

# e-Framework for Education and Research

## Core Components and Concepts

September 2006

**NOTE:** This document represents a work-in-progress primarily intended to inform the development of the e-Framework knowledgebase. In time it will be superseded by other documentation available on the website. Please check <http://www.e-framework.org> for the latest documentation.

Extracted from the Definitions document prepared for JISC and DEST by Dr. Daniel R. Rehak with contributions by the e-Framework Integrity Group and Editorial team.

This work is licensed under the Creative Commons Attribution-ShareAlike 2.5 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/2.5/au/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.



# Table of Contents

<b>1 Purpose.....</b>	<b>2</b>
<b>2 Introduction.....</b>	<b>2</b>
<b>3 A Formal Model for the e-Framework: Fitting the Pieces Together.....</b>	<b>3</b>
3.1 e-Framework Technical Description.....	4
3.2 e-Framework Technical Development Processes.....	9
3.3 e-Framework Application Design, Implementation and Deployment Processes.....	13
<b>4 Describing and Documenting Service Genres, Service Expressions, Service Usage Models and Service Patterns.....</b>	<b>16</b>
<b>5 Annotated Bibliography.....</b>	<b>17</b>
<b>6 Annex A: Service Design and Factoring Guidelines.....</b>	<b>19</b>
<b>7 Annex B: Principles and Practices in Designing Services.....</b>	<b>29</b>

# 1 Purpose

This document presents the core concepts and processes for describing and documenting the *technical* components, of the e-Framework. It is an edited extract of work commissioned by the e-Framework Partners and is intended to serve as an interim technical guide until the website is further developed with examples and supporting documentation. It is not intended to be a primary reference and is published in its current state as a means of providing some technical context to the initiative.

## 2 Introduction

In developing the e-Framework it has been of primary concern to focus on the definitions and descriptions of service usage models and services described through their service components, i.e., service genres, service expressions and service patterns. Including

- An overall presentation of the components of the e-Framework using the presented terminology (a formal model), including an outline of how to develop and document the technical components of the e-Framework (service usage models, service genres, service expressions) and how to design, implement and deploy applications and systems that are built using the components of the e-Framework.
- The (normative) format for a narrative (non normative) description of a service genre.
- The (normative) format for a narrative (non normative) description of a service expression.
- The (normative) format for a narrative (non normative) description of a service pattern.
- The (normative) format for a narrative (non normative) description of a service usage model.
- A service classification scheme.
- A set of concise definitions of terms.
- An annotated bibliography.
- Guidelines for designing and factoring service genres, service expressions, and service implementations.
- A collection of (normative) principles, practices and constraints applicable to the design of service genres and service expressions and to the development of service implementations and applications.

This document contains the core set of principles on which these documents and guidelines are to be developed.

**NB:** this document is not intended to exist in the long term as a single entity. Planned derivative works corresponding to most of the components above will be developed and released through the e-Framework web site. These derivative works will incorporate additional contextual content and will be living documents, updated as the e-Framework evolves.

**NB:** because of ambiguity, the word "service" is avoided throughout. "Service" is used as an adjective as in service genre, service expression, service pattern, service implementation, etc. When used without an adjective and otherwise unqualified, "service" implies a concept encompassing all of the qualified and unqualified terms used.

A collection of explanatory notes are distributed throughout the document tagged **NB**. Recommendations to the stakeholders are presented as notes tagged **REC**.

The words **MUST**, **MUST NOT**, **REQUIRED**, **SHALL**, **SHALL NOT**, **SHOULD**, **SHOULD NOT**, **RECOMMENDED**, **MAY**, and **OPTIONAL** in this document are to be interpreted as described in [RFC 2119].

### 3 A Formal Model for the e-Framework: Fitting the Pieces Together

Core components of the e-Framework are illustrated in Figure 1 below.



Figure 1: Core components of the e-Framework

The formal model presents the overall e-Framework in terms of its core technical components (service usage models, service genres, service expressions, service patterns), the overall process used to build the components of the e-Framework, and an explanation of how the e-Framework is translated into the design and development of service implementations, service toolkits and applications.

**NB:** this presentation is not complete. It contains only a basic explanation of the formal model. Additional illustrations and more explanations are needed before this material is presented to the public.

**NB:** the core components of service genre, service expression, service implementation, and service instance are inspired by the components of the Functional Requirements for Bibliographic Resources [FRBR] model: work, expression, manifestation, copy.

**NB:** the description does not address the deployment or operations of applications or of service implementations through their service instances.

**REC:** develop a more comprehensive presentation of the overall formal model, including contextual examples. Create additional illustrations, including showing how the various service components fit into the formal model and the process of developing applications from services, etc.

#### 3.1 e-Framework Technical Description

The SERVICES technical components of the e-Framework are illustrated in Figure 2 below. This diagram does not illustrate all of the technical components of the e-Framework. A more comprehensive narrative that outlines the relationships between the pieces of the model (including their multiplicities) follows. A more formal semantic model of the e-Framework is to be developed separately.

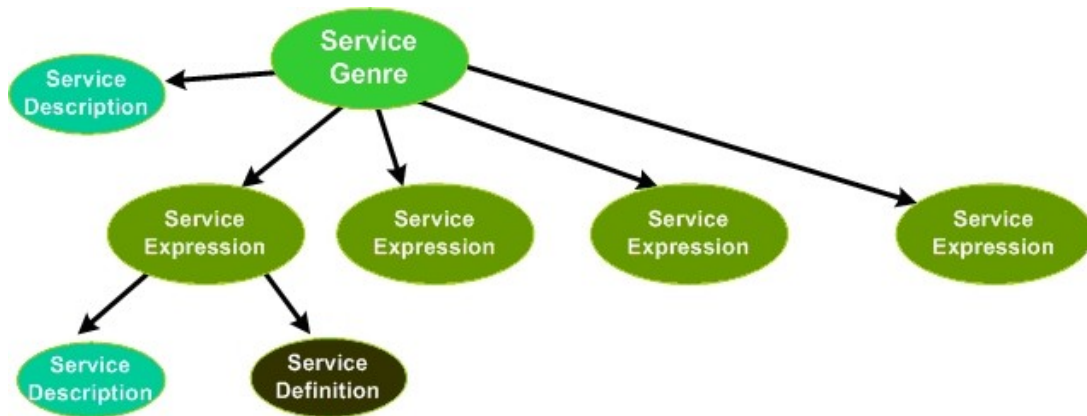


Figure 2: SERVICES

A broader view of the e-Framework and how the SERVICES technical components relate to standards and specifications development together with input from stakeholder communities is illustrated in Figure 3 below.

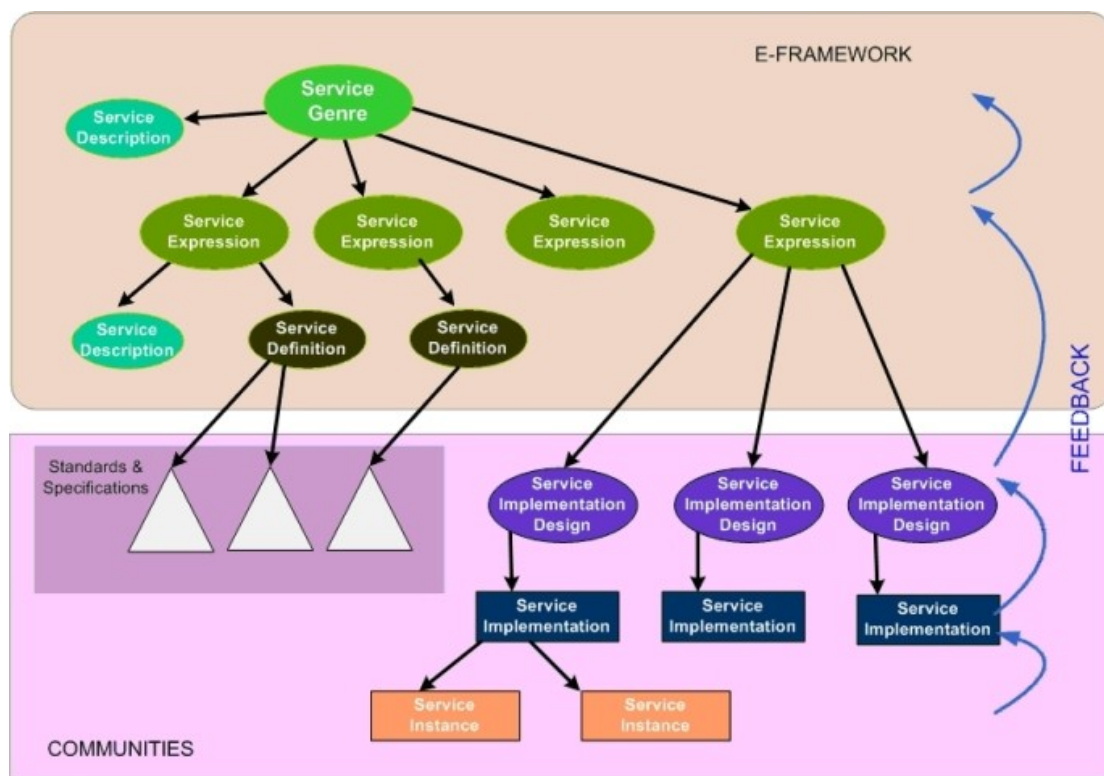


Figure 3: Development and Feedback processes

The technical description of the e-Framework focuses on the overall design and description of the service-oriented approach and technical components of the e-Framework, not e-Framework governance or policies. The collection of component descriptions is then used to document, build, deploy and operate service implementations and their service instances that in turn support tools, systems and applications for the provisioning of SERVICES or capabilities to the user community within the specific domains of the e-Framework.

**NB:** the formatted term SERVICES above has a very specific meaning, i.e., a general operational capability (e.g., a JISC production service), not an component (e.g., service genre, service expression) of the e-Framework.

**NB:** the presentation is informal; formal multi-value dependency statements could be developed. Dependency statements must be clearly specified to understand the “many” versus the “one” side of the relationships.

**NB:** the presentation does not detail the components to the same level as the web services ontology [WS OWL].

**NB:** governance or policy statements are not official; refer to e-Framework documents for formal procedures and policies.

- *e-framework:*
  - “facilitates technical interoperability within and across education and research through improved strategic planning and implementation processes”.
  - guided by a set of principles:
    - a service-oriented approach to system and process integration
    - development, promotion and adoption of Open Standards
    - community involvement in development of the e-Framework
    - open collaborative development activities
    - flexible and incremental deployment
  - covers a limited, enumerated set of domains:
    - learning and teaching, research, libraries, administration, IT-services
  - enables the development and deployment of applications:
    - each application may span one or more domains
    - the nature of each application, its design, deployment, etc., are not specified within the e-Framework
  - consists of a collection of core components (has many of each):
    - service usage models
    - service patterns
    - service genres
    - service expressions
  - enables the development of additional components, included in the e-Framework by reference (has many of each):
    - service implementations
    - service toolkits
  - has non technical components:
    - governance, policies, community, etc.

**NB:** the e-Framework does not document applications or service instances

- *service usage model:*
  - a service-oriented description of a collection of processes and workflows supporting at least one application, domain or business process
  - documented by a service usage model description
  - may cover one or more domains
  - may be related to one or more other service usage models
  - may be incorporated into other service usage models

- may overlap with the domains of one or more other service usage models
- is defined in terms of service genres xor service expressions

**NB:** XOR is "exclusive or". A single service usage model is defined either exclusively in terms of service genres or exclusively in terms of service expressions, but never a mix of both.

- may be defined in terms of one or more service patterns
- may be supported by one or more service toolkits
- references one or more open standards
- a core component of the e-Framework
- *service pattern:*
  - a reusable service-oriented design for a narrowly-scoped technical or business process
  - documented by a service pattern description
  - may be used in one or more service usage models
  - may be related to one or more other service patterns
  - may overlap with other service patterns
  - is defined in terms of (composed from) one or more service genres XOR service expressions and their relationships
  - a core component of the e-Framework
- *service genre:*
  - a generic or abstract capability to provide a set of related behaviours in support of a process
  - documented by a service genre description
  - may be used in one or more service usage models
  - may be used in one or more service patterns
  - may be used in one or more applications
  - may be related to other service genres
  - should not overlap in functionality with other service genres
  - is realised via a service expression through a service expression description
    - many possible independent service expressions
  - a core component of the e-Framework
  - the root of the collection of service components that refine the service genre
- *service expression:*
  - a specific set of related behaviours in support of a process that can be implemented with a defined set of technologies, interfaces and open standards
  - a specialisation of a single service genre
  - documented by a service expression description
  - defined by a service expression definition
    - behaviours and operations
    - interfaces

- may be used in one or more service usage models
- may be used in one or more service patterns
- may be used in one or more applications
- may be related to other service expressions
- should not overlap in functionality with other service expressions
- is implemented via a service implementation through a design
  - many possible equivalent, independent designs
- a core component of the e-Framework
- *service implementation:*
  - operational code providing the complete functionality specified in a service expression definition
  - the realisation of a service expression
  - not unique
  - is not deployed (is only deployed via a service instance)
  - may be used in one or more applications or domains (i.e., a common service)
  - instantiated and deployed through a service instance
  - may be instantiated or deployed one or more times
  - derived from a design
    - based on a service expression description
    - defined by the service expression definition
    - many equivalent, independent designs and corresponding service implementations
  - may be included in the e-Framework by reference
- *service instance:*
  - a deployed service implementation
  - functionally identical to all other service instances for the service implementation
  - operates under a set of policy rules
  - may be managed
  - not documented in the e-Framework (the stakeholder community does this)
- *service toolkit:*
  - a reusable collection of service implementations
  - is not deployed (is only deployed via a service instance)
  - may be distributed with auxiliary software (sample applications, sample data)
  - has no requirement to correspond to a service pattern; may encompass zero or many service patterns
  - may be used with one or more service usage models
  - may be used in one or more applications
  - should not overlap in purpose with other service toolkits
  - may use service implementations used in other service toolkits

- may be included in the e-Framework by reference
- *application:*
  - an ICT system that provides end-user functionality
  - built from a collection of deployed service implementations (service instances) and other components
  - follows zero or more service usage models
  - may follow a specific application design
  - may be instantiated and deployed one or more times
  - each instance operates under a set of policy rules
  - not documented in the e-Framework
- *service classification:*
  - a set of faceted classification schemes applied to any components of the e-Framework
  - a core component of the e-Framework

The problem space that the e-Framework encompasses is covered by a set of overlapping service usage models, some of which may lie partially outside of the problem space. The service usage models are defined using a set of service genres and service expressions. If the service genres and their corresponding service expressions are properly factored, they should not overlap. Structured collections (including compositions) of service genres and service expressions used within many different service usage models are identified as service patterns.

Users interact with applications, built from service implementations, designed following service usage models. Applications and their individual service instances may be deployed one or more times within an overall ICT computing fabric, and are governed and managed by identified policies.

**NB:** by implication, what applies to a service implementation also applies to a service instance. The description throughout the remainder is generally in terms of service implementation, as the same concepts apply to all deployed service instances.

## 3.2 e-Framework Technical Development Processes

The basic schematic view of the e-Framework technical development process is illustrated in the figure, inspired by the OASIS Reference Model for Service-Oriented Architectures [SOA RM]. A narrative that presents the process components and their relationships follows. Additional actions not present in the diagram are included in the narrative.

**NB:** the technical development process explicitly excludes governance, policies, management or community participation.

**NB:** there are overlaps between the technical description and the development process. The technical description focuses on descriptive statements. The development process focuses on actions.

**NB:** a formal methodology for developing the components of the e-Framework is not presented.

**NB:** service implementations, service instances, service toolkits, and applications are not core components e-Framework, and thus are not listed below (they are not documented in the e-Framework). These components are used when applying the e-Framework to build and deploy systems and ICT applications.

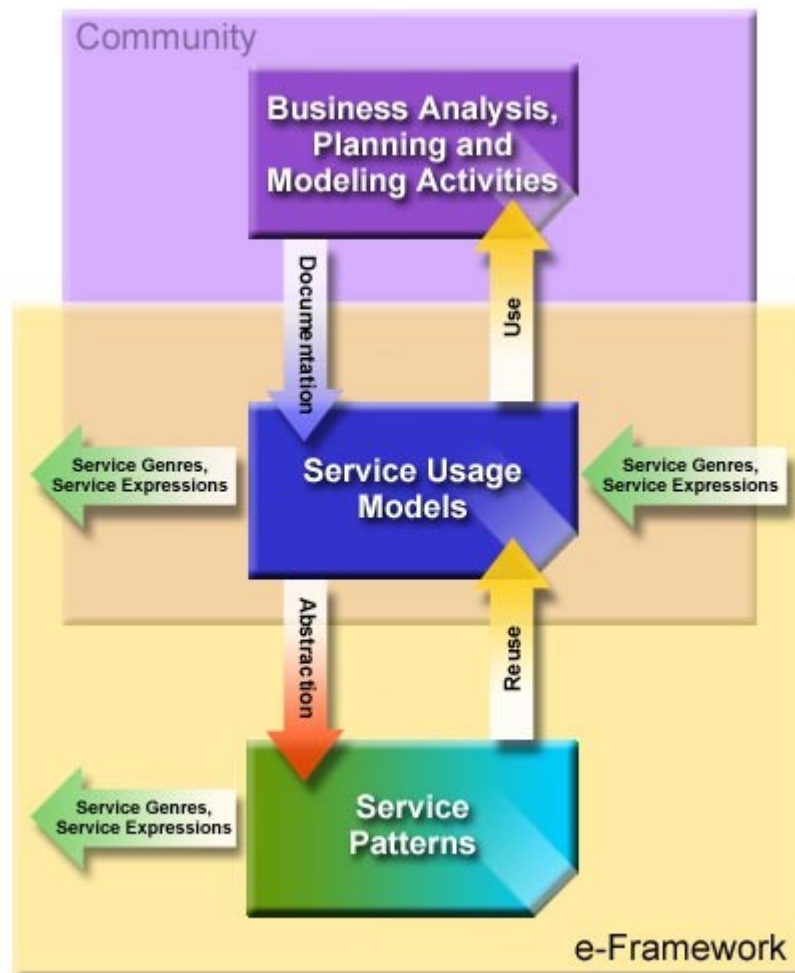


Figure 4: High-level view of Community Input and e-Framework Processes

- *e-Framework is:*
  - defined in terms of:
    - service usage models, service patterns, service genres, service expressions
  - may include by reference:
    - service toolkits
    - service instances
    - applications
  - described in a collection of documents:
    - methodology, definitions, descriptions, planning, service factoring, implementation guidance, and service implementation and application design and deployment
- *service usage model is:*
  - derived from:
    - user requirements, processes, data objects, workflows, standards
  - expressed in terms of:
    - service genres, service expressions, service patterns, other service

- usage models
  - standards, application profiles
- used to identify:
  - new service genres, service expressions
- used to design:
  - applications
  - service toolkits
- classified using:
  - service classifications
- documented:
  - following e-Framework standards and practices
- produced and registered by:
  - the community
- reviewed, maintained and published through:
  - the e-Framework
- *service pattern is:*
  - identified through abstraction from:
    - service genres XOR service expressions used in service usage models
  - used to refactor:
    - service genres XOR service expressions
  - used to design:
    - service usage models
    - applications
    - service toolkits
  - classified using:
    - service classifications
  - documented:
    - following e-Framework standards and practices
  - produced and registered by:
    - the e-Framework
  - reviewed, maintained and published through:
    - the e-Framework
- *service genre is:*
  - derived from:
    - service usage models
  - refactored from:
    - existing service genres
    - existing service patterns
  - used to derive:
    - service usage models

- service expressions
- applications
- service patterns
- classified using:
  - service classifications
- documented:
  - following e-Framework standards and practices
- produced and registered by:
  - the community
- reviewed, maintained and published through:
  - the e-Framework
- *service expression is:*
  - derived from:
    - service genres
  - refactored from:
    - existing service expressions
    - service patterns
  - used to design:
    - service implementations
    - applications
    - service usage models
    - service toolkits
  - classified using:
    - service classifications
  - documented:
    - following e-Framework standards and practices
  - produced and registered by:
    - the community
  - reviewed, maintained and published through:
    - the e-Framework

Service usage models are intentionally created de novo from requirements or identified from existing external sources. Service genres are intentionally designed de novo, identified as existing service genres or refactored from existing service genres according to a set of principles agreed and documented by the e-Framework. Service expressions are intentionally designed de novo, identified as existing service expressions or refactored from existing service expressions.

Service patterns are identified from the use and structure of collections of service genres XOR service expressions within service usage models and applications, and are abstracted and generalised from these service usage models and applications; they are not created de novo outside of existing examples. They are based on multiple, independent uses of a set of service genres XOR service expressions.

All components of the e-Framework are developed iteratively with refinements

made from actual use and community feedback. Multiple deployments and applications are encouraged to provide broad feedback.

### 3.3 e-Framework Application Design, Implementation and Deployment Processes

Developing the e-Framework has resulted in a set of documents that describe the process and methodology used in its development. These may or may not be further developed. The e-Framework also includes (normative) documents that are the technical descriptions of the collection of service usage models, service patterns, service genres and service expressions that are its core components. These descriptions are guided by the design, implementation and deployment processes and methods, and are used to design, implement, deploy and operate applications. A schematic of this development and deployment process is presented below, followed by a narrative presentation of the process used to create service implementations and service toolkits, and to deploy service instances and applications following the e-Framework's service-oriented approach.

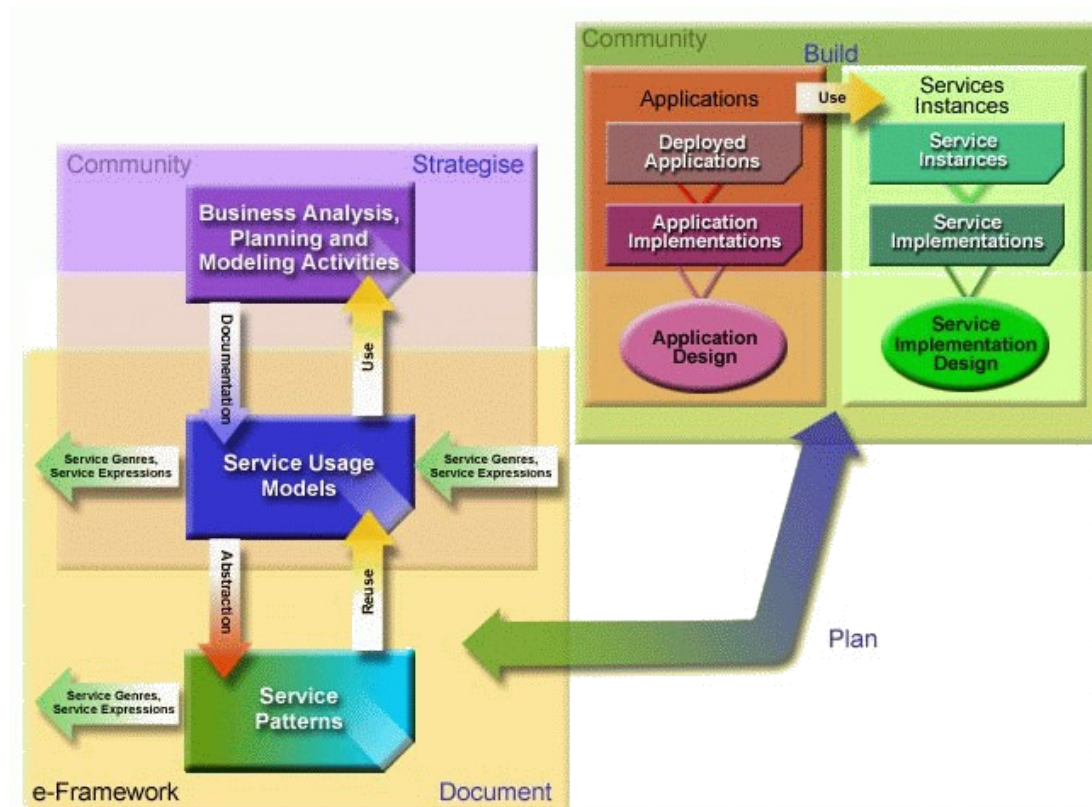


Figure 5: e-Framework Application, Implementation and Deployment Process

- *service implementation is:*
  - designed from:
    - a service expression description and service expression definition
  - classified using:

- service classifications
- implemented from:
  - the service implementation design
- available for:
  - deployment
- documented:
  - following community standards and practices
- *service instance is:*
  - a deployment of:
    - a service implementation
  - bound to:
    - a network end point
    - a collection of resources
  - classified using:
    - service classifications
  - deployed within:
    - managed environments:
      - as part of an application
      - under operational policies
  - uniquely identified in each deployment
  - documented:
    - following community standards and practices
- *(ICT) application is:*
  - designed from:
    - an application design using
      - zero or more service usage models
      - zero or more service patterns
  - implemented using:
    - the application design
    - service implementations
    - service toolkits
  - deployed:
    - one or more times
    - in one or more operational environments
      - under governance rules per environment
    - using physical entities (computational, storage, network)
  - bound to:
    - a network end point
    - a collection of resources
  - documented:

- following community standards and practices
- *service toolkit is:*
  - designed from:
    - a collection of service expression descriptions and service expression definitions
  - implemented from:
    - the service implementation designs
    - the service toolkit designs
  - associated with:
    - sample applications
    - supporting materials (test cases, sample data)
  - classified using:
    - service classifications
  - available for:
    - deployment
  - documented:
    - following community standards and practices

**NB:** the model does not permit a standalone service implementation. Service implementations are always integrated into an application, even if the application is only the proxy for a single service implementation, or if the application is deployed only once in the entire ICT computing fabric.

Neither the design and deployment of applications, systems and service toolkits nor the use of service implementations and service instances within these designs are defined within the e-Framework. The e-Framework documents the components of a service-based infrastructure for others to use to design, develop, deploy and operate applications using a service-oriented approach. An application developer will identify one or more service usage models and one or more service expressions (and their service implementations) that are relevant to the domain. The application developer is not constrained by the e-Framework in the selection of a particular approach to the design (software architecture) of the application or system, i.e., the e-Framework is silent on issues such as using a particular middleware approach (e.g., service bus, MOM). Multiple software designs may be developed for a particular application; design choices and external requirements will influence specific characteristics of an individual design.

**NB:** this discussion focuses on the design, development and deployment of applications. While the e-Framework does not prescribe processes, it will include operational guidance and supporting documents. These are not considered in the above technical discussion of using the formal model to create applications.

A design of the application is then implemented as an operational system (software plus hardware). Once implemented, the application may be deployed one or more times in one or more environments, e.g., a VLE application may be deployed at different institutions. An application might have one or more deployed instances, e.g., an administrative application might deploy both a production and a backup system instance. The technical characteristics of the e-Framework do not dictate which applications have multiple deployments or instances, or when a particular application is provided as single computational entity that can be utilised by all other applications and institutions across the ICT computing fabric.

## 4 Describing and Documenting Service Genres, Service Expressions, Service Usage Models and Service Patterns

The e-Framework encompasses a collection of service genres, service expressions, service usage models and service patterns. Each component in the e-Framework SHALL be described and documented in a common format, available as a template from the e-Framework website, and SHALL be submitted to the e-Framework according to the relevant processes and procedures.

## 5 Annotated Bibliography

**[Alexander]** Alexander, C., et al., *A Pattern Language: Towns, Buildings, Construction*, Oxford University Press, 1977, ISBN: 0195019199 - The source of "pattern" in software engineering.

**[e-Biz Patterns]** Adams, J., Koushik, S., Vasudeva, G., Galambos, G., *Patterns for e-business: A Strategy for Reuse*, IBM Press, 2001, ISBN: 1931180027

**[e-Framework]** *The e-Framework for Education and Research*  
<http://www.e-framework.org/> The e-Framework web site.

**[e-Framework Overview]** Olivier, B., Roberts, T., Blinco, K., *The e-Framework for Education and Research: An Overview*, Version R1, July 2005  
<http://www.e-framework.org/resources/eframeworkrV1.pdf>

**[FRBR]** *Functional Requirements for Bibliographic Records*, International Federation of Library Associations and Institutions, 1998, ISBN: 359811382X  
<http://www.ifla.org/VII/s13/frbr/frbr.pdf> Presentation of the FRBR model.

**[Foster]** Foster, I., "Service-Oriented Science" *Science* 6 May 2005 Vol. 308, no. 5723, pp. 814 – 817, DOI: 10.1126/science.1110411

A general discussion of how to build a "soa" information architecture for e-science. The discussion focuses on going from manual processes to automated ones. Critical problems to address are: interoperability (services and data), scale, management (including sustainment), QC (for data, metadata [including provenance], services), and the separation of concerns (domain-specific services, domain-independent services [management, registries, orchestration] and physical infrastructure).

**[GoF]** Gamma, E., Helm, R., Johnson, R., Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995, ISBN: 0201633612

The classic book on design patterns. It introduces the notation used to describe design patterns and includes a catalogue of patterns. The focus is on object-oriented software.

**[HLA]** IEEE Standard for Modelling and Simulation (M&S) High Level Architecture (HLA) — Framework and Rules, IEEE Std 1516-2000, The Institute of Electrical and Electronics Engineers, Inc., 21 September 2000, ISBN: 0-7381-2620-9

HLA defines a model of interacting simulation service implementations (federates) that operate within an overall simulation environment with a corresponding management and communications infrastructure (federation). Rules for federates and federations have applicability to services and applications.

**[Marca]** Marca, D., *Open Process Frameworks: Patterns for the Adaptive e-Enterprise*, Wiley-IEEE Computer Society Press, 18 August 2005, ISBN: 0471736112

Describes a process that can be used to map business requirements and workflows to a net-centric model.

**[OED]** Oxford English Dictionary, Oxford University Press, 2005  
<http://dictionary.oed.com/> - General definitions.

**[OPF]** Open Process Framework <http://www.opfro.org/>

OPF is a framework for defining software development; an alternative to the Rational Unified Process (RUP). The site contain an interesting glossary of terms, focusing on

software modelling (in the OO sense) and a descriptive template for describing components and processes similar to the format used to describe design patterns.

**[RFC 2119]** Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, IETF RFC 2119, March 1997 <http://ietf.org/rfc/rfc2119.txt>

Definitions of normative terms used such as SHALL, SHALL NOT, etc.

**[Ref Arch]** Gallagher, B., Using the Architecture Tradeoff Analysis Method to Evaluate a Reference Architecture: A Case Study, Technical Report, CMU/SEI-2000-TN-007, June 2000  
<http://www.sei.cmu.edu/publications/documents/00.reports/00tn007/00tn007.html>

Contains working definitions of architecture, reference architecture and presents an overview of a case study of how they are used in software engineering.

**[SOA Blueprint]** SOA Blueprints, OASIS Draft, 19 December 2005  
[http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=soa-blueprints](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-blueprints)

Presents a model for using patterns (service usage models in e-Framework terms) to define systems based on service-oriented architectures.

**[SOA RM]** MacKenzie, C. M., et al., Reference Model for Service-Oriented Architectures, OASIS Working Draft 11, 15 December 2005  
<http://www.oasis-open.org/committees/download.php/15966/wd-soa-rm-11.pdf>

OASIS's model of a set of concepts and their relationships to enable designers and developers to understand and describe service-oriented systems.

**[WS Arch]** Booth, D., et al., Eds., Web Services Architecture, W3C Working Group Note, 11 February 2004 <http://www.w3.org/TR/ws-arch/>

W3C's conceptual model of web services and their role within a larger SOA. It includes a set of definitions of components and conceptual models of the relationships among these components. Many of the core concepts are independent of a web orientation and can be applied to more general service-oriented architectures.

**[WS Gloss]** Hass, H., Brown, A., Web Services Glossary, W3C Working Group Note, 11 February 2004 <http://www.w3.org/TR/ws-gloss/>

A concise set of definitions used by W3C in describing their Web Services Architecture [WS Arch].

**[WS OWL]** Paolucci, M., Srinivasan, N., Sycara, K., OWL Ontology of Web Services Architecture Concepts <http://www.w3.org/2004/02/wsa/>

Formal OWL ontologies of the W3C Web Services Architecture [WS Arch].

**[WWW Arch]** Jacobs, I., Walsh, N., Architecture of the World Wide Web, Volume One, W3C Recommendation, 15 December 2004  
<http://www.w3c.org/TR/webarch/>

A summary of the core components and key constraints, principles and design choices underlying the web. The principles and practices can be applied in designing services.

## 6 Annex A: Service Design and Factoring Guidelines

The services within the e-Framework are designed, specifying behaviours and functionality from requirements. This design is then documented in the format described. Services in the service-oriented approach and in a web environment (not necessarily web services) differ from other programming models. Design guidelines for creating and (re)factoring services in the service-oriented approach follow, including:

- General characteristics of services (service implementations) within the service-oriented approach.
- A process and guidelines for service design and factoring (service genres, service expressions, service implementations).
- Design concepts for services (service implementations) and applications built upon and using these services and service implementations.
- Aspects of service and design (focusing on service expressions).

### A - General Characteristics of Services

Applications built following the e-Framework are used to access and manipulate a set of (managed) resources that are distributed over the network. The application's (idealised) model of the real world is represented through and encoded in the set of (stateful) resources, the state model. The resources are key information assets of the organisations that own, manage and maintain them. These organisations permit access to these resources via service implementations. This access provides a way of obtaining the information contained within the resources, or the means to specify how the service implementation is to manipulate the information, i.e., how a service implementation changes the resource's state. Key characteristics of the service-oriented approach that are important for designing these resources and the service implementations that access them follow.

**NB:** since this section describes operational capabilities, it is presented in terms of service implementations, not service expressions. The concepts, however, extend to the service expressions.

**NB:** by implication, what applies to a service implementation also applies to a service instance. The description is in terms of service implementation, as the same guidance is applied to all deployed service instances, and thus is expressed for the service implementation, not the service instance.

*stateful resources:*

resources (web resources) are *stateful*, i.e., they have a state. Resource values change over time through a series of events; each change is a transformation from one state to the next; the order of operations is meaningful; and the effect of the operations is reflected by the current value or state of the resources.

*service implementations manipulate state:*

service implementations provide the means used to manipulate the state of the resources. The service implementations essentially *own* the (access to) the resources. Manipulations change the resources, reflected by a change in their state.

**NB:** state may be manipulated independently by other means. There are no means to

limit such state changes.

*service implementations provide interfaces to resources:*

each resource is accessed and manipulated through the interface provided by the service implementation. The service implementation's messages provide the interface used to access and manipulate the resources.

*resource representations are unknown:*

clients do not know how resources are represented or manipulated. Representations and details of processing are generally private to the service implementation and hidden from view.

*communications are "by value":*

resources are not passed between service implementations; they remain under the control and access of the service implementations. The information that is passed to a service implementation through a message may include (1) control information (queries, actions) that specifies access or controls state manipulation and (2) documents (data). This data represents the values of some resource in some representation. In the service-oriented approach, the data is a copy or subset of the resource, not the resource itself. Alternatively, the message may include the identifier of the resource or data, not the resource. The service implementations must still then resolve and access the resource to manipulate its state; the actual resource is not passed to the service implementation.

**NB:** the resource value passed need not use the same representation or encoding as the resource itself.

**NB:** this differs from the object-oriented approach where resources, as objects, are passed in messages between methods.

*resource locations are unknown:*

clients do not know where resources are located in the network.

**NB:** in this use, services are themselves a resource.

*communications are via messages:*

messages provide the only way to communicate with a service implementation and for a service implementation to communicate with other service implementations.

*behaviours support transactions:*

individual messages and message sequences describe transactions that manipulate state. Transactions must either succeed and modify the state of the resources, or fail and leave the resource state unchanged. Transactions may be implemented through synchronous communications (the request and response are bundled in the communications model), or asynchronous communications (call backs, publish/subscribe). The service-oriented approach focuses on transactional manipulations of state, not the methods used to implement the required transactional model.

*stateless network infrastructure:*

at the application or service implementation level, network communication protocols, e.g., http, are often stateless.

These characteristics are combined to yield a service-oriented approach that uses transactional behaviours, implemented over stateless communications protocols, to provide stateful manipulations of private representations of resources where the resources are distributed over the network.

## B - Service Design and Factoring Process Guidelines

Service design, decomposition and factoring are complex and involve many tradeoffs. Decisions are often made under uncertainty. This is further complicated by conflicting goals (price, performance, capabilities) and the desire to build systems that are adaptable and long-lived, but function in an ever-changing implementation and deployment environment. Service design includes requirements gathering processes, design, implementation, deployment and refactoring. Key guidelines that cover each of these activities are listed below.

**NB:** these guidelines SHALL NOT be interpreted as a prescriptive software design methodology. Designers are free to choose any method or approach.

**NB:** by implication, what applies to a service implementation also applies to a service instance. The description is in terms of service implementation, as the same guidance is applied to all deployed service instances, and thus is expressed for the service implementation, not the service instance.

Develop requirements and constraints:

- Have target goals and metrics for price, performance (including QoS), capabilities.
- Differentiate required features from desired features.
- Identify known (not possible) future requirements and extensions.

**NB:** design for the known requirements only.

- Differentiate domain-specific from domain-independent concepts, context and content.
- Work from real-world (not theoretical) analyses.
- Know what entity has responsibilities and which are collaborators, i.e., what a service genre xor service expression does versus what service genre xor service expression it uses.
- Use nouns and verbs that correspond to documents and business functions, respectively, in the service-oriented approach to develop requirements.

Design the appropriate service genres and service expressions:

- Determine the appropriate granularity of service genres and service expressions, data, resources and behaviours.
- Decompose the problem space and factor service genres and service expressions using the service design principles listed in the annex titled *Principles and Practices in Designing Services*.
- Consider the service design aspects listed in the section titled *Aspects of Service Design*.
- Design for known requirements and extensions.
- Limit the complexity of service genres and service expressions.
- Map documents and business functions to the data objects and behaviours (messages) of the service expressions.
- The state model is encapsulated by the resources accessed and manipulated

through the service expressions (and their service implementations).

- Behaviours are the only way to cause change in the state model.
- Messages to service implementations are the only way to invoke the behaviour.
- Specify interfaces (messages, documents).
- Specify constraints on the implementations, but do not specify the implementation itself.
- Clients use only the published interfaces, never information about the implementation.
- Design for reuse.

Design, develop, deploy and manage service implementations and applications.  
[Marca]

- Identify the overall (business) processes, resources and workflows within the application.
- Identify the participants, their roles, rights and access requirements.
- Determine overall performance goals and metrics.
- Develop a service-oriented view of the processes.
- Factor the overall problem space into an overall service-oriented view.
- Identify (or design) appropriate service genres, service expressions and resources.
- Determine which service expressions will be open versus which will be closed.
- Reuse concepts, abstractions, resources, patterns, vocabularies, etc.
- Connect and compose the service expressions into the overall application design.

Then gather feedback on deployment and operations, and refactor as needed.  
Triggers for refactoring include:

**NB:** the list is not exhaustive.

**NB:** the list is not ordered or prioritised.

**NB:** use proxy and adapter service patterns to address changes and version incompatibilities as needed.

- change to the document(s) used by service expressions
- change to the messages, i.e., the service expression interface
- change to the format or structure of the document(s) passed in messages
- extension of and addition to service expression behaviours
- overly complex service expressions (or service genres)
- change in algorithm when behaviours are tied to specific algorithms
- change in underlying hardware, software, methodologies, etc., when the design is dependent on these
- failure to meet performance goals
- identification of tight couplings
- transitioning closed service expressions to open service expressions (or vice versa)

**NB:** a change in a design choice SHOULD NOT trigger refactoring.

## C - Design Concepts for Services

Much modern software is designed according to principles borrowed from object-oriented systems. Some of these key concepts can be applied and adapted to the service-oriented approach:

**NB:** by implication, what applies to a service implementation also applies to a service instance. The description is in terms of service implementation, as the same guidance is applied to all deployed service instances, and thus is expressed for the service implementation, not the service instance.

### *abstraction:*

abstraction provides a means to hide details. The objective is to simplify the model, focusing only on key aspects. Service implementations accessed through message interfaces abstract details of representation and processing. Layering of service implementations upon middleware, binding to specific languages, binding to communications protocols, etc., further abstracts the service genre and service expression concepts and models from use in service implementations and applications.

### *encapsulation:*

encapsulation groups data, resources, representations and behaviours into a single entity. A service genre or service expression (and its service implementation) is an encapsulation.

### *polymorphism:*

polymorphism defines how the same behaviour can be used with different data, e.g., polymorphic behaviour of "+": 1 + 2 yields 3; Monday + 2 yields Wednesday. Polymorphism has no direct analogy in the service-oriented approach; service expressions (and their service implementations) and their messages are designed to work with specific data or resources. A service genre may capture the same behaviour applied to similar data elements or documents. It is beneficial to develop a vocabulary of behaviours so that the same named behaviours applied to different representations, or that a message used in different service expressions or service implementations, are analogous.

### *inheritance:*

inheritance defines a structured hierarchal representation, where properties or attributes defined at one level are available to the lower levels directly, or these lower levels may override and change the properties. Inheritance has no direct analogy in the service-oriented approach; there is no direct refinement of service genres, service expressions or exposed representations.

These object-oriented concepts are augmented in the service-oriented approach with:

### *coupling:*

coupling describes the dependencies among components and how components are connected to each other, i.e., what knowledge one component has of other components. Components may be tightly coupled, with many dependencies, or loosely coupled, with minimal dependencies.

### *transactions:*

transactions join and coordinate multi-step processes into a single unit that either succeeds or fails.

Applying these key concepts to the service-oriented approach provides these guidelines:

*abstraction:*

- Hide details via abstractions.
- Describe service genres and service expressions in terms of the concepts, i.e., the documents or resources they manipulate, the behaviours.
- Provide additional elements that map the abstract concepts to their implementation details (service implementations, data models, resources, messages, bindings).
- Define interfaces, not implementations.

*encapsulation and factoring:*

- Encapsulate resources and their behavioural modifications in service expressions (and their service implementations).
- Factor to provide multiple encapsulations.
- Factor to group related behaviours on the same document or resource from unrelated behaviours.
- Factor to group related behaviours on the same document or resource from the same behaviours on different types of documents or resources.
- Factor those items that vary from those that remain the same.
- Factor to permit extensions without requiring modifications.

*composition:*

- Build service expressions (and their service implementations) through composition of other service expressions (and their service implementations) and other resources.
- Use composition as a replacement for inheritance.

*coupling:*

- Use loose couplings between interacting service expressions (and their service implementations).
- Minimise dependencies between components.

*transactions:*

- Enforce proper transactional controls and workflows.
- Minimise transaction dependencies and coupling between service expressions (and their service implementations).

## D - Aspects of Service Design

When designing a service expression or its service implementation, the following aspects should be considered and incorporated into the resulting design and documented in the service expression description. Where appropriate, the design decisions and design choices around these aspects SHOULD be included in the service expression description.

**NB:** the list is not exhaustive. Designers SHALL augment the list of aspects as necessary.

**NB:** while this list focuses on service expression design, many of the same aspects SHOULD be considered when developing a service genre, service usage model or service

pattern.

**NB:** there are additional aspects of software design that are not specific to service design that are not included here (e.g., use of declarative models, consistency in approach, symmetric models).

**NB:** the list is not ordered or prioritised.

*granularity and document orientation:*

service expressions (and their service implementations) may vary in size (number of behaviours) and complexity (of interfaces and of the behaviours provided). The service-oriented approach favours a smaller number of larger-grained service expressions, each providing a collection of related, meaningful (complex) business functions. Service expressions may provide behaviours that operate on and at the level of (complex or complete) documents, or lower-level operations on individual fields and elements of the data representations of resources. The service-oriented approach favours a document-oriented functional model. Service expressions (and service genres) should be designed around manipulating document-oriented resources through business-oriented functions and behaviours. Service expressions should not be designed to provide direct manipulations of (underlying, internal) data representations. Service expressions should not be used to implement CRUD interfaces (create, retrieve, update, delete) on data representations.

*interoperability:*

interoperability between service implementations or between clients and service implementations motivates the service-oriented approach. In an open, loosely connected system, interoperability is achieved through the use of standards (standards, specifications and application profiles). Identify, use or adapt standards when feasible. Selection of, and conformance to, these standards SHALL BE documented.

**NB:** conformance statements include exceptions, extensions, non conformance.

*scale:*

service expressions (and their service implementations) may be required to function over a range of use (scale). The underlying resources may range in scale (e.g., size, number, complexity). For example, will the service implementation have limited deployment or it will be widely deployed and accessible? Will the expected number of resource instances be small or very large? How might the service implementation need to change as scale of use and deployment change? Assumptions about the scale of service implementation deployment SHOULD be documented. The design of the service implementation SHALL consider scalability.

*latency:*

responses to messages and requests are not instantaneous. While the e-Framework is not designed for real-time applications, latency should be considered in the design of service expressions, service implementations and applications. What is the impact of a delay in response? What response times are acceptable? Expected and acceptable performance criteria SHOULD be documented, including information about behaviours when criteria are not met. Alternative strategies to meet performance criteria SHOULD be documented.

*incremental update of components:*

individual components (resources, applications, service implementations,

standards) may be incrementally updated or versioned. A service expression (and its service implementation) SHOULD be designed so that other components or elements can be updated without breaking any service implementation. When a service expression or service implementation is dependent on a particular version or feature such that an update to that one component will break the service implementation, this dependency SHALL be documented.

**NB:** an acceptable behaviour may be that the service implementation indicates that it cannot interoperate with a particular version of a document, resource or other service implementation. A service implementation should not fail when components are updated.

*reliability:*

there are many sources of unreliability in a service-oriented approach. Communications and messaging may be unreliable (messages may not be delivered); service instances may be unreliable (they may fail to respond, not be accessible); hardware may fail. A requestor SHALL be designed assuming the provider will not be reliable. Target performance goals for service expressions (and their service implementations) and providers SHOULD be documented. Single points of failure should be avoided, but identified when present.

*lack of shared state between requestor and provider:*

the entire state model is hidden. Service expressions (and their service implementations) expose behaviours that may not directly map to the underlying representations or state. State is accessible only via service implementations and it is difficult to obtain all of the state with large-grained service expressions. Thus different service instances and applications may have different models of state or assumptions about the current values in the state model. Service expressions (and their service implementations) shall be designed without relying on a shared state model. Service expressions (and their service implementations) should be designed assuming limited and potentially faulty state information. Approaches to providing shared state should be documented.

*transactions:*

business operations and workflow processes require coordinated series of messages, each of which changes the state model. Different concurrent access patterns are performing independent and potentially conflicting and incompatible changes to the state model. Transactions provide the mechanism to coordinate these messages and provide consistency in the state model. Reliability concerns imply that transaction patterns are not failsafe. The distributed network model and network latency implies that transactions may be long-lived. Transactional behaviours should be ACID (atomic, consistent, isolated, durable). Transactional behaviours and assumptions should be documented. Approaches to supporting transactional behaviour should be documented.

*concurrent access:*

service implementations (providers) may be accessed concurrently from many clients (requestors). Service expressions (and their service implementations) shall be designed for concurrent access. Different design solutions will impact latency, reliability and requirements to support data transactions. Independent of how the service expression is designed for concurrent access, the design of the service implementation SHALL consider concurrency. Approaches to providing concurrent access should be documented.

*message orientation:*

service expression interactions (and their service implementation interactions) are defined in terms of messages, not in terms of how the service expression is

implemented or structured. Implementation details should not be exposed via message interfaces. Clients should not be dependent upon knowledge of internal structure and behaviour of the service implementation.

*unreliability of transport:*

messages and service implementation interactions are implemented upon a network transport layer that is inherently unreliable. Service expressions (and their service implementations) should be designed assuming unreliable transport.

*security:*

access to resources and to service implementations may need to be secure and controlled. Service expression descriptions shall describe security requirements, e.g., what to secure, what are the policies for control, what are the authentication requirements and models, what are roles and permissions, what is confidential, where is nonrepudiation required, what needs to be encrypted, when audit controls and records are needed, where rights need to be managed, etc. Security solutions and approaches should be documented.

*privacy:*

service expressions (and their service implementations) may have access to private or confidential data. Privacy concerns SHALL be documented. Approaches to ensure privacy should be documented.

*policy:*

service expressions (and their service implementations) are managed under a collection of policies, including security, trust, privacy, availability. All governing policies shall be documented. Approaches for describing and enforcing policies and managing service implementations should be documented.

*QoS:*

quality of service addresses target performance goals, including reliability, availability, latency, response time. All QoS goals and measurement metrics should be documented. QoS constraints (goals that must be met) shall be documented. Approaches to meet QoS requirements should be documented.

*trust:*

service expressions may have or assume trust relationships with other service expressions. Trust requirements and assumptions shall be documented. Approaches to supporting trust relationships should be documented.

*provenance:*

requirements for provenance of data and resources should be documented. When required, approaches to support data and resource provenance should be documented.

*discoverability and description:*

service expression descriptions should be complete in exposing all public information needed to use and understand the service expression. Formal, machine-processible service expression definitions SHALL be provided.

*operations and sustainment:*

deployed service implementations, particularly those that are used by multiple applications, may require production-level support and operations (reliability, system and data backup, scale, operational procedures). Requirements for sustainment and operational plans for service implementations should be included in the service expression and service implementation design. Sustainment and operations shall be considered in designing and deploying applications.

*service complexity:*

service expressions should have realistic complexity and extensibility. Factor service expressions to limit complexity. Limit extensibility to known requirements. It may be useful to factor special cases into separate service expressions.

## 7 Annex B: Principles and Practices in Designing Services

A list of considerations in designing service usage models, service genres, service expressions, service patterns, service implementations, service toolkits, resources and applications. Principles, practices and constraints are based in part on [WWW Arch, HLA].

**NB:** the list of issues is incomplete and unordered.

**NB:** a structured organisation may be needed.

**NB:** this is a more formal (normative) list than presented in the section titled *Service Design and Factoring Process Guidelines*.

**NB:** by implication, what applies to a service implementation also applies to a service instance.

*unique identification of resources:* [PRINCIPLE]

resources SHOULD have unique identifiers.

*identifier opacity:* [PRACTICE]

anyone using an identifier SHOULD NOT infer properties of the identified resource from the identifier.

*version information:* [PRACTICE]

descriptions and specifications (service usage models, service genres, service expressions, service patterns, data models, schemata, standards) SHOULD include version information.

*extensibility mechanisms:* [PRACTICE]

descriptions and specifications (service usage models, service genres, service expressions, service patterns, data models, schemata) SHOULD include extensibility information.

*extensibility conformance:* [PRACTICE]

extensions to specifications (service usage models, service genres, service expressions, service patterns, data models, schemata) MUST NOT interfere with conformance to the base specification.

*separation of content from presentation:* [PRACTICE]

specifications (service usage models, service genres, service expressions, service patterns, data models, schemata) SHOULD separate content from both presentation and interaction.

*communications reliability:* [CONSTRAINT]

network communications may be unreliable. All designs and implementations MUST consider communications reliability.

*service communications:* [PRINCIPLE]

within an application, all communications with service implementations SHALL be through their message interfaces.

*abstractions:* [principle]

implementation and representation details should be hidden through the use of abstraction.

*coupling:* [practice]

components should be loosely coupled.

*granularity:* [practice]

service expressions and service implementations should be large grained.

*document orientation:* [practice]

service expressions and service implementations should be document oriented.

*transactions:* [principle]

within an application, transactional behaviour shall be provided and enforced.