Implementation/Realisation
Issues for OSS

David Lewis
These two lectures cover a range of issues related to the Implementation/Realisation of OSS. It addresses the wide range of technologies that are currently being applied to the OSS domain as well as the issues raised by the necessary co-existence of these technologies and the approaches that can be taken to dealing with this heterogeneity.

The emphasis is on approaches that are subject to standardisation or industry agreements, especially those targeted at the management domain.

The following are covered:

Integration Technologies for OSS: An overview of different technologies being applied to management covering, interface adaption issues, message based systems, workflow management systems, web-based management,

CORBA and its evolution into Component-based frameworks for enterprise integration. This shows how the J2EE platform brings together many of the preceding issues, applied to OSS development via the OSS through Java (OSS/J) initiative.

Architectural and Modelling Issues: This covers non-technology related issues concerned with the modelling of OSS and the services its needs, using the TeleManagement Forum’s NGOSS architecture.
The Evolution of OSS Technologies

- Early OSS: Management specific protocols
  - SNMP, CMIP
- OSS in the 1990s: General purpose distributed computing
  - DCE (DME), CORBA, COM, RMI, …
- Today:
  - Components-based Software
    - EJB, .NET, COBRA Components
  - Web-enabled management
    - WBEM, SOAP, WSDL …
  - Workflow Management
  - Adaptation technologies

Early attempts to define standard OSS platform focussed around the definition of management protocols such as SNMP for Internet Management and CMIP for OSI Systems Management. These were tailored to the needs of network elements management, providing efficient and timely retrieval of management information, both synchronously and asynchronously. However, the limited size of the network management market made the development of OSS based on these types of protocols a fairly specialised skill with a very narrow and expensive tool set.

During the 1990s, distributed computing penetrated the software engineering mainstream, resulting in the emergence of de facto and de jure distributing computing standards from the likes of X/Open (DCOM), the Open Management Group (CORBA), Microsoft (COM) and Sun (Java-RMI). These have been adopted in many OSS, especially in areas away from element management where the requirements for large scale, timely information retrieval are not so stringent. This has allowed OSS development to benefit from the economies of scale resulting from sharing skills-base and tools-base with other sectors.

The trend of exploiting general purpose software development and system integration technologies has continued strongly in OSS development. Currently we can therefore observe the use of component-oriented development techniques, web-based integration, workflow management and various forms of adaptation techniques being adopted in OSS development.
Technological (Un)Certainties

• There is a steady flow of technological innovation applicable to management systems

• Many innovations have involved adoption of different technology specific models and interaction paradigms, e.g.:
  – GDMO and manager-agent paradigm for CMIP
  – IDL and RPC paradigm for CORBA
  – XML and message passing for SOAP

• Developers must acquire new technology skills whilst maintaining and enhancing existing management know-how!

• Will always need technology interworking to accommodate legacy systems

The constant technology churn experienced in the management domain presents an ongoing challenge for system developers. Developers must often need a grasp of several different computing platforms and development environments. However, more fundamental to the development of long-live solutions are the differences in the levels of abstraction offered to developers by different technologies, the expressiveness of the languages used at APIs and for defining interoperable interfaces and the data type set employed. For example:

OSI Management offers the GDMO language for defining interfaces, but not a standardised API (the TM Forum did standardised a TMN C++ API but this is little used). The CMIP protocol provides powerful but relatively complicated operations, of which the programmer must have a good understanding. The ASN.1 encodings used, while optimised for data transmission, are relatively inaccessible to the programmer.

CORBA offers the Interface Definition Language (IDL) which is close in syntax and data types to common OO programming languages. CORBA also supports IDL compilers which generate code for programming applications. The IIOP protocol is a simple protocol supporting remote procedure calls (RPC), which supports the programmer’s objected oriented method invocation view.

The Simple Object Access Protocol (SOAP) offers a simple RPC protocol, can also be used for message based interactions and enjoys simple API support in various platforms. XML, which is used in SOAP payloads, only provides a way of defining tags for user defined languages, so application developers need to understand the specific XML vocabularies defined for exchanging interoperable messages.
Taking a typical OSS development scenario, developers must address several issues:

*Selection of a computing platform:* A common computing platform applied across the OSS will simplify the skill set needed by the development team and reduce problems in defining, implementing and integrating different parts of the OSS. The choice of platform must satisfy the OSS requirements for distribution, performance, reliability, scalability and security.

*Selection of third party components:* Components from independent software vendors may enjoy a wide market as well as being exposed to competition with similar components from other vendors. This may reduce software lifecycle costs compared to bespoke components, and improve their reliability.

*Support for user interaction:* The OSS must be able to interact with local operational support staff as well as customer service administrators and users.

*Integration with existing OSSs:* already in place within the service provider organisation, especially when these use a different computing platform to that adopted for the OSS under development.

Integration with existing corporate databases.

*Integration with Element and Network management systems* using manager agent style protocols, such as CMIP and SNMP.

Integration with the systems of other service providers.

OSS software must be *flexible* to respond to fast moving business requirements.
The variety of different technologies potentially used in the develop of an OSS raises potentially insurmountable problems of interworking between components which use different interface technologies, such as SNMP, CORBA, SOAP etc. Typically this involves the use of a stand-alone interworking component that adapts from one technology to another. A separate adaptation solution is then required between every pair of technologies. The usual approach to handling this problem is to adopt a single core interaction technology for an OSS project, e.g. a CORBA bus, and to ensure all components used in the OSS support that technology. Then adaption solutions only need to be found for interaction with systems outside the control of the OSS developer (such as other existing OSS or element agents, or systems in other organisations) for which other technologies may be selected.

Such technology adaption is suitable for problems where one component is able to cope with the model (both functional and information) exposed by the other, once the appropriate technology/protocol adaption has been made. A more difficult problem involves mapping the model presented by one component to another existing model that is required by another. The typical solution to this problem is to avoid it by agreeing on these models beforehand, preferably through an international standardisation process. However the range of different management standards and the tendency to supplement them with vendor specific elements for competitive advantage, means that components from separate vendors may well have differing models. Highly different models inevitably require hand-craft adaption solutions, however tools are available for mapping between relatively similar models, for instance where one model is more strongly typed than another, e.g. integers and reals in one SLA structure could be converted to string tuples in another. Such conversions can be driven by scripts passed to a generic model adaption solution.
The figure provides two suggested configurations for run-time interaction between CORBA and OSI based management systems. The aims of both of these schemes is for manager and agent systems to be constructed without knowledge of the technology used for corresponding managed and managing systems.

The left-hand figure shows a situation where a gateway translates between a CORBA based manager and an OSI-based managed system. To operate in a generic way this gateway must offer an IDL interface that corresponds closely to the functions offered by CMIS. In addition the gateway must be able to handle notifications emitted by the OSI agent’s MOs and forward them the appropriate CORBA managers. Some form of integration with the CORBA Event Service may provide a solution for this.

The right-hand figure shows a possible scenario for allowing an OSI manager access to both conventional OSI MOs via an OSI agent as well as CORBA based managed information via a gateway. The gateway would need to be able to translate from CMIP requests into operations on CORBA MOs (including support for scoping and filtering) as well as translating events from the CORBA objects into CMIP event notifications.

These are examples of interworking where the client component is happy to accept the model generated by adapting the existing model to the client technologies. The Joint Inter-Domain Management (JIDM) taskforce has defined such mapping between SNMP, CMIP and CORBA, these being part of standardised interworking solution between these technologies.
Message Based Systems

- Communication is based on the transmission of asynchronous messages
- Publish Subscribe model:
  - Sender does not care who receives the information
  - Receiver decides to which senders it listens
  - Messages may be filtered
  - Messages may be received by multiple listeners, i.e. multicast/broadcast
- Interoperability standardisation focusses on message formats rather than interface definitions

Message based systems aim to support loose coupling in design between different components, and is therefore seen as useful in system made up of software components obtained from separate sources and designed at different times.

In the *Publish-Subscribe* model of message-based communication, components communicate via asynchronously transmitted messages. Components emitting information transmit a message without being concerned which components receive it. They simply publish to a messaging service the message types that they will emit. Components wishing to consume this information uses a common interface of the messaging service to subscribe to a chosen stream of messages. Additional message subscription service capabilities may allow messages to be filtered based on source components, type of message, logical conditions on the content of the message or the occurrence of specific events. This model allows more than one component to consume the same message stream, thus supporting the multicasting of information.

This approach does not require agreement on a interface with a fixed set of operations and data types. Instead, only the interface for subscribing to emitted messages need be agreed. The receiving components need to be able to understand consumed messages, either through standardization of message structure, or the co-transmission of the message structure with the message, e.g. self description by sending the DTD with an XML message.
Workflow & Process
Automation at a glance….

Workflow: “… defines, manages and executes processes … driven by a computer representation of the process logic.”

Process: “… one or more linked procedures or activities which collectively realise a business objective or policy goal, normally within the context of an organisation structure defining functional roles and relationships …’

A workflow is a description of a sequence of tasks that need to be carried out to achieve an objective. These tasks may be carried out manually or automated by a computer system. A Workflow Management System (WFMS) manages an organised set of workflows (also termed business processes). Example workflows in an organisation may describe the sequence of tasks necessary to process an insurance claim, fulfil a book order or handle a customer complaint. The WFMS streamlines these workflows by automating tasks that can be competently completed by the system, leaving the human worker free to handle more challenging tasks. A process definition is a complete system specification of a business process that can be executed at runtime in the WFMS. The process definition defines the control flow, data flow and actors involved in the business process.

The major benefit of a workflow approach in constructing enterprise business systems are rapid realisation of business processes using existing business function implementations with little or no coding. Additional benefits are built-in process administration and monitoring tools, and a tight coupling between the business domain process model and the actual process definition implementation. It is also noted that these business processes are the most transient part of an enterprise system; the WFMS enables rapid modification/updating of these transient processes. However the downside of current enterprise workflow systems is that they are still very proprietary. An organisation wanting to adopt this approach will have to buy into the workflow vendor’s complete solution. The biggest drawback currently is their use of proprietary graphical process models, proprietary process definition languages and lack of extension mechanisms for handling more complex control structures. The OMG is addressing standardised graphical representations of business processes in revisions to the Unified Modelling Language (UML).

Some international standardisation of Workflow has occurred in the Workflow Management Coalition (WfMC), and to a lesser extent in the OMG (Workflow Facility).
The WfMC defines Workflow Management as: ‘… the management and automation of a process (in whole or in part) during which documents, information and tasks are passed from one participant to another for action according to a set of rules …’

The WfMC defines a number of standards for interoperating with a workflow engine:

**Interface 1:** Process Definition Tools - Process Definition meta-model and interchange language

**Interface 2:** Workflow Client Applications - APIs for process activity, worklist and process definition operations, together with naming conventions for the above.

**Interface 3:** Invoked applications and **Interface 4:** Other Workflow Enactments Service - standards that define logical message sequences and content, and their mapping to MIME based encodings and XML-based encodings

**Interface 5:** Administration and Monitoring Tools - Standards for logging operational audit data
Web-based Management

- Business Drivers:
  - Ubiquitous, low cost, extensible WWW infrastructure
  - User familiarity
  - Widespread development skills
  - Proven scaleability and security features

- Application Areas (machine to human and machine to machine):
  - Workstations for human operators
  - Business transactions between operators and with customers
  - Access to management information

- Standardisation efforts:
  - W3C for HTML, XML, SOAP etc
  - E-commerce community, e.g. ebXML
  - T1M1, tML
  - DMTF addressing management

Web based management is motivated by the proven features of its application in the world wide web, including:

The ready availability and low cost of the appropriate infrastructure, including HTTP browsers, HTTP servers and the underlying TCP/IP network.

The familiarity of users with the web interfaces, as well as the large skills base in web server administration and web application development.

Proven security of interactions over public networks and scalability characteristics of existing HTTP server technologies.

These features make Web technology particularly suited to the following areas:

Use of web browsers for presenting management information to users. The ubiquity of browsers enables user access from a wide range of terminals, not just ones with expensive manager applications installed.

The web is used for many business to business applications, exploiting the emergence of e-commerce development tools. This application is being extended to allow service providers to interact with their customers and other service providers.

The small memory footprint of web servers has enabled the use of HTTP server acting as network or element management agents, effectively replacing SNMP and CMIP with HTTP with the relevant structuring of the payload for management information, typically using XML. As a information retrieval protocol, HTTP lacks support for notifications, but various push HTTP mechanisms are available to support this.

The WWW Consortium (W3C) is the primary web standards body, responsible for HTML, XML and SOAP standardisation. There is a range of bodies using XML to standardise HTTP based protocol, such as ebXML for e-commerce transactions. In the US T1M1 has been working on a Telecommunication Markup Language (tML), to provide a basis for defining and sorting XML management vocabularies. The Distributed Management Task Force is is the main proponent of using XML for accessing management information and will be examined in more detail in the following pages.
Distributed Management Taskforce

- Manager-Agent approach based on the definition of managed objects
- Technology Neutral Common Information Model captured using:
  - Managed Object Format (MOF)
  - UML class diagrams
- Multiple protocol mappings:
  - Desktop Management Interface (DMI) – RPC-based, DCE mapping
  - Directory Enabled Networks (DEN) – mapping to LDAP/X.500
  - Web-Based Enterprise Management (WBEM) – Mapping to XML/HTTP

The DMTF is a consortium of IT and management system vendors originally targeting management standards for desktop computers, but later expanded to cover the management of the entire enterprise infrastructure including desktops, servers and enterprise networks. It is also working in areas such as policy based management and QoS management that can be applied to communication services and network.

The approach is similar to the manager-agent paradigm used in Internet and OSI system management, with the object-oriented definition of managed resources as managed objects. Comparitively, the DMTF approach has a stronger emphasis on class associations other than containment and a simpler attribute type set (no composite/complex data-types). The DMTF approach is distinctive, however, in that it takes a technology neutral approach to information models, i.e. information modelling occurs in a language that is independent of the management protocol used to transfer it. A textual, object-oriented language is used called the Managed Object Format (MOF). In addition, the DMTF use UML class diagrams to convey the methods and attributes of classes and their association, though these cannot represent all the language features of MOF.

To support specific technologies, models represented in the MOF are transformed to a technology specific format suitable for transporting using a specific management protocol. Several technology specific mappings have been defined to date, each supporting a different technology and information usage paradigm:

DMI was an initial attempt to provide an RPC-style access to desktop computer management information. The specific technology used, DCE, is now largely replaced by CORBA, RMI and COM.

DEN allows management information to be stored on the existing Internet directory structure accessed via LDAP, allowing a high degree of distribution and federation of management information.

WBEM exploits the previously discussed advantages of XML and HTTP for accessing and...
Web-Based Enterprise Management (WBEM) is an initiative based on a set of management and Internet standard technologies developed to unify the management of enterprise computing environments. WBEM provides the ability for the industry to deliver a well-integrated set of standard-based management tools leveraging the emerging technologies related to XML.

The CIM schemas define a common set of models and semantics for the management systems. Applications can therefore be coded against a known, consistent set of models.

The XML-CIM language defines a common means of representing CIM information in XML documents. With the semantic definitions of interactions with a CIM Object Manager based upon the messages encoded in XML-CIM, applications can reliably interoperate.

The CIM HTTP mapping specification defines the manner in which HTTP is used to transport the CIM information. Applications can be located anywhere in a networked environment.
Information modelling agreements reached by the DMTF, expressed using the MOF, are what is known as the Common Information Model (CIM). CIM was used initially to model management information from Desktop and Server Systems. This has been expanded to cover a wider range of enterprise management problems, including Applications, Systems, Physical resources (racks etc), Devices, Networks and DAP. CIM can be used to describe management information between differing Management Applications such as Tivoli Management Software, CiscoWorks, HP OpenView, Microsoft SMS etc. in order to provide common understanding of management information.

As part of WBEM, an XML Schema has been defined that describes CIM meta-schema, such that CIM classes and instances are valid XML documents that can be passed between management systems.

Compared to other XML encoding approach this has the advantages of only requiring one DTD, avoiding the limitations of DTD (e.g. ordering, scoping and lack of inheritance) and it is straightforward and easy for developers to understand.

The following is an example of class definition in CIM mapping to XML:

```xml
<Class Name="Disk">
  <Property Name="Freespace">200</Property>
</Class>
```
The WBEM messages encoded in XML-CIM are the payload. These messages allow functional access to a CIM Object Manager and allow network based applications the means of performing management operations. The operations fall into four broad categories:

- Data operations which deal with instances of CIM classes
- Meta Data operations which deal with the definitions of CIM classes and qualifiers
- Queries which allow us to access the data and to traverse the models as expressed by the CIM schemas
- Methods which can be invoked on CIM objects and namespaces

The specification defines all interactions between CIM entities as Operations. CIM Operations belong to a larger category known as CIM Messages.

The XML-CIM DTD defines a subelement under the root `<CIM>` element called `<MESSAGE>`, which consists of requests or responses, the version of the protocol in use and identifies the messages so that requests and responses can be coordinated across HTTP messages. To illustrate this, here is a set of incomplete document fragments for a request and response:

```xml
<CIM CIMVERSION="2.0" DTDVERSION="2.0">
    <MESSAGE ID="87872" PROTOCOLVERSION="1.0">
        <SIMPLEREQ>
            ...
        </SIMPLEREQ>
    </MESSAGE>
</CIM>

<CIM CIMVERSION="2.0" DTDVERSION="2.0">
    <MESSAGE ID="87872" PROTOCOLVERSION="1.0">
        <SIMPLERSP>
            ...
        </SIMPLERSP>
    </MESSAGE>
</CIM>
```
The aim of CORBA is to provide an object model and language bindings that allow application programmers to access the services offered by remote objects without regard to distribution issues, i.e. in a manner similar to how one would if the object was in the same address space.

Client objects access services offered by server objects as operations that make up an interface. Objects may have multiple interfaces (though inheritance of interface definitions) and different objects may offer the same interface (though not necessarily with identical behaviours).

Operations are defined by a signature written in the Interface Definition Language (IDL), which has a syntax similar to common programming languages such as C/C++. Bindings to various programming languages allows software written in different languages to interact via CORBA.

**CORBA Overview**

- Object Oriented Client-Server Interaction Model
- Clients invoke operations on server objects via interfaces defined using an Interface Definition Language (IDL)
- IDL bindings standardised for several programming languages:
  - C, C++, Smalltalk, COBOL, Ada, Java ….
The ORB is responsible for ensuring that a client is able to invoke an operation on a remote object. The stub and skeleton code are generated by an IDL compiler thus hiding the complexities of the ORB operation from the application programmer. The programmer writes client codes that calls the stubs and server code that serve the skeletons.

A Dynamic Invocation Interface to the ORB Core is also defined. This supports applications where the syntax of object interfaces are only known at run-time, e.g. object browsers or inter-ORB bridges. It necessarily offers a more complex API to the application programmer than IDL stubs.

OMG has defined the Generic Inter-ORB Protocol (GIOP) that allows for ORB to ORB communication over any connection-oriented transport protocol. The Internet Inter-ORB Protocol (IIOP) allows GIOP messages to be transported over TCP/IP. These protocols therefore support interoperability between clients and server running on ORBs from different vendors.

Activation and deactivation of server object instances and their mapping to processes are controlled via the object adapter. CORBA initially identified the Basic Object Adapter, but did not define it, which led to vendors providing their own solutions, thus preventing server code being easily ported from one vendor’s ORB to another. Later, OMG specified the Portable Object Adapter, standardising the server objects interactions with the ORB, thus helping to overcome this problem.

This combination of distributed computing and software portability form the foundation for later component-based software architectures.
The Common Object Services are types of services that are seen as generally useful regardless of the application domain, therefore to encourage reuse the OMG works towards providing standardised IDL interfaces for them.

The Naming Service provides the management of names and directory-like contexts to objects, and facilities for locating objects by name and context.

The Event Notification Service provides a mechanism for passing asynchronous event notification messages from event suppliers to event consumers. This is an important service in the network management application domain.

The Lifecycle Service supports the creation, moving, copying and deleting of objects.

The Persistence Service supports the persistent storage and retrieval of object state while the object is inactive.

The Transaction Service provides functionality needed for dependable applications, e.g. banking. It is compatible with the X/Open Distributed Transaction Processing model.

The Relationship Service supports the management of relationship between objects, e.g. “containment” and “reference” relationships. Relationship and role objects enable this service to operate separate from the state of the related objects.

Enterprise integration frameworks are software architectures that have evolved from server side platforms such as Transaction Processors and Application Servers, and which build on the capabilities of distributed computing environments such as CORBA and COM.

Such frameworks go beyond a single vendor platform in that they provide more focussed support for the ready deployment of third party application components on a common software platform with no programming intervention. This is a prerequisite to establishing a market in commercial off-the-shelf components that are purchased and deployed without access to the source code.

These frameworks tend to address a wider portion of the software lifecycle, than precursor distributed processing environments. They cover support for the development of components, the assembly of components into specific applications, the deployment of applications onto specific runtime environments and the runtime monitoring and administration of applications.

Another common characteristic of such frameworks are the structuring of their constituent software components into three or more computing tiers. This is an evolution from client-server model, which required the software implementing business logic to be integrated either in the client or the database server. A three tier system separates components implementing the business logic from those implementing the user interaction function and those handling enterprise information. This allows user interaction components to be more lightweight, thus exploiting thin client and WWW browser technologies. Separating business logic implementation from the components handling enterprise information allows the latter to be more general purpose, serving a wider range of business processes. The business processes supported by implementations of business logic tend to be more volatile than the structure of information maintained by an enterprise, so those implementations are more subject to change than those handling enterprise information.
Enterprise Java Beans

- Server Component Model for the Java Language
- Build on visual JavaBeans for clients:
  - build tools customise Beans using properties or customising methods
- Standardised API for EJB Containers providing component management and control services:
  - naming context
  - lifecycle management
  - transaction control
  - persistence management
  - transparent distribution services
  - security services
- Deployment descriptors

Enterprise Java Beans (EJB) is the server components framework for enterprise system developed in Java. It builds on the JavaBean concept developed for visual client applications, where portable beans from different sources can be imported into applications, using properties to customise their behaviour.

EJB run in a container which mediates the interaction of EJBs with other EJBs, client applications, other (non-EJB) software entities and support services. The container provides support for Process Management, Computing Resource Management, Multithreading, Connections Pooling and Caching, freeing the EJB developer from these concerns.

The container provides a set of APIs that provide a platform for portable EJBs. These APIs provide a fixed set of distributed processing service similar to the Object Service defined for CORBA, and in some cases the same API is used.

EJBs are distributed packaged with deployment descriptors as well as properties. The deployment descriptor provides the container with information on how an EJB object should be created and maintained, whether it manages its own persistence or delegates it to the container, what transaction semantics the EJB requires and which security rules should be applied to the EJB.
The *Java 2 Enterprise Edition (J2EE)* Software Development Kit (SDK) builds on the capabilities of the Java 2 Standard Edition SDK. The latter provides support for User Interface pluggable components (Java Beans), applets, distributed interaction via both CORBA and RMI, database access via *Java Database Connectivity* (JDBC) and naming and directory services via the *Java Naming and Directory Interface* (JNDI).

J2EE defines a Container for server application components, which include EJBs as well as *Java Server Pages (JSP) and servlets* provide component that provide web-server functionality.

The container support *connectors*, which provide a mechanism for defining a portable service API to legacy systems such as Transaction Processing Systems, Enterprise Resources Planning systems, Databases or Protocol Stacks.

J2EE provides support for message based communications, both using the native Java publish-subscribe mechanism, the *Java Messaging Service* (JMS), and by providing support for XML based messaging through the *Java API for XML Messaging* (JAXM) and the *Java API for XML Parsing* (JAXP).
OSS Through Java (OSS/J) is a standardisation effort being conducted through the Java community process to define a set of APIs for exploiting Java Technology for OSS. It initially provides APIs for Service Activation, QoS and Trouble Ticketing.

By exploiting the capabilities of J2EE and associated Java APIs, OSS/J provides a platform for OSS development that addresses many of the problems faced by OSS developers:

The development, packaging and deployment of software components as EJBs allow components from multiple vendors to be integrated in a single OSS solution. OSS/J APIs provide standardised interfaces for such components.

Container support for CORBA, RMI and HTTP interactions with managing applications, with HTTP being useful for communication with users in the customer’s domain.

Container support for CORBA allows integration with existing non-Java OSS within the same organisation, while container support for JMS allows loosely-couple integration with existing Java based OSS.

Container support for JDBC allows for integration with existing database installation.

Container support for XML based messaging using the Java API for XML Messaging and the Java API for XML Parsing allow for integration with OSSs in other service provider domains.

Container support for connectors to other management specific protocols such as CMIP, SNMP and WBEM allow for integration to agents and managers that implement these protocols.
For its initial focus the OSS/J community has selected the application area of Service Activation (SA), Quality of Service (QoS) and Trouble Ticketing (TT). These represent area where there is a great demand for off the shelf solutions and where common APIs will provide better consistency between different products, and thus ease the integration task. These APIs offer multiple functions which can be used in different component interfaces, and specialised where needed. A typical configuration of components using these APIs is shown above.

Service Activation is related to the creation, amendment and cancellation of orders from service clients. These API operations initiate the desired changes in the service including activation, deactivation and configuration of the service.

Quality of Service relates to providing a service user with guarantees about the quality of the service they will receive and then maintaining the service to satisfy that guarantee. The API must therefore allow the retrieval, calculation and presentation of QoS data, as well as managing the Service Level Agreement (SLA) maintained between provider and customer and reacting appropriately when this SLA is violated.

Trouble Tickets are documents that enable the tracking of faults from their detection or reporting through to their resolution. The API therefore allows for the creation, removal and modification of trouble tickets and for the notification to clients of these operations.

In addition to API for these specific application areas, OSS/J is defining some common classes that may apply across them (such as ‘users’) as well as documenting design patterns for the development of new OSS/J APIs.
The TeleManagement Forum’s New Generation Operations Systems and Software (NGOSS) initiative aims to define an architectural framework for the development of multi-vendor component-based OSS and for the standardisation of key models in support of consistency between and integration of such components.

A key feature of NGOSS is the separation of business logic from software implementation. This is reflected in the use of process management techniques to flexibly integrate components to perform business processes, using concepts from workflow management. This separation is also supported at a finer grained level by allowing components to be policy enabled, i.e. to have their behaviour modified by changing declarative policies.

The sole means for component interoperability in NGOSS is the Contract, which is an abstraction of an interface offered by a component. Contracts package together logical pre- and post-conditions for the functions it performs with a technology neutral description of the contract and a mapping to a technology specific interface definition.

The TM Forum identifies the following areas where it intends to provide integrated standard models to enable the adoption NGOSS:

- The identification and definition of business processes that make up the operational support activities of a service provider. This is defined in the Telecoms Operation Map (eTOM).
- The definition of contracts that may be used by software vendors as interoperable interfaces to their components.
- The use of shared information through the capture of shared data models which can be used to ensure consistency between the information portion of contract specifications and the development of data stewardship by Shared Data Management Services.
NGSS does not aim to define a new distributed computing or component platform. Instead it aims to develop an abstract definition of the features that are needed in any specific OSS platform supporting NGOSS. This abstract platform definition can then be mapped to specific platforms such as CORBA or J2EE.

Broadly a NGOSS system requires:

- **Business Aware Components**, which are application components that can only interact via their (business aware) contracts.
- A **Common Communications Bus** to provide a single vehicle via which components and other elements of the platform communicate.
- A **Process Engine** which would manage the interactions between contracts in order to implement a specific business process. Changes to the business process result in changes to which contracts are invoked and in which order, but without requiring changes to the contract design or component implementation.
- A **Repository** holding information about available components, the contracts they offer and the policies bound to them.
- **Mediation** capabilities to non-NGOSS systems and systems using a different type of communication bus.

Several services supporting distributed computing including:

- **Naming and Location Service** for contracts
- **Trading Service** to locate contracts through attribute matching
- **Policy Service** to evaluate policies associated with components
- **Security service** providing identification, authentication and authorisation of those invoking contracts
- **Invocation Service** that integrates the invocation of a contract with its location, checking of pre- and post-conditions, satisfying of security rules and logging of transactions.
- **Shared Data Management Service** that controls access and changes to information used by more than one component.
NGOSS promotes the use of the Unified Modelling Language (UML) in developing NGOSS compliant systems, and the TM Forum uses UML in modeling and documenting its standards. NGOSS advocates a development methodology that begins by capturing business processes, which are then mapped to system level processes that may then be realised using NGOSS Business Aware Contracts and Shared Data Models.

The business process flow captures the business requirements for a particular organisation and expresses them as business process flows. In the eTOM the TM Forum has developed a general purpose business process model to provide a common hierarchical set of business processes and end-to-end business process flows. The top level business process grouping cover Customer Relationship Management, Service Management and Operations, Resource Management and Operations, Supplier/Partner Relationship Management, Marketing, Sales and Offer Management and Enterprise Management. End-to-end process flows address Infrastructure Lifecycle Management, Product Lifecycle Management, Supply Chains Lifecycle Management and the Fulfillment, Assurance and Billing of Services.

Process flows are then mapped onto system process plans, which map elements of the Business Process onto sets of System Activities, which are defined in terms of operations on Business Aware Contracts and the exchange of elements from Shared Data Models. The aim in the TM Forum is to use the eTOM to drive the standardisation of a set of Business Aware Contracts and a Shared Data Model from which solutions to system processes in a particular application can be selected, and if necessary specialised.
The public view of a component is advertised using contracts. Each contract has a technology neutral and technology specific part. The technology neutral part is developed by a component designer, and is also the part of the contract that would be subject to standardisation in the TM Forum. The technology neutral part contains descriptive information, which may provide traces to other specifications (such as process models), which have influenced its design. It also contains a definition part which details the arguments of the operations provided by the contract, their pre and post conditions and any exceptions it may raise. This provides the core information needed for interoperability with the contract.

The Technology Specific part is split into an implementation part and an activation part. The implementation part is provided by the component’s implementers and captures details of the implementation technology and any dependencies present in the component's implementation of the technology neutral part of the contract. The activation part would be provided by the application builder who deploys the component in a particular OSS. This part provides details on access and security properties of the contract, typically expressed as policies, as well as the configuration of any attributes that influence the behaviour of the component and its requirements on its computing environment (similar to EJB properties).
### Approaches to Handling Different Technologies

- **OSS/J** focuses on software development and integration:
  - Does not address use of models in non-J2EE environment
- **DMTF** focuses on interoperability models, no component-model
  - Information model style may restrict effective mappings to future
- **NGOSS** captures abstract Contracts and Shared Information Model derived from Business Process Model
  - Needs to manage technology specific mapping to ensure interoperability

In summary it is interesting to compare some of the different approaches to dealing with technology change and the resulting technological heterogeneity that influences the development of an OSS:

The OSS/J approaches leverages the enterprise application integration features of J2EE, which provides the developer with a lot of assistance in interworking between the Java based components developed and other technologies. However, though OOS/J provides some assistance in developing Java APIs for OSS components, it does not address the development of non-Java components, merely the interworking with them. It is therefore restricted to application where Java is a reasonable implementation choice.

The DMTF approach is not tied to any particular technology, but allows its technology neutral information models to be mapped to different technologies. However it is only concerned with interface definition and does not address the the development of components or the platform services that are needed in a software architecture. In addition, its models are firmly placed in the manager-agent paradigm, and therefore may not be well suited for interface definitions in message-based or RPC systems.

The NGOSS approach has the strongest separation between the modelling of interoperable interfaces and their binding to technologies. It has a strong focus on modelling at a business level as well as at a design level, which may be important in comparing and selecting between models from different sources. It provides a component-based systems view, but only in an abstract manner, with the mapping to concrete software platforms remaining largely untested. The potential problems in interworking between different technology mapping of the same technology neutral contract specification also remains unexplored.
Further Information

- OSS/J
  - java.sun.com/products/oss/
- DMTF
  - www.dmtf.org
- NGOSS
  - www.tmforum.org
- UML
  - www.uml.org
- OMG CORBA
  - www.omg.org
- XML, SOAP etc
  - www.w3c.org
- Joint Inter-Domain Taskforce
  - www.jidm.org
- WfMC
  - www.wfmc.org
- FORM web site:
  - www.ist-form.org