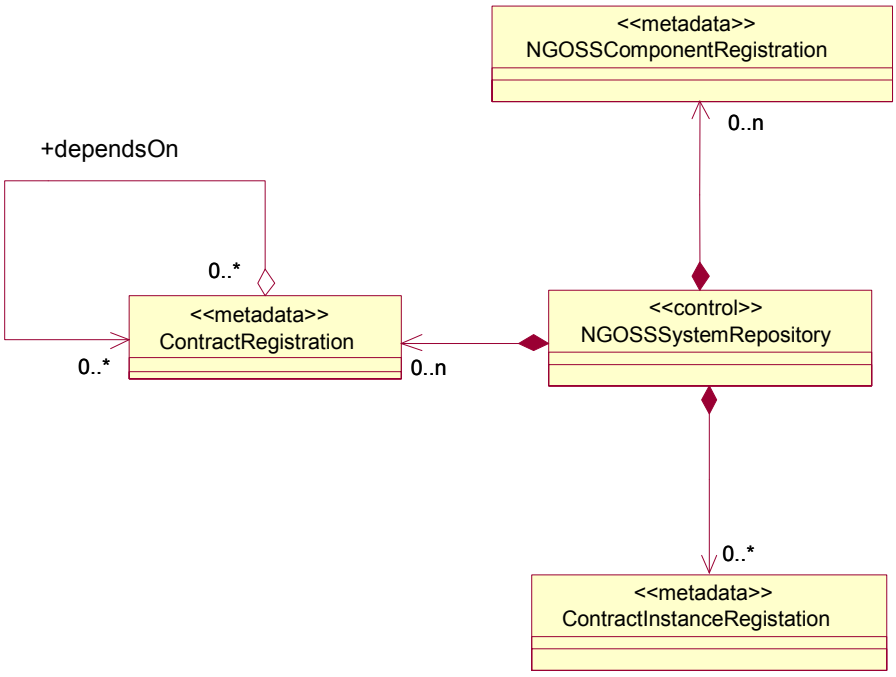

Figure 9: Architectural entities to support location transparency

4.1.3.1.3 Runtime Aspect

Finally, a basic understanding of the key entities from a runtime point of view must be established in order to understand the context in which services will be delivered. The following entities have been identified as being critical to the creation of a complete runtime view of the architecture.

- Component Execution Environment
- Contract Instance
- Contract Instance Descriptor
- Client

The Component Execution Environment is that portion of the system that provides the capability to create Contract Instances from the contract implementations contained within NGOSS Components. A Contract Instance can be thought of as being a process or task that is executing on a computing platform. The Contract Instance Descriptor (CID) represents a remote invocation handle that is used by a Client to request that a service be performed. By way of example, in a CORBA based NGOSS deployment, the CID is an IOR; in a Java JINI environment, it would be the client proxy reference.

4.1.3.2 Control View

Figure 10 shows a Control Diagram, indicating the typical sequence of interactions that occur between Components in an NGOSS system. The labeled arrows indicate the sequence of messages that flow as the Process Manager invokes a Business Service that in turn depends upon a Framework Service. Depending on the particular process flow implemented, a service in a business Component can be invoked either directly by the process manager or by another business Component.

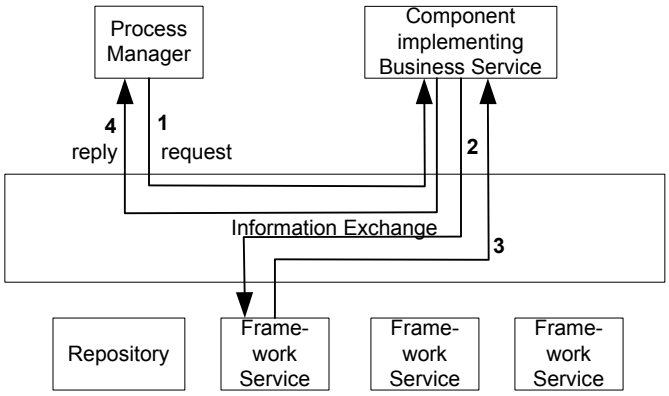

Figure 10: Control diagram illustrating invocation

This diagram shows that the services (and the Components that implement them) are decoupled via the Information Exchange, which is a communications mechanism. In order to invoke an operation, a Component creates a request and sends it via the communications mechanism. The response comes back in a similar way. It is important to note that there may be more than one communications mechanism in the NGOSS system.

Furthermore, the exact nature of the NGOSS communication mechanism is a technology-specific dependency. That is to say, in some technology-specific implementations, the NGOSS communication mechanism may be supplied by a specific identifiable runtime entity, whereas in others, it may be an endemic functionality provided transparently by the technology.

This control diagram simplifies a number of steps, which are shown in more detail in later diagrams, in order to emphasize the decoupling of services and their implementations. Later diagrams build on this figure, and add important concepts such as locating and requesting the target service, providing the proper transport semantics

of the request/reply interaction, error handling, and the role of the contract specification when the target service is invoked.

4.2 Structuring Rules

A computational specification describes the functional decomposition of an NGOSS system, in distribution transparent terms, as:

- A configuration of computational Components
- The interactions that occur among the Components

A computational specification is constrained by the rules of the computational model. These comprise:

- Interaction rules, binding rules and type rules that provide distribution transparent interworking
- Failure rules that apply to all computational Components and identify the potential points of failure in computational activities

A computational specification defines an initial set of computational Components and their behavior. The configuration will change as the computational Components:

- Instantiate further contract instances
- Perform binding actions
- Delete contract instances
- Create additional Component instances
- Destroy Component instances

4.2.1 Naming Rules

Each kind of name defined in the computational model has an associated context, as follows:

- An invocation name in an interface signature is an identifier in the context of that signature;
- A termination name in an interface signature is an identifier in the context of the operation definition in which it appears;
- The name of a parameter in an invocation definition in an interface signature is an identifier in the context of that invocation definition
- The name of a parameter in a termination definition in an interface signature is an identifier in the context of that termination definition

A computational interface identifier is unambiguous within its context.

4.2.2 Interaction Rules

Each interaction of a computational Component occurs at one of its computational interfaces. Interaction at an unbound interface fails. The binding rules impose constraints on how interfaces can be bound.

A client Component using an interface invokes the operations named in the interface's signature. A server Component offering an interface expects any of the operations named in the interface's signature. In the case of an interrogation, the server responds to the invocation by initiating any one of the terminations named for the operation in the interface signature. The client expects any of the terminations named for the operation in the interface signature. The duration of the operation (invocation + termination) is arbitrary.

The parameters for invocations and termination can include identifiers for computational interfaces and computational interface signature types.

A formal parameter that is an identifier for a computational interface is qualified by a computational interface signature type. The corresponding actual parameter must reference an interface with that interface signature type (or one of its subtypes). The actual parameter can only be used as if it referenced a computational interface with the same signature type as the formal parameter (or one of the formal parameter's supertypes).

4.2.3 Binding Rules

The NGOSS architecture supports primitive binding actions of either an explicit or implicit nature.

A primitive binding action binds two computational Components directly.

Explicit binding requires that the client Component invoke the binding mechanism in the infrastructure explicitly, providing an identifier for the target computational interface instance as a parameter to the binding mechanism. Upon a successful return from the invocation of the binding mechanism, the client Component is bound to that particular interface in the server Component.

If an invocation by a client Component references an operation interface to which the client is not bound, implicit binding is required. In addition to performing the explicit binding activity described above for the client, the runtime infrastructure may need to dynamically obtain code that can support the client side of the interaction.

Primitive binding either establishes a binding between the Components concerned, or fails. Deleting a Component that has been bound to another using a primitive binding action deletes the binding, as well.

4.2.4 Type Rules

Signature subtyping rules define minimum requirements for one interface to substitute for another. They are sufficient to ensure that a substituted interface can consistently interpret the structure of any interactions that occur.

Substitutability is used to support extensions of contract, for details see [TMF053B].

Interface ***Sub*** is a signature subtype of interface ***Super*** if the conditions below are met:

- For every interrogation in ***Super***, there is an interrogation signature in ***Sub*** (the corresponding signature in ***Sub***) which defines an interrogation with the same name;
- For each interrogation signature in ***Super***, the corresponding interrogation signature in ***Sub*** has the same number and names of parameters;
- For each interrogation signature in ***Super***, every parameter type is a subtype of the corresponding parameter type of the corresponding interrogation signature in ***Sub***;
- The set of termination names of an interrogation signature in ***Super*** contains the set of termination names of the corresponding interrogation signature in ***Sub***;
- For each interrogation signature in ***Super***, a given termination in the corresponding interrogation signature in ***Sub*** has the same number and names of result parameters in the termination of the same name in the interrogation signature in ***Super***;

- For each interrogation signature in **Super**, every result type associated with a given termination in the corresponding interrogation signature in **Sub** is a subtype of the result type (with the same name) in the termination with the same name in **Super**;
- For every announcement in **Super**, there is an announcement signature in **Sub** (the corresponding signature in **Sub**) which defines an announcement with the same name;
- For each announcement signature in **Super**, the corresponding announcement signature in **Sub** has the same number and names of parameters;
- For each announcement signature in **Super**, every parameter type is a subtype of the corresponding parameter type in the corresponding announcement signature in **Sub**.

4.2.5 Component Rules

A Component can:

- Initiate operation invocations;
- Respond to operation invocations;
- Initiate operation terminations;
- Respond to operation terminations;
- Instantiate contract-defined interfaces;
- Bind to contract-defined interfaces;
- Delete one or more of its contract-defined interfaces;
- Obtain a computational interface identifier for an instance of the registration service
- Obtain a computational interface identifier for an instance of the service location service.
- Instantiate an instance of the management contract

A Component must:

- have each contract instance registered in the Repository through the Registration Service
- locate any needed contract instances through the Service Location Service
- Abide by the provisions of those contracts to which it is bound and through which it is interacting

4.2.6 Failure Rules

The failure modes visible to an entity are determined by its behavior.

Any of the computational actions in the previous section can fail and that failure can be observed by the Component performing the action and/or some other external entity. Interaction can be disrupted by failure of the Components involved, or the binding between them, or both. Failure need not occur in all the participants, and may occur at different times and with different parameters for each failing participant.[16]

The failure of the server Component to respond to invocations or to initiate terminations can be observed by the client Component involved.

[16] Examples of interaction failure include security failure, communication failure and resource failure.

4.2.7 Shared Information Model

An NGOSS system is characterized by the usage of a common information model for enabling integration and interoperability. This model is called the ***Shared Information and Data (SID)*** *model*. The representation of the information is described in GB922 and its addenda, and consists of business and system views of information [SID]. This takes two forms: UML class diagrams and a data dictionary that defines the syntax, semantics and use of classes, attributes and other elements of the SID. This allows sharing and reuse of information by NGOSS systems through Contracts. The business view grounds the Contracts in fundamental requirements, and the system view provides Contract details (e.g., operations, terminations, behavior).

Contracts are not strictly part of the current SID, as they define the interfaces to Services made available by the OSS. However, it is mandatory that the data and metadata in Contract specifications will use information defined by the SID.

4.2.7.1 Data Stewardship

An NGOSS system is also characterized by the stewardship of distributed enterprise data. The existence of an official repository of record does not imply that NGOSS uses a single and/or centralized repository. NGOSS systems are likely to have multiple repositories, some of which may be distributed, in order to better effect communications. The prescription of having an owner for each piece of data does not preclude the existence of shared data that are stewarded by multiple "owners" (e.g., a multi-master directory) – in the case of such systems, there is still a clearly identified "owner" process that decides which process (of possibly many that are trying to write the same object with different values) will ultimately succeed in writing its value to the exclusion of all others.

A data steward also supports a control mechanism, so that a service can be informed of changes to critical, shared data.

5 Realization of Principles and Requirements

The purpose of this section is to show how the architecture defined in section 4 realizes the principles and requirements documented in section 2. In the cases where a particular principle is documented in an available addendum, the reader is referred to that annex for more details.

5.1 Contract-defined Interfaces

The nature of TNA contracts is fully described in annex TMF053b. The reader is referred to that document for more details.

5.2 The NGOSS Technology-Neutral Component Model

When an NGOSS Component is activated, it obtains a contract instance descriptor (CID) for an instance of the Registration Service contract and a CID for an instance of the Service Location Service (SLS) contract. (see Section 2.3.7.1); it uses these CIDs to bind to these services, if needed.

When an NGOSS Component instantiates a contract instance, the Component must register the contract instance through invocation of appropriate operations in the Registration Service contract to which it has created a binding. As the Component instantiates each additional contract instance (if any), it must register each instance through invocation of appropriate operations in the bound Registration Service contract.

If an NGOSS Component needs to bind to additional functional contract instances, it must first bind to the Service Location Service contract instance for which it possesses a CID. It then invokes appropriate operations in the bound SLS contract instance to obtain a CID for each required contract instance, and binds to those contract instances when it is necessary to invoke operations on those instances.

There is no assertion that the CIDs for the Registration Service and SLS instances obtained by an NGOSS Component are global – i.e. that all Components obtain the same CIDs. It is expected that a Component that is an assembly of other Components will provide localized instances of these services in order to prevent the advertised instances from the Components making up the assembly from being seen globally.

5.3 Separation of Business Process from Component Implementation

Annex TMF053C describes how process management and policy management work together to make the behavior of an NGOSS system more manageable. A more complete description of the process management features that an NGOSS system must support can be found in TMF053m. The following is to give an overview of the concepts contained therein.

The basic premise of the NGOSS architecture is that through the identification of well-defined, contract-specified interfaces, implemented by commercial off-the-shelf Components and supported by a distributed processing framework, it is possible to build a highly scalable, extremely flexible OSS that enables and facilitates the rapid deployment of new service products. The reality of the situation is that, without the ability to rapidly tailor OSS supplied services according to the business processes of the service provider, such a premise cannot become a reality.

The Process Management Service acts as a conductor or coordinator of activities spanning across the NGOSS Components implementing the Business Services. This Service provides the externalized process control that has been mandated by the NGOSS Service Provider stakeholder. The Process Management Framework Service ideally executes logic expressed using a means that is different from the implementation language used for the Component. This makes it easier to rearrange and/or alter the business process steps, and then have the Process Management Framework Service rearrange the interaction between the Components (if necessary).

There may be multiple such Process Management Services within an NGOSS environment, each one used at multiple levels of abstraction in implementing the control structure of a system.

For each NGOSS business process, Process Management can decide or can execute decisions on behalf of external systems that control which business contracts need to be invoked in what order and in what relation to each other. Each decision is based on a number of factors: the process plan, which includes policies, product definitions, and service level agreements; the outcome of the previous contract invocations; the states of the pertinent business and functional service Components; the availability of business contract instances of the next business contract type required for invocation.

Taking each of these decision factors in turn:

- The outcome of business contract instance invocations is communicated to the Process Management.
- The Process Management Service is responsible for registration and maintenance of available process definitions. The process definitions can be managed directly by the Process Management Service, or can be made available to the Process Management Service by external systems through contracts offered by other NGOSS Components.
- The Process Management Service gathers knowledge about the state of business and functional service Components through the usage of standard NGOSS contracts, where applicable.
- The Process Management Service may use service location capabilities as appropriate to determine the availability of business contract instances.

5.3.1 Process Definition

A *process* is a set of one or more linked activities that collectively realize a business or system goal [WFMC]. There are a variety of graphical notations and textual languages that can be used to model, define, and document processes. A module written in one of these notations or languages that is used to implement a process for execution in an NGOSS system is called a *process definition*.

The definition of the behavior of a business process can be expressed in terms of named process definition modules. Additionally, each activity within the process definition will be named, and corresponds to the invocation of a contract.

5.3.2 Process Definition Lifecycle

Process Definitions can be offered, implemented, activated, and administered like Contract Specifications. A Process Definition can also be made “inactive”, which will cause all Process Managers to be unable to start more processes that are based on that definition.

5.3.3 Process Execution Environment and Context

A *Business Process* is executed by the *Business Process Engine*. The business process engine maintains an overall execution context, termed the *business process engine context*, which keeps the state of the set of business processes currently executing within the system up to date. A business process is defined by a *process definition script*, which contains the sequenced description of the business processes (in terms of contracts) to be executed. Here, a process definition script is not meant to imply any particular scripting language – rather, it simply means a sequenced collection of business processes.

The execution of a contract is called a *contract invocation*, which corresponds to a defined task within the business process itself. This may be the invocation of an information provider contract, or of a (business) service contract[17]. The business process engine executes a process definition script, maintaining a *context* for each *script execution instance.*

From a security standpoint, the NGOSS technology neutral architecture makes the following assumptions:

- A business process, as defined by a process definition script, executes under one or more specified roles
- During the execution of a process, the business process engine assumes the role(s) specified by the process definition script
- The access privileges of the business process engine are restricted as appropriate to the specified role(s)
- An executing script does not have access to the globally maintained business process engine context unless one or more of its roles have access to the business process engine context

In general, an executing business process has access to the following items:

- Any information passed to the process definition script execution context by the business process engine
- Any request parameters passed to previous contract invocations within the same process definition script execution context
- Any response parameters generated by previous contract invocations within the same process definition script execution context

5.4 Security-Enabled Architecture

Security is an essential ingredient in the development of NGOSS systems and should not be considered an item that can be incorporated into a solution at a later date. Rather, Security capabilities must be pervasive throughout the whole NGOSS environment and must be woven into NGOSS system solutions from the outset.

The NGOSS security-enabled architecture is described in annex TMF053s. The reader is referred to that document for more details.

5.5 Policy-Enabled Architecture

The policy aspects of the NGOSS architecture are described in annexes TMF053c and TMF053p. The following provides an overview of the description contained therein.

Users, applications, services and resources of an NGOSS system may be categorized by their role(s), which defines the set of resources and services that they have access to. Policies may be used both to determine which role(s) an entity user or resource is assigned, as well as to dynamically determine the specific set of resources and services that a given role has access to. Note that policies may be used to alter the set of resources and services, or otherwise constrain them, based on dynamic conditions in the managed environment. The net effect of this is that an entity may be denied access to, or use of all or part of, another entity that it should normally have access to.

The NGOSS policy subsystem acts as a supervisor of operations being carried out by all other services that it manages. Hence, the Policy Management Framework Service can apply constraints and/or conditions on what operations are executed, as well as when, by whom, and how they are executed, within an NGOSS system. These constraints can be used to impose local rules on processing sequences and conditions that are required to be met before (i.e., pre-conditions) and after (i.e., post-conditions and exceptions) any action.

[17] At this time, it is assumed that service providing contracts are providing services of a business aware/supporting nature. It is possible that at some time in the future it may be determined that the execution of a business process may require the ability to directly invoke a framework service.

Policy and process management each have their own advantages. Policy management prescribes declarative control of the system, whereas process management is more imperative in nature. The notion “behavior and control” is defined in TMF053C to represent a balance between policy and process management. Behavior here connotes a way of describing how an entity responds to a certain set of conditions, while “control” is defined as the use of a set of mechanisms to direct how a particular behavior is to be managed or executed.

While an NGOSS system can use either policy management or process management, the NGOSS architecture emphasizes that to do so is to not realize all of the important benefits that can be provided. Rather, the *combination* of using policy management and process management provides a more powerful and effective way to manage and control the behavior of a system.

Another important principle of policy management is the use of a policy continuum. Policies exist at various levels of abstraction within a given system. This is to enable different constituencies to use different types of policies in different ways, while relating each different expression of policy to each other. This is described in more detail in TMF053C and TMF053P.

Each discrete system capability can be constrained by any applicable policy rules. It is important to note, that from an architectural perspective, the NGOSS policy subsystem is an integral element of the overall NGOSS system architecture. This enables a policy to be mapped onto one or more technologies, as defined in the TNA Policy Reference Model [TMF053p]. This approach will enable a policy to be constructed using the most advantageous technology available, thereby “future-proofing” abstract policy definitions by separating specification from implementation.

5.5.1 Towards NGOSS Policy-Based Management Systems

The need for policy is determined by the system builder (re: Section 3.1) based on the operational requirements of the system user (re: Section 3.1). Current OSSs are typified by the following characteristics:

- Large distributed systems are shared by many owners and thus, multiple management systems will be required to cooperatively manage the system(s) as a functional whole.
- The correlation of management activities requires one or more special correlation systems
- A single management system cannot handle the set of unique problems that each component in the distributed system has
- User behavior may differ from one subsystem to another, making it difficult for one single management system to manage user behavior across systems and subsystems.

Systems that exhibit one or more of the above characteristics are prone to failure because, if the management system fails, there is no backup management mechanism. Policy-Based Management Systems (PBMSs) have been proposed to solve the preceding list of problems. This has two attendant benefits. First, it improves the reusability of the management system. Second, it increases the availability of the overall system.

Current operational support systems (OSSs) are built in a “one-off” fashion according to the needs of a specific Service Provider and/or Network Operator. In contrast, an OSS that contains a PBMS will be built to a more generic set of requirements. The functionality of a policy-managed OSS will be constrained to the requirements of any given system user through the application of policy

5.6 A Shared Information and Data Environment

A basic tenet of NGOSS is that all enterprise-wide data is stewarded. That is, one or more processes control all access to stewarded data based on applicable access policies.

The SID is a federated set of models, each of which covers one or more management domains. It is not considered feasible to have a single information model that can be used to represent the full diversity of information that would need to be shared within a typical NGOSS compliant system. The NGOSS SID, as defined

in GB922, is two things – a federated information model as well as a set of mappings to a set of data models. An information model is independent of the type of repository, access protocol, software implementation, and platform used – its purpose is to define common information structure, characteristics, behavior, and inter-entity relationships. The SID is federated for two reasons. First, this enables the SID to function as a framework, where lower-level details may be provided by the SID or by other existing models from other standards communities. Second, it enables applications to view the SID as a set of extensible building blocks. They don't need to use the entire SID (which will be quite large) – rather, they use only those portions that are applicable to their application.

This single information model is then translated to a set of data models. Each data model is optimized for a specific combination of repository and access protocol that is being used to steward particular types of data. This is necessary because management data is diverse in nature, and no one repository and access protocol have sufficiently diverse characteristics to steward all of the data concurrently. This illustrates the necessity of having a single common information model – if there were multiple information models, then data coherency of a given entity could not be maintained across different implementations in different repositories.

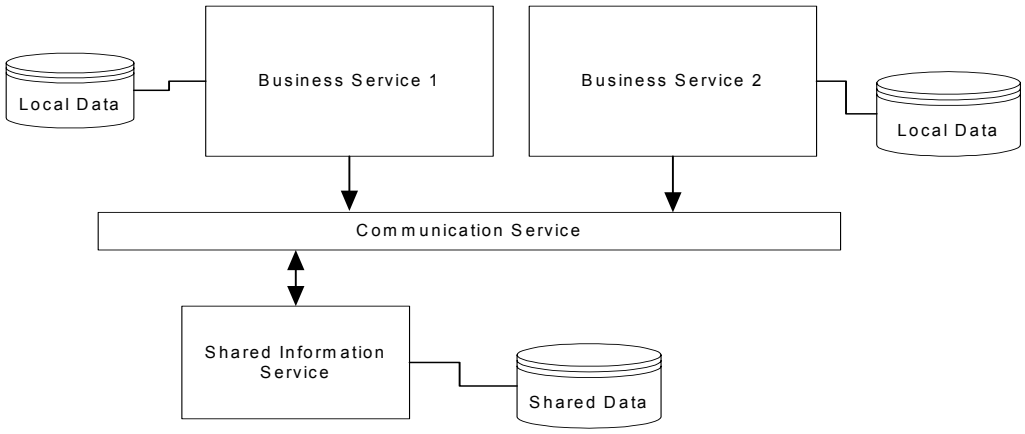

Figure 11: Shared Information Management

Figure 11 depicts two business services, each managing its own local data store, that also have a dependency on some shared information, which is managed by a system-wide shared information service. In other words, the SID is enterprise-wide, and allows heterogeneous systems that have their own private data to communicate with each other using public shared data that is owned not by any one system, but rather by the entire enterprise. The SID enables the two business services to agree on common definitions of the shared data and reuse it for their own application specific purposes. They are each free to refine the shared data to suit their own needs, but can interoperate based on the common understanding and representation of the shared data.

5.6.1 The SID

The concept of the SID is fundamental to the principles of NGOSS-based systems. It is the means by which useful information, relevant to a given set of business processes, may be factored out, shared, and operated on by multiple business processes on a system-wide basis. Examples of such information include SLAs (Service Level Agreements), Customer Details, Network Topology, and Performance information.

GB922 defines the NGOSS Shared Information and Data (SID) model. Please note that this is a work in progress. See the SID charter for the current and future scope of this work.

Instances of elements defined within the SID are accessed by the entities within an NGOSS deployment via well-defined contract interfaces, termed information providers. Information providers support *information service*

contracts. Collectively, the set of *information services* provided through these Interfaces provide access to the entirety of the shared information in an NGOSS deployment. The entities that present information provider contracts are termed *data stewards*.

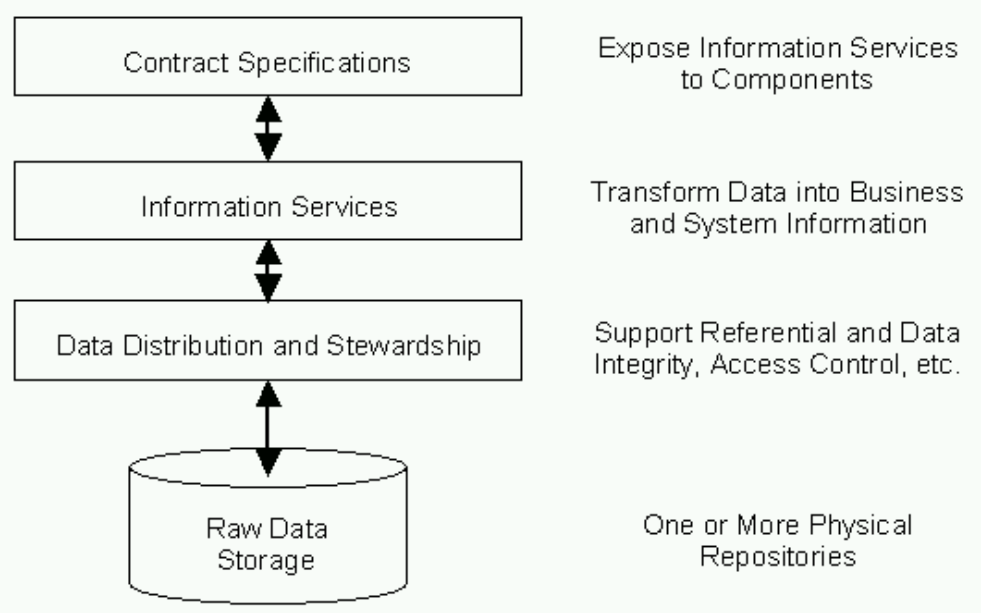

Figure 12: Information Layers

In order that the logical concept of the SID may be implemented, a number of supporting Framework Services will need to be physically deployed. These are shown in Figure 12 as layered services, where each builds upon the services offered at other layers below it:

- At the top-most level are the Contract Specifications, which support the SID as outlined above.
- The Information Services layer adds business value, transforming raw data into useful business information (this may include data mining and filtering). This layer is also responsible for adapting and/or mediating data from the enterprise-wide common format into the particular format that any contract or Component implementation needs. See section 5.6.2 for details on adaptation and mediation.
- Supporting the Information Services layer is the Data Distribution and Stewardship layer, which supports access and sharing of data across a number of physically distributed repositories. Such items include support for data synchronization and maintaining transactional and referential integrity.
- At the bottom-most level are the separate raw repositories that hold and maintain the data. This level performs raw data distribution, but may also include ancillary functions, such as acting as a data proxy.

The NGOSS SID is a shared information model that is refining based on the UML metamodel. The SID consists of a set of entities that are organized into a set of management domains. These domains provide a mapping to the business and system views of the NGOSS system, as defined using eTOM and SIM terms. This is achieved by starting with the UML metamodel and refining its generic concepts to suit the application-specific needs of an NGOSS system. The instantiation of the SID provides one or more SID-compliant data models, which are present for all NGOSS Components to share and use. An NGOSS repository is used to define which data stores have which data. The details are as follows:

- Components offering contract services do so in terms of common data types. These are defined in the SID, and use standard UML extension mechanisms (e.g., stereotypes, tagged data, and OCL) to extend the data types defined in the UML metamodel.

- Components can internally deal with non-standard models, but require adaptation and/or mediation to the SID. In such a case, Components offer contract services along with adaptation and/or mediation functionality.

The benefit of these principles, as embodied in the SID, is that it decouples the consumers of data from the producers of the data. Instead of every OSS Component being tightly coupled to every other, they are coupled to only to the specification of common SID data within an NGOSS deployment. Therefore, whenever an OSS Component needs to be removed, upgraded or replaced, all other Components are unaffected - the only thing that needs to change is the definition of the contract interface.

It is expected that an NGOSS deployment will contain a (perhaps large) number of shared information providers. This number will be dependant on the partitioning of the SID data based on the requirements of the deployment.

5.6.1.1 Example of Using Shared Information

Consider three different applications: inventory, service activation, and topology. Without a shared information model, these applications have to implement common sub-services, such as resource discovery, to perform identical tasks that they all need. Unfortunately, this usually results in the three applications implementing the same function three different ways. This in turn leads to either loss and/or misrepresentation of information.

Now consider the same system that uses the SID. A single discovery application can be used to place common data in a shared repository that can be used by each of the three applications. For example, suppose a new IP router is discovered. The discovery service can obtain characteristics such as its IP address, number of active ports, current configuration, and routing policies. Some or all of these data can be used by each application.

Even better, the discovered data can be modified by a second application and used by a third. Continuing the example, the inventory service updates its repository with the IP address of the newly discovered device. This is then used by the topology service to find out what that device connects to, and adds the resulting information into the shared repository. The service activation application takes this information, modifies the configuration of the newly discovered IP router and another existing router, and constructs a VPN. The service activation application then places these modified data back into the shared repository. This modified information is then extracted by the inventory service to update its record of which ports on which devices are now in use, and what services are supported by what devices.

Within the NGOSS architecture, adaptation needs to be carried out to enable different NGOSS Components using disparate technology specific implementations to be able to communicate with each other. Adaptation may be used to address differences in messaging technology, data representation and differences in implementing entities from disparate data models.

5.6.2 Adaptation and Mediation

5.6.2.1 Adaptation between Information Models

A Business Service may be built to one information model but deployed in an enterprise that uses a different information model. It will then be necessary to add either adaptation or mediation services to enable the Business Service to interoperate with the remainder of the system. This enables the business process control entity to use the Business Service without compromising either information model.

An adapter is a Component implementing an Adaptation Service. An NGOSS adaptation service is one that converts one entity that has a particular syntactic representation to another entity that has a different syntactic representation in order to enable the two to interoperate. The specification of this mapping may be hard-coded within the adapter, or may be more dynamic.

5.6.2.2 Adaptation between Data Encodings

This is seen as a technology-specific issue, and is therefore out of scope for this document.

5.6.2.3 NGOSS Mediation

Mediation is defined as a mechanism to enable semantically different entities to interoperate. Often, the means to achieve mediation is to translate each of the entities to be mediated to a common third form (e.g., a common information model, such as the NGOSS SID). Mediation enables entities to interoperate in a loosely coupled way. Note that adaptation is concerned with the translation of syntactically different entities and may therefore lose some of the semantic meaning in the process. Mediation seeks to preserve the semantic meaning of each entity.

Mediation is important because, in order for the concept of plug-and-play to become a reality, software systems must be able to communicate between one and another, regardless of the implementation technology of the systems. Mediation is therefore useful to enable the semantics of different systems to be preserved while enabling them to interoperate.

Contract interfaces for legacy systems (e.g. non-NGOSS) may be supported in an NGOSS environment by creating virtual Components through *mediation interfaces* placed between legacy systems and the NGOSS system. This approach may also be used to supply a migration strategy for legacy systems into a pure NGOSS environment (e.g., Customer Details information migrated out of a legacy system into an information server – based upon a common data warehouse that uses common shared data).

This approach enables business processes and information, which are already available but 'locked-up' within legacy systems, to be accessed by other systems in the NGOSS deployment. Without mediation, these items of business process and information would have to be reproduced in the NGOSS deployment. Not only does this represent a significant cost, the likelihood of introducing errors is considerable. An NGOSS Legacy System Component is a specialized NGOSS Component in which mediation has been used to enable the legacy Component to appear to be an NGOSS Component. An NGOSS Legacy System Component is bound by same requirements that an NGOSS Component is bound by, at the contract-defined interface.

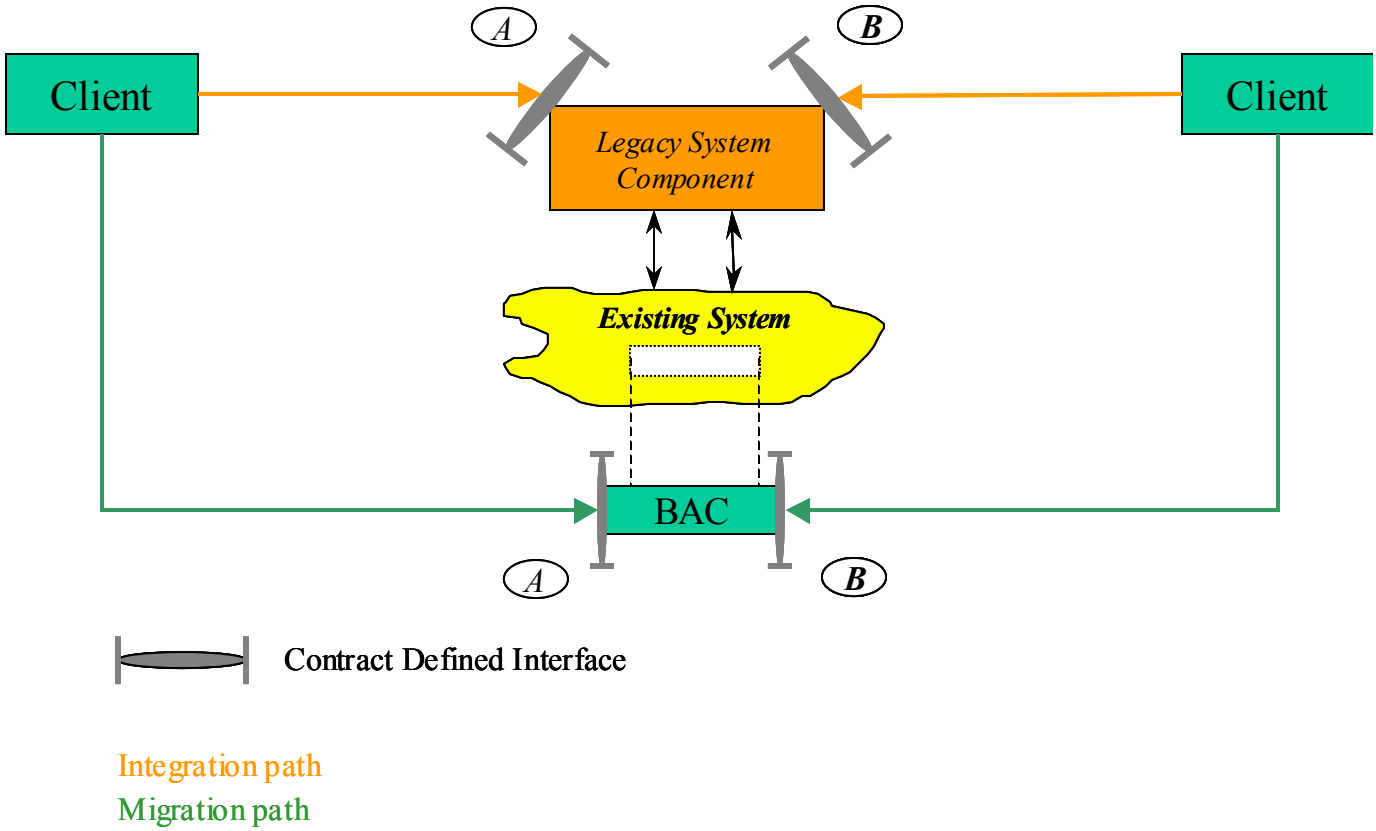

Figure 13: NGOSS Legacy System Component conceptual view

Figure 13 graphically depicts the NGOSS Legacy System Component concept. The upper integration paths connecting to the NGOSS Legacy System Component enable clients to interact with a legacy system as if it were a native NGOSS Component. The NGOSS Legacy System Component can be an integral piece-part of the legacy system, or an entity that exists externally to, and independently of, the legacy system. The lower migration path shows the eventual replacement of the legacy system with a native NGOSS component implementing the required contracts.

5.7 Distribution Transparency

The distribution transparency aspects of the NGOSS architecture are described in annex TMF053f. The following provides an overview of the description contained therein.

5.7.1 A Repository of Runtime Information

As discussed earlier, an instance of the NGOSS architecture is composed of loosely coupled distributed Components communicating with each other, providing value-added services and managed using policy and process management engines. NGOSS clients as well as Components need to locate other NGOSS Contracts and the Components that implement them without knowledge of their physical location in the network. This is called Distribution Transparency.

The essential framework entity that supports Distribution Transparency is the Repository. The Repository enables the Naming Service, Registration Service and the Service Location Service to interoperate to provide distributed

storage, management, and retrieval of NGOSS information, such as contracts, shared information and data objects, and so forth. The naming service enables names to be defined and manipulated for objects; the registration service provides a management interface to add, modify delete and browse objects; the service location service enables fuzzy searching and matching based on attributes that have been bound to a name.

The service location service (in conjunction with the naming services) provides for contracts also to be traded based on client specifications.

5.7.1.1 The Naming Service

The NGOSS Naming Service manages the various named objects in the system. These include the Contract Specifications, the identity of the Component implementing the Contracts, and the metadata and contents used for shared data. It is organized around a federated, multi-level namespace, which allows groups of names to be partitioned into different namespaces and still provide a consistent and coherent naming methodology. This is done to avoid name collisions as well as to perform "named object acquisition" by having name-based searches continue into parent namespaces when a match is not found. This enables the deployment of a system to have standard named objects located in higher namespaces, while deployment-specific overrides are stored in lower namespaces and acquire the standard environment when not overridden.

The namespace tree is designed to hold standard groups of names, much as how an operating system's file system has standard directories. For instance, example namespaces could include the "Component ID" group, the "Shared Information Object" group, and the "Contract" group. These are simply examples and should not be construed to represent or suggest the layout of the NGOSS namespace in an actual deployment. The standardization of high-level Components of the namespace tree is essential in order to provide interoperability. Without these standardized Components, different applications will be unable to find common information. This prevents multiple applications from sharing common NGOSS information.

The NGOSS architecture assumes the existence of four (4) different types of classes to represent names. The most important of these classes is *Name*. Class Name defines the basic behaviors common to all types of Names. The syntax of a given Name is governed by a n*aming system.* A naming system contains a *schematic convention* and a *naming convention.* The schematic convention of a naming system governs the relationships that can be established between Names within that naming system. The naming convention of the naming system governs the construction of a Name from the set of symbols that are valid within the naming system. A *name space* is the set of all names within a naming system.

Three classes implement the interface defined by Name. These classes are

- *AtomicName -- used to represent a single* Name *that cannot be further decomposed.*

  example: "www", "tmforum", and "org" are all atomic names within the DNS naming system

- *CompoundName – used to represent a* Name *that is an ordered collection of* Names *from the same naming system*

  example: "www.tmforum.org" is an instance of a CompoundName within the DNS naming system

- *CompositeName – used to represent a* Name *that is an ordered collection of* Names *from two or more naming systems.*

  example: "http://www.tmforum.org" is an instance of a CompositeName from the URI Naming System, and is a valid name in the Internet Namespace.

Next figure graphically illustrates the relationships between class Name and its three (3) subclasses.

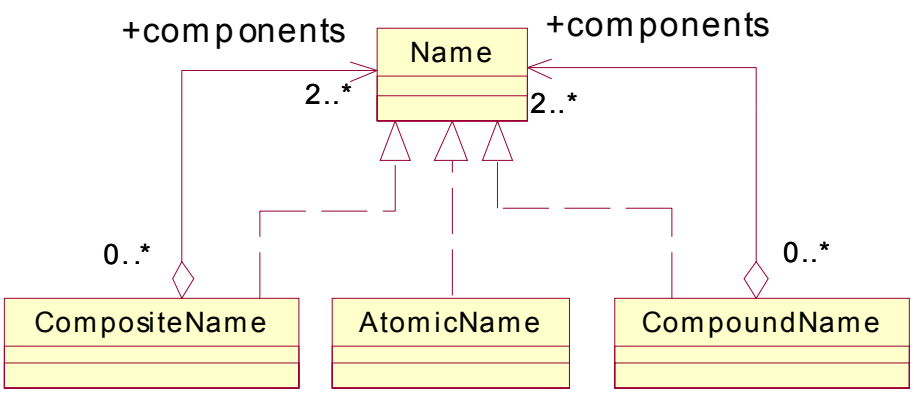

Figure 14: Types of Names

5.7.1.2 The Registration and Naming Services

The Registration Framework Service uses a specific Naming Service in order to allow the binding of names to objects.

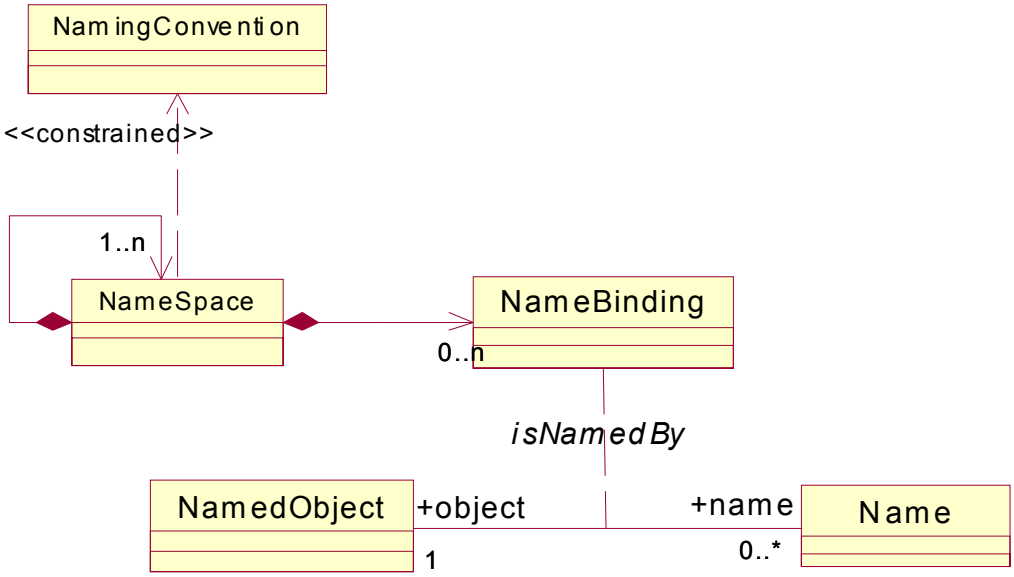

Figure 15: Binding of a Name to a NamedObject

The Registration Service is used to maintain information about the run-time state of the system, such as the deployed Components and their Contract Specifications, where they are located, and how to invoke them. The Registration Service manages the global namespace for contracts, information objects, Components, processes and their specifications in the system. At design-time, it provides a catalog of required and implemented Contract Specifications, and at run-time it provides name-based lookup of shared Components and Services. This is used by the Service Location Framework Service to perform attribute-based matching. There may be one or more Registration Services.

The NGOSS Repository provides a unified access to NGOSS System for the Naming Service, the Registration Service, and the Service Location Service. The NGOSS Repository provides a multi-level namespace through

these three services. This allows groups of names to be partitioned. Example namespaces include the "Component ID" group, the "Shared Information Object" group, and the "Contract" group. An important aspect of the NGOSS Repository is that namespaces can be contained within other namespaces, so that entry in the Component ID namespace for a Component can contain subordinate namespaces for other Components.

Next figure provides a control diagram that shows registration of a Component as it is installed into the system, and how later NGOSS software elements discover it by querying the Repository.

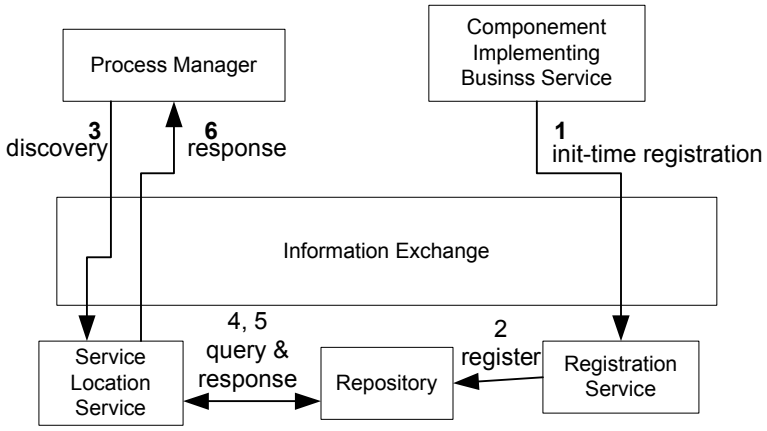

Figure 16: Using the Registration Service and Repository

For larger implementations, the NGOSS Repository will support federation and derivation for the purposes of partitioning and scalability of its location services. Federation is the ability to group physically separated entities (in this case registries) into a single logical whole. Derivation is the ability to create subset caches or superset unions from Repositories. Partitioning enables the different physical Components of the logical Repository to be treated differently (e.g., managed as well as accessed). The scalability of the Repository is increased by decoupling the various Components that make up the Repository. This enables them to be developed and managed in parallel. Through these mechanisms, the NGOSS Repository can grow and support NGOSS installations as they grow over time.

Next figure shows how the Registration Service Naming Services interact with each other and the Repository.

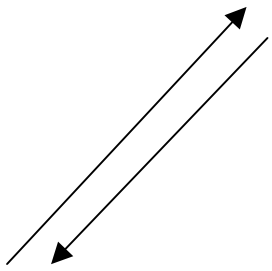

Figure 19: Using the Registration and Naming Services

5.7.1.3 The Service Location Service

The Service Location Service extends the functionality of the repository by supporting the capability to bind attributes to objects as well as the facility to search for the best match against a set of objects using the attributes that have been bound to the objects as part of the search criteria. This service is intended to be used to support locating NGOSS contract instances, given the instance properties, rather than the instance identifiers.

The use of the Service Location Service is similar to that shown in the control diagram for the Naming Service, except that the Service Location Service is invoked to perform discovery of NGOSS Contract Instances. The Service Location Service uses the same information contained in the Repository that is registered via the Registration Service.

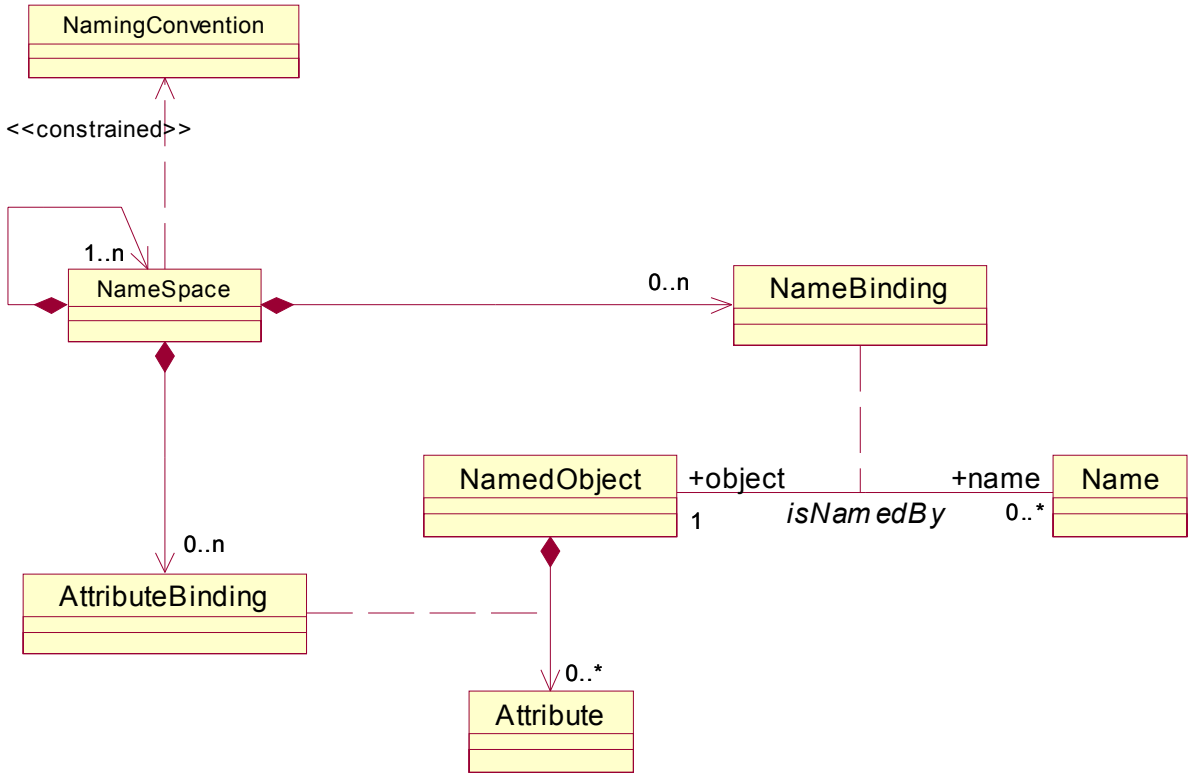

Figure 17: Service Location Service Extensions

The service location service maintains a set of AttributeBindings. Each AttributeBinding allows the attachment of an attribute to a NamedObject that can then be used by the Service Location Service to resolve trading requests.

5.7.2 The Invocation Mechanism

There are several steps associated with requesting invocation of a Service on a remote Component, namely: the Service implementer must be properly located, the Communications Service must be properly used, and results must be logged. Security must be applied at each step in the process. For this reason, the Invocation Mechanism is supplied as a convenient way to perform all of these steps and ensure the integrity of the operation in the context of the overall system and the client invoking the service. The Invocation Mechanism calls upon the Communication Mechanism and other services, as necessary.

The Invocation Mechanism will be the primary way that remote, authorized, and contract-validated invocation of one or more Business Services supplied by another NGOSS Component is performed. The specific steps associated with invocation include the following:

(1) Locates an implementation of a Business Contract.

(2) Queries the Security Service to determine if the operation is authorized.

(3) If authorization fails, reports an exception condition; otherwise, continues.

(4) Marshalls all parameters of the request into the implementation requirements. Sends the request and waits for the reply.

(5) Marshalls the response from service implementation and handles exception conditions.

(6) Logs the invocation and its status (e.g., whether it was successfully invoked).

(7) Returns the response to the requestor of the invocation.

The Invocation Mechanism performs all of the above steps to ensure the integrity of the invocation.

6 References

The follow documents were referenced within this document, or are considered be relevant to and supportive of this document.

ACT The TMF Application Component Team.

BIZ TMF 051: New Generation Operational Support System (NGOSS™) PART I (Business Case). TeleManagement Forum, 2001.

CIS TMF 045: The Common Information Structure. TeleManagement Forum, 1999.

COMP TMF 050, “NGOSS Compliance Testing Strategy Technical Specification”, TeleManagement Forum, 2002.

CORBA “Common Object Request Broker Architecture (CORBA/IIOP)”, Object Management Group, 2002.

CORBAAppNote TMF 055, "NGOSS Phase 1 Technology Application Note - XML", version 1.5, TeleManagement Forum, August 2001.

DCOM “Distributed Component Object Model Protocol-DCOM/1.0”, Microsoft Corporation, 1996.

DIM TR 127, “NGOSS: Development and Integration Methodology”, TeleManagement Forum, 2001.

eTOM GB 921, enhanced Telecom Operations Map™ (eTOM), version 3.0. TeleManagement Forum, 2002.

GAM95 Erich Gamma, et al. Design Patterns: Elements of Reusable Object Oriented Software. Addison-Wesley, 1995.

GRAY93 Jim Gray and Andreas Reuter. Transaction Processing: Concepts and Techniques

HER00 Peter Herzum and Oliver Sims. *Business Component Factory*. OMG Press, 2000.

Java RMI “Java™ RemoteMethodInvocation Specification”, Sun Microsystems, 2002.

JINI “Jini™ Architecture Specification”, version 1.2, Sun Microsystems, 2001.

MDA *Model Driven Architecture,* OMG White Paper, November 2000.

MIL David Milham, British Telecom

REQ TMF 052: New Generation Operations Systems and Software (NGOSS™) PART 2 (Requirements). TeleManagement Forum, 2001.

RM-ODP “Information technology – Open Distributed Processing - Reference Model: Overview”, ISO/IEC 10746-1, 1998.

SEI00 “Volume II: Technical Concepts of Component-Based Software Engineering, 2nd Edition”, Software Engineering Institute, Technical Report CMU/SEI-2000-TR-008, 2000.

SID GB 922, “Shared Information/Data (SID) Model”, TeleManagement Forum, 2002.

SIM GB 914: System Integration Map, version 2.5, TeleManagement Forum, 2000.

SOAPAppNote TMF 057: "NGOSS Phase 1 Technology Application Note - XML", version 1.5, TeleManagement Forum, December 2001.

SZY99 Clemens Szyperski. Component Software: Beyond Object Oriented Programming. Addison-Wesley, 1999.

TINA-C “TINA”, Martine Lapierre, Prentice Hall, ISBN: 0130954004, 1999.

TMF053A TMF 053: The NGOSS™ Technology Neutral Architecture Specification; Annex A: Glossary. The TeleManagement Forum, 2001.

TMF053B TMF 053: The NGOSS™ Technology Neutral Architecture Specification; Annex B: Contract Specification. The TeleManagement Forum, 2001.

TMF053C TMF 053: The NGOSS™ Technology Neutral Architecture Specification; Annex C: Behavior and Control Specification. The TeleManagement Forum, 2001.

TMF053D TMF 053: The NGOSS™ Technology Neutral Architecture Specification; Annex F: Metamodel Specification. The TeleManagement Forum, 2001.

TMF053F TMF 053: The NGOSS™ Technology Neutral Architecture Specification; Annex F: Distributed Transparency Services Specification. The TeleManagement Forum, 2001.

TMF053M TMF 053: The NGOSS™ Technology Neutral Architecture Specification; Annex F: Process Management Specification. The TeleManagement Forum, 2001.

TMF053P TMF 053: The NGOSS™ Technology Neutral Architecture Specification; Annex P: Policy Specification. The TeleManagement Forum, 2001.

TMF053S TMF 053: The NGOSS™ Technology Neutral Architecture Specification; Annex P: Security Specification. The TeleManagement Forum, 2001.

TOM GB 910: The Telecom Operations Map, version 2.1. The TeleManagement Forum, 2000.

UML James Rumbaugh, Ivar Jacobson, Grady Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

WFMC *WfMC Glossary* - The Workflow Management Coalition Terminology and Glossary, May 1996. Document Reference WFMC-TC-1011.

7 Open Issues

- representation of behavior of contract
- component persistency: understand better what really persistency of the component means (differently than of the pure collection of the contained contracts)
- versioning and deprecation for backward compatibility and smooth migration
- refinement of process execution environment description
- more detail on the management contract