Net-Centric Services Framework

Version 2.1

2009-10-01

Authors
Robert Palmqvist, Saab
Lars Schylberg, Saab
Allen Jones, Boeing
Jerry Sonnenberg, Harris

Developed on behalf of the
Network-Centric Operations Industry Consortium (NCOIC(TM))

Approved for Public Release
Distribution Unlimited
NCOIC-NIF-NCSF-20091005v2.1

Change Record

#	Date	Change	Person
10	5/15/2008	LASC-ROPA-ALJO Review in Stockholm	Lars Schlyberg
11	26-May	Minor changes discussed with Lars	Allen Jones
12	9-Jun	Incorporation of Alaka Shivananda changes	Allen Jones
13	11-Jun	Minor changes discussed with Kobie	Allen Jones
14	16-Jun	Minor changes discussed with Lars	Lars Schlyberg
15	28-Jul through 8/7	Changes resulting from discussions with John Yanosy plus experience with judging a major program. Provider and Producer were used for the same actor, and so producer was replaced by provider.	Allen
16	8/7 through 8/10	This version incorporates all the proposed changes in 15 and reorders the principles section with an eye to combination of certain sections	Allen
17	20080820	Processing of all change requests and comments from Allen.	Lars
18	20080822	Allen and Lars agreed on the last changes in 2.2 to remove the last bullet list	Lars & Allen
19	20080907	Ginger Carney added appropriate document markings.	Carney
20	20080922	3.1.12 security principles section added and rewritten text after comment from Mark B. at the Plenary.	Allen & Lars
21	20081107	Incorporation of Initial Formal Review CRs prior to ballot	Allen
22	20081112	Incorporation of Final Formal Review CRs prior to ballot	Allen & Lars
23	20081121	Reconciliation of CR spread sheet with v.22 above	Allen
24	20081126	Edits made by the NIF WG during review of CR implementation by Allen and others	Tim
25	20090430	Edits for v2	Jerry Sonnenberg
26	20090506	Edits for v2 – add additional Annexes	Jerry Sonnenberg
27	20090527	Edits for v2 – add additional Pattern Check Lists items and text for Section 5 - Process	Jerry Sonnenberg
28	20090615	Final edits from Brussels Plenary prior to Formal Review	Jerry Sonnenberg
29	20091001	Incorporate accepted changes from formal review	Jerry Sonnenberg

Table of Contents

1 Introduction

The Network Centric Operations Industry Consortium (NCOIC™) provides guidance for Network Centric Operations (NCO) interoperable systems. The NCO Interoperability Framework (NIF™) provides an organizational construct and repository for enabling this guidance. NIF is a framework that assists industry in designing interoperable systems. NIF is based upon standards, including patterns, principles, and processes. Intended readers of this document, described in the NIF Solution Description (NSD)[1] , are those concerned with services: system architects, engineers, acquirers, integrators and testers.

NIF identifies the service approach as a key interoperability concept. It encourages service orientation. It fosters Service Oriented Architecture (SOA) as the preferred approach for Information Systems. Services enable information sharing by connecting people/systems that have information with people/ systems/ nodes that need information. For people/ systems/nodes who have information, SOA provides a method for information advertising and delivery. For people/ systems/nodes who need information, SOA provides methods to find/receive information. At lower, or infrastructure, levels, SOA provides key infrastructure capabilities. At upper, or operational, levels, SOA supports capabilities for key mission areas. Services visible on the network are available to all nodes and enable consumers and providers to use common applications.

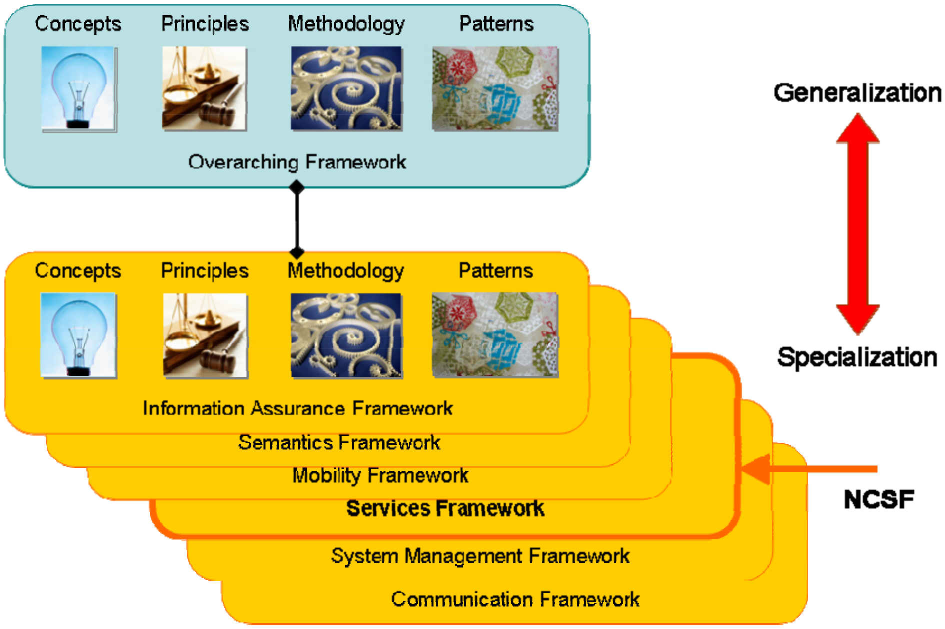


Figure 1 – Relations between NIF Overarching and Specialized Frameworks, with Emphasis on this, the NCS, Framework. Source: NCOIC

[1] See the NCOIC NIF Solution Description-Reference Manual (NSD-RM), version 1.0

The Net-Centric Services Framework is contained in the NIF overarching framework, consistent with the NIF structure requirements: Concepts, Principles, Patterns and Processes[2]. Other planned specialized frameworks shown in Figure 1 define additional specialized aspects of the information communication space. These frameworks also will be organized along interoperability concepts, principles, patterns & processes. While separation of architectural concerns facilitates decomposition, any real world implementation of systems will involve some coupling of these different frameworks to each other.

The NIF describes architectural constructs supporting net-centricity and interoperability at a high level. The Net-Centric Services Framework complements the NIF at a practical level. There are NCOIC standardized frameworks that provide guidance on realization of complete systems solutions; there are others that describe service oriented architecture approaches for specific system solutions, and still others that address design of specific services. This framework complements the NIF primary focuses, which are to give guidance leading to interoperability between systems & their services, and promote net-centric[3] architectures.

The scope of this document includes information communications systems functions contained in each system node. These nodes may be contained in a system of systems, purpose built system, or platform. The scope includes enterprise to enterprise interoperability, a unique consideration for NCOIC. Although services addressed in this document are primarily software; they interface with the person(s) providing or needing information and hardware that contains visual, tactile and processing interfaces to human users.

The organization of this document mirrors the description of a framework contained in the NIF: Section 2 describes the concepts; Section 3 contains the principles, separated into Service Principles and Net Centric Principles; Section 4 describes pattern requirements; Section 5 contains the processes used when implementing a Net-Centric Services Framework.

[2] NIF version 2 contains information on the "overarching framework" and how the various frameworks interrelate to each other.

[3] The Essence of Net-centricity, AFEI DS3WG, October 12, 2006, Hans Polzer, Lockheed Martin Corp

2 Concepts

Concepts explicitly capture the necessary knowledge about the framework. Concepts are described using Definitions, Attributes, Structures, Relations, and Constraints. [4]

The service information system approach can be exemplified as an individual or system needing information from another individual or system having that information. Finding published information that can be delivered in a timely manner is the service premise.

Implementation of such a system is similar to the Plain Old Telephone Service (POTS) operation. A directory, often known as the Yellow Pages, is employed; it is known as a “registry” in the service context. In the context of general use, finding the edition, or version, of the yellow pages that is useful and can lead to a supplier who can deliver in a timely manner is not an unusual problem, the analogy of which leads to the structure of a Service Oriented Architecture.

2.1 Service Framework Concepts

Concepts are described by an information structure that identifies three concepts central to this framework: (1) Service Orientation, (2) Service Oriented Architecture, (3) Service.

> **Service Orientation** is a way of thinking in terms of services, service-based development and the outcomes of services.
>
> **Service-Oriented Architecture (SOA)** is an **architectural style** that supports **service orientation**.
>
> **Service:** (1) A logical representation of a repeatable business activity that has a specified outcome, (2) self contained (3) or may be composed of other services, (4) a “black box” to service consumers.

2.1.1 Service Orientation

Service-orientation has emerged as a distinct design paradigm that, when applied, fosters creation of automation logic in the form of services. The service-orientation paradigm is comprised of a set of common design principles that realize specific service design characteristics in support of the strategic goals associated with service-oriented architecture.

Like previous design paradigms, service-orientation provides a distinct means of achieving a separation of concerns. This theory is based on the notion that it is beneficial to break down a large problem into a series of individual concerns. This allows the logic required to solve the problem to be decomposed into a collection of smaller, related pieces. Each of these pieces addresses a concern or a specific part of the problem. Separation of concerns has been implemented in different ways with different development

[4] Excerpted from NIF version 2.

platforms. Object-oriented programming and component-based programming approaches, for example, achieve a separation of concerns through the use of objects, classes, and components.

Service-orientation may be viewed as a mechanism to realize a separation of concerns. The service-orientation principles provide a way of supporting this separation while achieving a foundation paradigm upon which contemporary SOA characteristics may be built. The term “service-oriented design” is used when referencing a process for designing SOA services. When used in general terms for solution logic to a design approach for SOA services, service-oriented design is synonymous with service-orientation.

Service-orientation is described by the following combination of distinctive architectural features:

- A service design mirroring real-world business activities that comprise enterprise business processes
- Service representation that utilizes business descriptions providing context (i.e., business process, goal, rule, policy, service interface, & service component) implementing services using orchestration
- Unique infrastructure requirement translating into implementations that use open standards for interoperability and location transparency
- Environment-specific implementations which are constrained or enabled by context, and must be described within that context
- Governance of service representation and implementation
- A “Litmus Test"[5], determining the quality of a service.

2.1.2 SOA Definition

A Service Oriented Architecture is a paradigm that uses loosely coupled services to support business processes and users. Loose coupling is a condition wherein a service acquires knowledge of another service while still remaining independent of that service. SOA exchanges are interactions initiated by service requests between service consumers and service providers. Both the consumer and the provider make their requirements explicit and theoretically make no assumptions about each other except for what may be discerned through a registry or other means of discovering accessible services. Resources on a network in an SOA environment are made available as independent services accessed without knowledge of their underlying platform implementation. These concepts can be applied to business, software and other types of provider/consumer systems. [6]

The Organization for the Advancement of Structured Information Standards (OASIS) defines SOA as:

> *“a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains”*[7].

[5] Litmus Test – a simple Pass/Fail test that can be applied to examine suitability

[6] Adapted from Wikipedia, Service-oriented architecture, 07:57, 30 April 2007.

[7] Organization for the Advancement of Structured Information Standards, Reference Model for Service Oriented Architecture 1.0 OASIS Standard, 12 October 2006 http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf

Richer definitions are discussed below, and are derived from The Open Group work. Note that an architecture that does not follow all the SOA defined precepts in this section may still have a “Service Orientation’.

2.1.2.1 SOA Style Characteristics

The architecture style defining a SOA describes a set of patterns and guidelines for creating loosely coupled, business-aligned services that provide unprecedented flexibility in responsiveness to new business threats and opportunities. This flexibility is achieved through separation of concerns between description, implementation, and binding. In the distributed computing world, one computer agent’s needs are often met by another owner’s computer agent. There may not be a one-to-one correlation between needs and capabilities; satisfying a need may require combining several capabilities. A single capability may address more than one need. The SOA value is to provide a framework for matching needs to capabilities and combining capabilities to address needs. OASIS has defined the following SOA style parameters (or characteristics) that are excerpted below:[8]

2.1.2.1.1 Visibility

Visibility refers to capacity for those with needs (consumers) and those with capabilities (providers) to locate and potentially access each other. This is done by providing descriptions of functions and technical requirements, related constraints and policies, and mechanisms for access. Descriptions must be in a form in which their syntax and semantics are accessible and understandable. Factors that affect visibility are:

- Mode of Operation (Operational vs. Maintenance)
- Quality of Service
- Identity
- Information Assurance (IA)
- Safety
- Access Management
- Node Configuration
- Inter Federation Policy
- Geography

2.1.2.1.2 Interaction

While visibility introduces the possibility to match needs to capabilities and vice versa, interaction is the activity of using a capability. An interaction involves information exchanges and invoked actions, facets of which form a path between consumers and providers. This permits them to interact and provides decision points for policies and contracts. The purpose of using a capability is to realize an effect(s). An interaction is “an act” as opposed to “an object”. The result of an interaction is a set/series of effects. This effect may be return of information, or change in the state of those interacting. Interactive services are those where every interaction results in one or more effects such as a state change or return of information.

[8] OASIS Reference Model for Service Oriented Architecture 1.0, Committee Specification 1, The Organization for the Advancement of Structured Information Standards (OASIS), 2 August 2006

2.1.2.1.3 Effect

Real-world effects are described in terms of changes among those involved in the current execution context, and possibly shared by others. Expected real-world effects form an important part of the decision on whether a particular capability matches similarly described needs. At the interaction stage, description of real world effects establishes expectations for users of the capability, so that real world effects of provided services are visible to, and understood by, intended service consumers.

2.1.3 Service

The term service[9] can be defined differently within the concept of service orientation, within SOA and even between different SOA frameworks. A Service is an object containing a set of functionality provided by one entity for use of other entities independent of underlying technology. The OASIS Reference Model for Service Oriented Architecture v1.0 is an example of a way of defining service within a specific SOA framework:

> *"Service – The means by which the needs of a consumer are brought together with the capabilities of a provider."*[10]

OASIS published the following Axiom set describing the basis for services within a SOA:

1. A Service is a set of functionality provided by one entity for use of others.	6. A Service Description is comprised of three logical parts:	7. A security policy is a specialized type of Service Description policy noted above.
2. Services are conceptually autonomous (self sufficient) and opaque (independent of underlying technology) in nature.	Data Model - The logical expression of a set of information items associated with the consumption of a service or services;	8. Service Policy may mandate security requirements to be met, and *if they are not, interaction may be refused.*
3. There is no need to make architectural distinctions between *services consumed as part of a process vs. ones that are not.*	Policy - Assertions and obligations that service consumers and/or providers must adhere to or provide, and;	9. A null security policy is still logically considered a policy.
4. There is not always a one to one correlation between "on the wire" requests to invoke a service and service responses being consumed.	Contract (and/or offer thereof) - the syntactic, semantic and logical constraints governing on the use of a service.	10. A Service Description is advertised to consumers on a fabric to make it discoverable.
5. Each logical Service has *exactly one canonical Service Description.*		11. *Discovery does not constitute authorization* to execute against the service.

[9] Refer to ANNEX A – Comparison of for a comparison of additional definitions of the term "service".
[10] OASIS Reference Model for Service Oriented Architecture 1.0

The first axiom is a definition of a Service similar to that adopted by OASIS for SOA. Axiom two, independence of underlying technology, is also a key factor in making diverse applications interact. The third axiom gives the ability of a system to deliver product, key for interactions such as software updates. Axiom six is the key to the definition of the service: Data, Policy, and Contract. Axioms 7-9 address security. The last two axioms address discovery, a unique SOA factor.

2.2 Net-Centric Services Framework Concepts

Net-centricity addresses the concept that

> *-- the consumer of information should be provided or be able to get information needed in a timely fashion when ever or wherever the consumer may be.* [11]

Net-centricity seeks to leverage evolving networking capabilities to improve global enterprise efficiency. Starting with this goal in mind, the properties of Net-centricity can be summarized as:

Environment awareness is the ability to both passively acquire useful information (sensors) and to actively seek (discovery) new information (actuators).

Mutual sustainment, or interdependence, is the ability to both share with other systems and to use their capabilities through common policies that support a will-to-share, thus enabling systems to team together to support their individual and joint missions.

Autonomy is the ability to continue operations when partially or completely disconnected from the network during short or long periods of time.[12]

Trust supports the ability to share information and is the expectation that the right information will be delivered to the right person at the right location at the right time.

From a technical point of view, the NCOIC Interoperability Framework and ancillary documents such as this support these properties using core principles of Network Centric Operations described in the next section.

The following Figure 2[13] provides context for the discussion in Section 3 on the principles of a Network-Centric Services Framework. The entities described in this figure are also used in a series of related figures in Annex C. Annex C contains more detail on the relationships among the entities. These relationships are documented in SysML[14] diagrams and complement the text in Sections 2 and 3. Most of the principles described in Section 3 have either an entity or an entity relationship within this diagram. Each principle that is illustrated in Figure 2 has text referencing the figure in its section.

Future versions of this framework document may expand Figure 2 or provide additional figures that illustrate additional principles and their relationship to each other.

[11] Excerpted from NCOIC Interoperability Framework, Version 2.

[12] In this context, Autonomy refers to self management, differing from Autonomicity which refers to self governance.

[13] This figure replaces Figure 2 from the NCSF v1 which dealt with entities relating to Service Choreography. As a more general entity diagram, it is intended to provide context for all the entities discussed in Section 3.

[14] OMG Systems Modeling Language http://www.omgsysml.org/

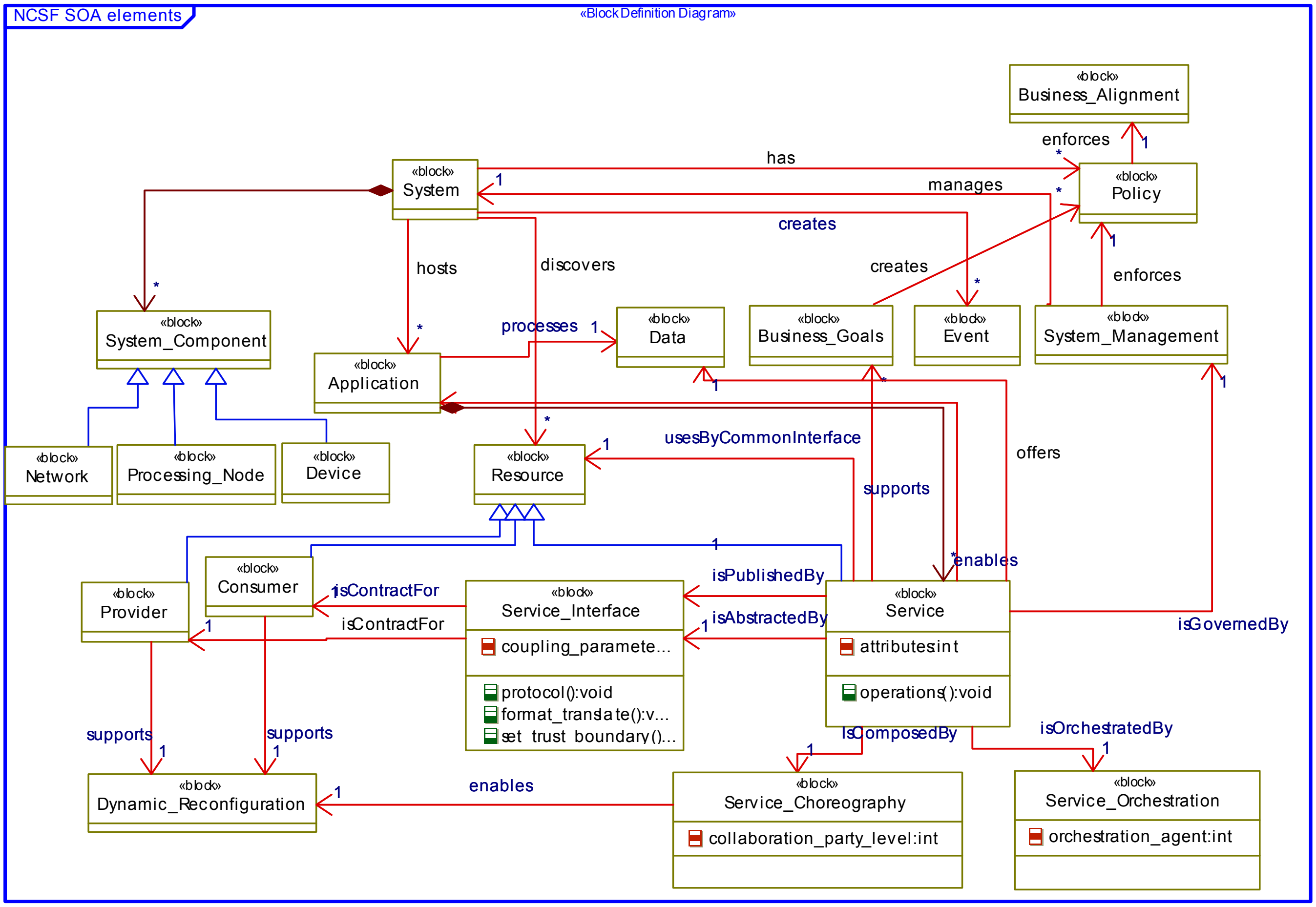


Figure 2: Relationships among Elements of an SOA and Net-Centric Principles. Source: NCOIC

3 Principles

This section identifies and describes Net-Centric Service oriented principles.

Net-Centric Services Principles are the overall requirements, goals, tenets or best practices that should be applied to foster Net-centricity. Net Centric Services Principles serve to guide development of services, service specifications, service patterns, and service-oriented architectures that are net-centric and conform to NIF architectural principles.[15]

General principles of service orientation described in Section 3.1 are the following:

- Service Description
- Service Interface
- Service Contract
- Service Abstraction
- Loose Coupling
- Service Interoperability
- Service Composition
- Service Orchestration
- Service Choreography
- Discovery
- SOA Governance
- Location Transparency
- Security

The Net-Centric Service principles are described in section 3.2. Complementary to the above general principles, these Net-centric principles more specifically address tenets of Net Centricity for Service Oriented Architecture. These are the following:

- Execution Platform Independence
- Explicit Service Business Model
- Environment Monitoring, Assessment, & Adaptation
- Reuse
- Net-Centric Data
- Dynamic Configuration
- Explicit Service Scoping
- Autonomicity
- Robustness (Graceful Degradation)

Not all of the principles described are at the same level. For example, service interfaces and service contracts may be considered as a subset of a service description although used independently. The Service Orientation Principles may overlap with the Net-Centric Services Principles due to the emphasis taken to separate the two sets of principles. If one were to merge the two lists, this overlap vanishes.

The syntax for describing Principles is contained in NIF RM section 2.3.1. The **Identifier** syntax is repeated here for convenience:
<source document> . <framework name> . "principle" . <category> . <principle name>

[15] Excerpted from NIF version 2.

In all cases, the <source document> for principles in this specialized framework, the Net-Centric Services Framework (NCSF), is the NIF, denoted by "Nif".
In all cases, the <framework name> for principles discussed in this document is NCSF, denoted by "Ncsf".

This framework discusses two types of <category>, "service-orientation" and "net-centric-service".

For example, the Service Description principle discussed in section 3.1.1 would be identified as: Nif.Ncsf.principle.service-orientation.service-description.

The principles described in Section 3 are derived from the NIF Solution Description Reference Manual (NSD-RM), but are presented in this Specialized Framework with specific context. Because of this, the principles in Section 3.1 are complementary, but with a different focus, to those in Section 3.2. The Pattern Check List items are another set of items that illustrate the principles in the NCSF, but from the perspective of creating net-centric services patterns. Annex B contains a mapping of the NIF principles to the NCSF section-specific principles.

3.1 Service Orientation Principles

Services facilitate information sharing and provides ability to mix and match information system components. Service Orientation allows users to:

- Find information, services, and applications when needed, independent of physical location;
- Understand and employ the discovered information and tools, no matter what platform supports them, whether local or remote; and
- Evolve a processing environment for commercial use without being constrained to a single vendor's offerings.

3.1.1 Service Description

Identifier	Nif.Ncsf.principle. service-orientation.service-description
Source	SOA for NBD[16]: Principles and Considerations, FMV[17] Document ID: 33477/2006, Issue: 3.0.
Version	V1.0

The intended information interface between a provider and consumer is expressed in a service description. The service description must be explicit. This description is expressed in a formal language.

[16] NBD is Network Based Defense
[17] FMV is the Swedish Defense Material Acquisition group, or Forsvarets MaterielVerk..

Currently, there is no agreed formal syntax to express the service description's semantic parts, for example: achievement of a service. Therefore the service description parts are informal, i.e. natural language text.

The most important part of the intended coupling between a consumer and a provider of a service is the description of the information exchanged. The consumer requests a service and the provider produces the relevant information and distributes it to the consumer. On a concrete level, this is described by a set of interfaces and a protocol that describes the sequence of interactions. It is crucial that all information exchanged is covered by the service description.

In an Event Driven Architecture there may be no direct Request/response or 'publish/subscribe' relationship between a service consumer and a service provider. In this situation an external 'event' may act as the trigger for a consumer or provider service to initiate an action within the system.

Programmers need service descriptions, when implementing service providers and consumers. Tools for program development may help a programmer to automatically generate some parts of the service description so that both consumers and providers may use the service description. A template for a standard service description is needed.

Figure 2 illustrates service description in a SysML block definition diagram. The formal definition of service attributes and operations comprise the Service Description. Services support the Business Goals of an enterprise. See Annex C for additional discussion of how services support business goals.

The principle is stated as:

> *The service description should, as far as possible, be explicitly expressed in a formal language, the service description language.*

All information exchange between service providers and consumers should be through protocols and formally defined interfaces in the service description. The interfaces should emphasize achievement of the service and hide the implementation as much as possible. The service descriptions can be stored anywhere and communicated by whatever means available, including mail, telephone, and written notes. Consumers extracting useful information when connected to services not explicitly programmed for use by their application is a paradigm that is not mature in today's world. It is, however, the promise of the future. Also the structures and standards described in this framework will increase the maturity of services. The information exchange part of the service description is, therefore, primarily used when programs are designed and implemented, rather than, at run time. This is an area of extensive work and the risk of implementing "late binding" at run time will change over the decade to come. Service Certification may provide risk reduction in this regard.

From an interoperability point of view there must be an unambiguous definition/description of the service, independent of how the service is defined within the context of a specific SOA. Provision for service reference models, either based on a service meta-model or services described using a standardized service description, should be provided for service interoperability assessments.

Different implementation techniques for information exchange between computer systems have influenced the current formalisms used for service descriptions. As an example, two well-known formalisms currently used are different versions of Web Service WSDL[18] and CORBA IDL[19]. Another formalism in which it is possible to express service descriptions is UML[20]. In tactical systems, there are other paradigms in use such as Data Distribution Service (DDS).

The primary language today used for formal web service descriptions is the Web Services Description Language (WSDL). It is a standard specification for describing networked, XML-based services. It provides a way for service providers to describe the basic format of requests to their systems, regardless of underlying run-time implementation.

WSDL defines an XML format for describing network services as a set of endpoints operating on messages containing either data-oriented or procedure-oriented information. Operations and messages are first described abstractly and then bound to a concrete network protocol & message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages, regardless of which message formats or network protocols are used. This means interfaces are defined abstractly with XML schema, then bound to concrete representations appropriate for the protocol. WSDL lets a service provider specify the following:

- Name of the web service and addressing information
- Protocol and encoding style used when accessing the public operations of the web service
- Information such as operations, parameters, and data types comprising the web service interface

The service description contains:

a. Achievement of a service
    i. outcome of the service, effect
    ii. information and data required for performing a service
    iii. state of change or shared state
b. Allowed service protocols (process) to be used for information exchange
c. Interfaces that are used to exchange information between a service consumer and a service provider including:
    i. stateless or stateful interactions
    ii. transactions

[18] Web Service Description Language
[19] Interface Definition Language
[20] Unified Modeling Language

d. Definition of the data used in the interfaces and therefore in the information exchange model [21]
e. Properties that consumers can use to distinguish between different implementations of a service
f. Service functionality
g. Reachability (Visibility)
h. Basic information concerning the service contract and policy
i. A model of the service behavior

It is possible to extend service descriptions into new versions in such a way that service consumers and service providers implemented according to the old version can still function.

Context / Pre-conditions	Service Orientation, Service Oriented Architecture, Net-centricity
Results / Post-conditions	• Service descriptions explicitly expressed in a formal language. • Information exchange between service providers and consumers only through protocols and interfaces defined by service description. • Extendable service descriptions with maintained backward compatibility.

3.1.2 Service Interface

Identifier	Nif.Ncsf.principle.service-orientation.service-interface
Source	Adapted from http://msdn2.microsoft.com/en-us/library/ms998421.aspx
Version	V1.0

The Service Interface provides an entry point that consumers use to access functionality exposed by an application. The Service Interface is network addressable, so that it can be accessed by the consumer over a communication network. The network address can be a well-known location, obtained from a service directory such as UDDI[22] or an Enterprise Service Bus (ESB[23]). A key aspect of the design of a service interface is to decouple the implementation needed to communicate with other systems from the application's business logic. The service interface provides a more coarse-grained interface while preserving the semantics and finer granularity of the application logic. It also provides a barrier that enables the application logic to change without affecting the consumers of the interface.

The service interface implements the contract between the consumer and provider. Figure 2 illustrates the service interface as a published set of service operations, along with the contract relationship between the service interface and its provider/consumer. The service interface offers the capabilities of the service. The service contract represents an agreement between the service provider and the service consumer. This contract allows an exchange of information even if they are members of different systems. The service

[21] OASIS Reference Model
[22] Universal Description, Discovery and Integration
[23] An **Enterprise Service Bus** (ESB) refers to a software architecture construct -- typically implemented by technologies found in a category of middleware infrastructure products, usually based on recognized standards, which provide fundamental services for more complex architectures via an event-driven and standards-based messaging engine (the bus).

interface is responsible for implementation details that perform this exchange. Details include but are not limited to:

- **Network protocol**: The service interface encapsulates aspects of the network protocol used for communication between consumer and service. For example, suppose a service is exposed to consumers through HTTP over a TCP/IP network. It implements the service interface as a component published to a URL. The component receives the HTTP request, extracts information needed by the service to process the request, invokes the service implementation, packages the service response, and sends the response back to the consumer as an HTTP response. From the service perspective, the only component that understands HTTP is the service interface. The service implementation has its own contract with the service interface and has no dependency on the specifics of the technology used to communicate with the service interface.
- **Data formats**: The service interface translates between consumer data formats and the data formats that the service expects, either directly or via a mediation proxy. The translation of data formats is provided by a service or mediation proxy that exists in the SOA Infrastructure, potentially an ESB. For example, consumers external to the enterprise may supply data and expect reply data to be in an XML format conforming to an industry-standard XML schema. Internal consumers may use an XML format optimized for this particular service. The service interface is responsible for transforming and mapping both data formats into a format that the service can use. The service implementation does not have any knowledge of the specific data formats the service interface might use to communicate with the consumers.
- **Security**: The service interface should act as a trust boundary based on system policies. Different consumers may have different security requirements. The security (authentication) level provided by the service provider is part of the interface specification. The security level required by the service consumer is mapped to the offered security level. In case of conflicts, the security policy, security assertions and policy enforcement service components have to resolve the issue. Together with the security specifications, a role based access attribute is needed in the interface specification. For instance, consumers external to the enterprise will have more restrictive security requirements than consumers internal to the enterprise. External consumers may have strong authentication requirements and may only be authorized to perform a limited subset of the operations authorized for internal consumers. Internal consumers may be implicitly trusted for most operations, and only require authorization for the most sensitive operations.
- **Service Level Commitment:** Services interfaces describe the capability to fulfill specific requirements requested by a service provider. Some of these capabilities are specified as QoS, such as scalability, availability and fault-tolerance. How these QoS agreements are implemented is up to the service implementation or the capability of an ESB, if implemented. There might be service containers, which utilize caching to decrease response time and reduce bandwidth consumption. Multiple instances of the service may be deployed across a load-balanced set of processing nodes to achieve the QoS requirements.

  This illustrates that service level commitments are not strictly achievable by the service provider, but rather represent a collaborative interdependence between the service provider, the network transport provider, and the enterprise service manager. The latter may need to add network resources or instantiate multiple instances of service access points to meet service level commitments.

Use of a single, unchanging operational interface is one of the keys for service to access heterogeneous resources. This principle is vital to achieve Service Interoperability. To be interoperable, one should actively be engaged in the ongoing process of ensuring that the systems, procedures and culture of an organization are managed in such a way as to maximize opportunities for exchange and re-use of information, whether internally or externally."[24]

Context / Pre-conditions	Service Orientation, Service Oriented Architecture.
Results / Post-conditions	• Service interfaces are decoupled from the implementations • Application logic can change without affecting the consumers

3.1.3 Service Contract

Identifier	Nif.Ncsf principle.service-orientation.service-contract
Source	Adapted from http://www.serviceorientation.org/, http://docs.oasis-open.org/soa-rm/v1.0/
Version	V1.0

Service contracts include a definition of:

- The service endpoint
- Each service operation
- Every input and output message supported by each operation
- Rules and characteristics of the service and its operations
- Service Policies
- Quality of Service (QoS)
- Terms of use
- Price

Service contracts, often referred to as Service Level Agreements (SLAs), define almost all of the primary parts of an SOA. Useful service contracts may provide semantic information that explains how a service may go about accomplishing a particular task. This information establishes the agreement made by a service provider and its service requestors.

Because this contract is shared among services, its design is extremely important. Service requestors that agree to this contract can become dependent on its definition, so contracts must be maintained and versioned after every release. An SLA is published prior to service use. It sets the expectations between the consumer and provider. It is the cornerstone of how the service provider sets and maintains commitments to the service consumer.

[24] Paul Miller (2002), Interoperability Focus, UKOLN, University of Hull

A good SLA addresses the following five key aspects:

1. What is the provider promising?
2. How will the provider deliver on promises?
3. Who measures delivery, and how do they measure it?
4. What happens if the provider fails to deliver as promised?
5. How will the SLA change over time?

The following emerging standards are useful examples of how a SLA could be implemented in a SOA:

- Web Services Agreement Specification (WS-Agreement)
- eXtensible Access Control Markup Language (XACML)
- Web Services Policy (WS-Policy)

Web Services Agreement Specification (WS-Agreement), is a Web Services protocol for establishing agreement between two parties, such as between a service provider and consumer, using an extensible XML language for specifying the nature of the agreement, and agreement templates to facilitate discovery of compatible agreement parties. The specification consists of three parts which may be used in a composable manner: (1) a schema for specifying an agreement, (2) a schema for specifying an agreement template, and (3) a set of port types and operations for managing agreement life-cycle, including creation, expiration, and monitoring of agreement states.

XACML provides a policy language that allows administrators to define the access control requirements for their application resources. The language and schema support include data types, functions, and combining logic which allow complex (or simple) rules to be defined. XACML also includes an access decision language used to represent the runtime request for a resource. When a policy is located which protects a resource, functions compare attributes in the request against attributes contained in the policy rules ultimately yielding a permit, or deny decision. As an access control language, XACML aids in implementation independent SLA definition. For example, XACML can define authentication rules that are independent of the authentication infrastructure utilized.

The relationship between the Service Interface and the Consumer/Provider as shown in Figure 2 illustrates where the Service Contract principle is invoked.

Context / Pre-conditions	Service Orientation, Service Oriented Architecture.
Results / Post-conditions	The service should have a contract that defines the following properties: • the service endpoint • each service operation • every input and output message supported by each operation • rules and characteristics of the service and its operations

3.1.4 Service Abstraction

Identifier	Nif.Ncsf.principle.service-orientation.service-abstraction
Source	Adapted from http://www.serviceorientation.org/
Version	V1.0

Abstraction of underlying logic (also referred to as service interface level abstraction), is the principle that allows services to act as black boxes, hiding their details from the outside world. The scope of logic represented by a service significantly influences the design of its operations and position within a process.

There is no limit to the amount of logic a service can represent. A service may be designed to perform a simple task, or it may be positioned as a gateway to an entire automation solution. There is also no restriction as to the source of application logic a service can draw upon. For example, a single service can expose application logic from two or more systems. Service interface level abstraction is one of the inherent qualities provided by services. The level of abstraction is often referred to as Granularity. The coarser the grain, the less overhead in the SOA; the finer the grain, the greater the possibility for reuse. A loosely coupled structure requires that the only piece of knowledge services need to exchange prior to binding is the service description.

The "isAbstractedBy"[25] relationship between the Service and the Service Interface as shown in Figure 2 illustrates where the Service Abstraction principle is invoked.

Context / Pre-conditions	Service Orientation, Service Oriented Architecture.
Results / Post-conditions	The service should be able to act as a black box, hiding its details from the outside world. The only piece of knowledge needed to exchange prior to binding should be the service contract.

3.1.5 Loose Coupling

Identifier	Nif.Ncsf.principle.service-orientation.loose-coupling
Source	Adapted from http://www.serviceorientation.org/
Version	V1.0

[25] Note that formal tags for relationships and other entities within a SysML or UML diagram can be limited by the capabilities of the SysML tool. Often, spaces or hyphens are not allowed in such tags. In this case, many users revert to a technique known as "camel case" where each new "word" of the entity tag is capitalized, giving the reader clues about the string of words in the tag by the location of the capitals (camel humps). Breaking the tag apart with spaces in descriptive text might be easier on the eye, but can lead to disconnects in references to diagrams. It does, however, cause technical editors to cringe.

To respond to unforeseen changes in an efficient manner is a key goal of applying service-orientation. Realizing this agility is directly supported by establishing loosely coupled service relationships (i.e., "Service Independence").

Loose coupling is a condition wherein

> *"a service acquires knowledge of another service while still remaining independent of that service"*.

Loose coupling is achieved through the use of service contracts that allow services to interact within predefined parameters. Within a loosely coupled architecture, service contracts tightly couple operations to services. Once a service is formally described as being the location of an operation, other services will depend on that operation-to-service association. The "coupling_parameters"[26] attribute of the Service_Interface entity in Figure 2 illustrates where the Loose Coupling principle is invoked.

Context / Pre-conditions	Service Orientation, Service Oriented Architecture.
Results / Post-conditions	The service should acquire knowledge of other services while still remaining independent of those services. It should be achieved through use of service contracts that allow services to interact within predefined limits.

3.1.6 Service Interoperability

Identifier	Nif.Ncsf.principle.service-orientation.service-interoperability
Source	Interoperability. What is it and Why should I want it?, Paul Miller, 21-Jun-2000, Ariadne Issue 24, http://www.ariadne.ac.uk/issue24/interoperability/intro.html
Version	V1.0

Service Interoperability is the ability for services to access heterogeneous resources by means of a single, unchanging operational interface. "To be interoperable, one should actively be engaged in the ongoing process of ensuring that the systems, procedures and culture of an organization are managed in such a way as to maximize opportunities for exchange and re-use of information, whether internally or externally."[27]

Service Interoperability facilitates information sharing and provides ability to mix and match information system components through services without compromise. Service interoperability allows users to:

- Find information, services, and applications when needed, independent of physical location

[26] Note that the tool used to create Figure 2 allows the underscore character in labeling SysML attributes and methods, but is not used in labeling relationships. Presumably this was not done by the tool software developers to intentionally irritate technical editors. But it is a possibility.

[27] Interoperability. What is it and Why should I want it?, Paul Miller, 21-Jun-2000, Ariadne Issue 24, http://www.ariadne.ac.uk/issue24/interoperability/intro.html

- Understand and employ the discovered information and tools, no matter what platform supports them, whether local or remote
- Evolve a processing environment for commercial use without being constrained to a single vendor's offerings.

Service Interoperability between systems and system components through services has several aspects:

- **Network Protocol Interoperability** allows basic communications between components through services
- **Standard Interface Specifications** allow "client applications to execute procedures on remote systems" through services
- **Data Transport Interoperability** allows for sharing of data and services through transparent access, regardless of any proprietary data storage formats, data transmission formats or dynamic capacity variability
- **Semantic Interoperability** – refers to applications interpreting data consistently in the same manner in order to provide the intended representation of the data.

Note that the first three of these aspects deal with the syntax required to enable interoperability. Semantic interoperability is also required but is the task of the application. The interface cannot ensure that applications are correctly written.

From an interoperability point of view there must be an unambiguous definition/description of the service independently of how the service is defined within the context of a specific SOA. Provision for service reference models either based on a service meta-model or services described using a standardized service description should be provided for service interoperability assessments. The definition/description should include, but not be limited to, the following service properties:

- Service Achievements
  - outcome of the service, effect
  - information and data
  - state of change or shared state
- Service Interface and Protocols
  - stateless or stateful interactions
  - transactions
  - visibility
- Service Contracts and Policies
  - Service Level Agreement
  - Quality of Service (QoS)
  - terms of use

The details of the Service_Interface element in Figure 2 illustrate the SysML properties of the Service Interoperability principle.

Context / Pre-conditions	Service Orientation, Service Oriented Architecture.
Results / Post-conditions	

3.1.7 Service Composition

Identifier	Nif.Ncsf.principle.service-orientation.service-composition
Source	Adapted from http://www.serviceorientation.org/
Version	V1.0

A primary benefit of SOA is the ability to compose applications, processes, or more complex services from less complex services. This activity, often called service composition, allows developers to compose applications and processes using services from heterogeneous environments without regard to details and differences of those environments. While data abstraction, as described in Section 3.1.4, provides for greater reuse with finer-grained interfaces, business service composition is more achievable if the services are constructed with coarse-grained interfaces. With a palette of effectively designed and composed coarse-grained services, a business expert can productively compose new business processes and applications. Orchestration and Choreography each has a distinct purpose but can perform together to achieve service orientation.

Orchestration and Choreography are both critical pieces of service composition and collaboration. These techniques are being developed to effect service "work flow", or the combination and execution of services in an optimum manner. It is important to describe these two terms in a way that is most useful to industry, with the understanding that implementers may tailor these terms to their own needs. At a recent Open Group Conference the following comparison was provided to the musical world: Orchestration is analogous to having the correct composition of instruments in a Symphony, and Choreography is analogous to bringing those instruments on line in specific order. For example, the second violins may begin after the first violins and play differing but complementary themes than the first violins.

Standards bodies that address composition include: OMG, OASIS, and the World Wide Web Consortium (W3C). Each of these organizations takes a somewhat different approach to addressing Orchestration and Choreography in both detail and level. A summary of their definition of these terms is provided in the table below:

Term	OMG	OASIS	W3C
Orchestration	*"the modeling of directed, internal business processes"*	*A technique used to compose hierarchical and self-contained service-oriented business processes that are executed and coordinated by a single agent acting in a "conductor" role.*	*An orchestration defines the sequence and conditions in which one Web service invokes other Web services in order to realize some useful function. I.e., an orchestration is the pattern of interactions that a Web service agent must follow in order to achieve its goal.*
Choreography	*"the specification of interactions*	*A technique used to characterize and to compose service-oriented business collaborations based on ordered message*	1. *A choreography defines the sequence and conditions under which multiple cooperating independent agents exchange messages in*

	between autonomous processes".	*exchanges between peer entities in order to achieve a common business goal. Choreography differs from orchestration primarily in that each party in business collaboration describes its part in the service interaction in terms of public message exchanges that occur between the multiple parties as standard atomic or composite services, rather than as specific service-oriented business processes that a single conductor/coordinator (e.g., orchestration engine) executes.*	*order to perform a task to achieve a goal state.* *2. Web Services Choreography concerns the interactions of services with their users. Any user of a Web service, automated or otherwise, is a client of that service. These users may, in turn, be other Web Services, applications or human beings. Transactions among Web Services and their clients must clearly be well defined at the time of their execution, and may consist of multiple separate interactions whose composition constitutes a complete transaction. This composition, its message protocols, interfaces, sequencing, and associated logic, is considered to be a choreography. [WSC Reqs]*

From the table above, one can see that some of the definitions are complementary while others are not so closely aligned. The issues include (1) centralized management of service application versus a peer to peer (or autonomous) coupling, (2) approaching service management through a "business process" versus a thrust to make the services re-usable for other, sometimes future applications, and (3) questioning whether orchestration and choreography are aligned or contradictory (top down versus peer to peer viewpoint). Factors that should be addressed to achieve orchestration and choreography follow in Sections 3.1.8 and 3.1.9, respectively.

In addition to Orchestration and Choreography, composition includes overlapping principles such as governance and security, and service contract aspects of composition. It is important to address these aspects of composition, even though they may add to workflow overhead.

A service can represent any range of logic from various sources, including other services. The primary reason to implement this principle is so services can participate as effective members of other service compositions. This requirement is irrespective of whether the service itself acts as composer of others. The principle of making any service composable places an emphasis on service operations. Composability is simply another form of reuse and therefore operations need to be designed in a standardized manner and with an appropriate level of granularity in order to maximize composability opportunities.

The attributes and operations of the Service entity in Figure 2 provide the details of the components for Service Composition.

Context / Pre-conditions	Service Orientation, Service Oriented Architecture.
Results / Post-conditions	The service may participate as an member of other service compositions irrespective of whether the service itself acts as service composer.

3.1.8 Service Orchestration

Identifier	Nif.Ncsf.principle.service-orientation.service-orchestration
Source	Adapted from http://www.serviceorientation.org/
Version	V1.0

Key attributes for Orchestration include (1) participant and role definition, (2) variables, (3) properties enabling conversation including security (access control), (4) fault handlers for exception processing, (5) compensation handlers for error recovery and (6) event handlers responding to concurrent events with the process itself and set of activities. Web Services Business Process Execution Language (WS-BPEL) is an XML based programming language that describes high level business processes. A "business process" is a term describing interaction between two businesses or two elements in one business. An example of this might be company A purchasing an item from company B. BPEL allows this interaction to be described easily and thoroughly, such that company B can provide a Web Service and company A can use it with minimum compatibility issues. WS-BPEL may be executed on an orchestration engine and is an example of a relevant Web Services Standard. An orchestration engine is a server that is responsible for acting as an intelligent intermediary between services.
The Service_Orchestration entity in Figure 2 illustrates where the Service Orchestration principle is invoked.

Context / Pre-conditions	Service Orientation, Service Oriented Architecture
Results / Post-conditions	Processes can use services based on: • participant and role definition • properties which enable conversation • fault handlers for exception processing • compensation handlers for error recovery • respond to event driven activities

3.1.9 Service Choreography

Identifier	Nif.Ncsf.principle.service-orientation.service-choreography
Source	Adapted from http://www.serviceorientation.org
Version	V1.0

A SOA is an information system architecture for linking resources on demand. SOA may scale from low level tactical functionality to that of an enterprise. In a SOA, resources are made available to participants in a network, enterprise and line of business (typically spanning multiple applications within an enterprise or across multiple enterprises). It consists of a set of business-aligned IT services that collectively fulfill an organization's business processes and goals. One can choreograph these services into composite applications and invoke them through standard protocols. Figure 2 shows how Service Objects can be combined by Choreography, enabling dynamic reconfiguration which, in turn, decouples Providers and Consumers of the System's Resources.

For Choreography, message structure, asynchronous communication, message rules, invocation, events and event handling are important design factors. An example is Web Services Choreography Description Language (WS-CDL), a language for describing multi-party contracts. It is somewhat like an extension of WSDL: WSDL describes web services interfaces; WS-CDL describes collaborations between web services and serves as an example of a relevant Web Services Standard. Business Process Modeling notation (BPMN) is another example of a standard that may be used to describe choreography. Choreography can be performed at design time or run time. As mentioned earlier in this paper, run time choreography is not a widely used or mature technology; however in the decade ahead, run time Choreography will be a significant factor in achieving the promise of service utilization.

Context / Pre-conditions	Service Orientation, Service Oriented Architecture.
Results / Post-conditions	Services can be combined into composite services and be invoked through standard protocols,

3.1.10 Service Discovery

Identifier	Nif.Ncsf principle. service-orientation.service-discovery
Source	Adapted from http://www.serviceorientation.org/
Version	V1.0

Discoverability on the level of architecture is meant to refer to the technology architecture’s ability to provide a mechanism of discovery, for example a service directory or registry. Such extensions effectively become part of the overall infrastructure that is meant to support the implementation of a Service Oriented Architecture. On the level of service, the discoverability principle can be referred to the design of an individual service so that it becomes as discoverable as possible – no matter whether the discoverability extension or product actually exists in the surrounding implementation environment.

The reason behind this is that even if there is no need for a service registry, owing to the fact that there is not enough of a service inventory to warrant the need for one, services should consistently be designed as resources that are highly discoverable in some fashion. By doing so, the evolutionary governance of those services can be better managed when the service portfolio increases in size, as each service will then be equipped with the metadata that is required to properly communicate its capabilities and meaning.

Discovery helps avoid the accidental creation of redundant services or services that implement redundant logic. Because each operation provides a potentially reusable piece of processing logic, metadata attached to a service needs to sufficiently describe not only the service’s overall purpose, but also the functionality offered by its operations. The desire to use the discoverability principle may depend on the mode of operation; for example, a service might need to be discoverable in the maintenance mode but not in an operational mode. UDDI is one example of a standard that can be used to implement discovery for an SOA.

The “discovers” relationship between the System and Resource in Figure 2 is inherited by the Service (a sub-class of Resource). This relationship illustrates the Service Discovery principle.

Context / Pre-conditions	Service Orientation, Service Oriented Architecture.
Results / Post-conditions	The service should be discoverable.

3.1.11 SOA Governance

Identifier	Nif.Ncsf.principle.service-orientation.governance
Source	http://www-306.ibm.com/software/solutions/soa/gov/
Version	V1.0

SOA governance is an extension of IT governance that focuses on the service lifecycle and composite applications in an organization’s service-oriented architecture (SOA). The SOA governance defines:

- Decision rights for the development, deployment and management of new services.
- Monitoring and reporting processes for capturing and communicating governance results.

Because SOA applications are intrinsically federated, they introduce governance challenges. With proper policies, principles, standards, procedures and processes in place, businesses may realize the full benefit of service orientation. An effective SOA governance platform not only helps business and IT teams better identify which projects contribute most to business goals, but it also empowers employees to work and collaborate more efficiently by clearly defining their roles and responsibilities.

Governance provides service lifecycle management. Once the SOA governance framework is implemented, it is used to assemble, deploy and manage phases within the service lifecycle. Service lifecycle management focuses on development and deployment of services. SOA governance supplies the decision rights, processes and policies for those activities. Once a service is deployed, there must be management aspects in place to control and monitor the service. Within service lifecycle management there is a set of functions that will need to be in place and governed to ensure that the value proposition of SOA, particularly reuse and cost reduction, is achieved.

The "isGovernedBy" relationship between the Service and System_Management entities in Figure 2 illustrates where the SOA Governance principle is invoked.

Context / Pre-conditions	Service Orientation, Service Oriented Architecture
Results / Post-conditions	Decision rights for the development, deployment and management of new services are in place. Monitoring and reporting processes for capturing and communicating governance results are in place.

3.1.12 Location Transparency

Identifier	Nif.Ncsf.principle.service-orientation.location-transparency
Source	SOA for NBD: Principles and Considerations, FMV Document ID: 33477/2006, Issue: 3.0.
Version	V1.0

One aspect of the loose coupling between service implementations is that the reference (address) of a service provider should never be hard-coded in the consumer. A prerequisite to obtain this is to:

> *Manage references (addresses) to service providers outside the implementation of the service consumer.*[28]

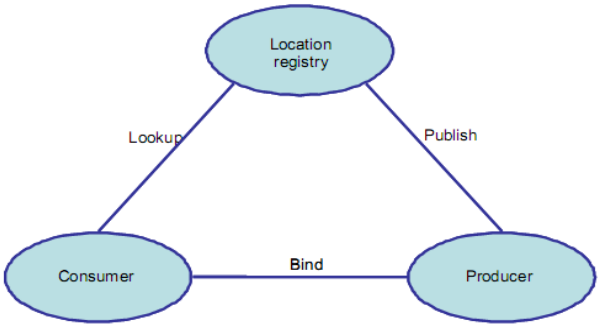


Figure 3 – Relationships: Source: NCOIC

The most common way to do this is to use a location registry that acts as a repository for addresses and characteristics of available service implementations. Figure 3 illustrates the relationship between consumer, provider and the registry of available service implementations. The provider publishes its services to the registry. The consumer looks up and finds the appropriate service and binds (the service) to the provider in order to use the service provided by the provider.

The location registry contains enough information in order for a consumer to choose a relevant service provider. The service type, the service provider identity, and other properties, relevant to the particular service type, must be available. These are attributes of the "Service" entity as shown in Figure 2. One property that is relevant for some services is the geographical position; another property that is relevant for sensors is the coverage area. All this information should be possible to use by the consumer when it searches the registry for a suitable service provider. Another way to accomplish location transparency is to use an Enterprise Service Bus (ESB) design.

Context / Pre-conditions	Service Orientation, Service Oriented Architecture, Net-centricity, Swedish NBD Manage references (addresses) to service providers outside the implementation of the service consumers
Results / Post-conditions	Location transparent and loosely coupled services

[28] Swedish FMV, SOA for NBD: Principles and Considerations, FMV Document ID: 33477/2006, Issue: 3.0

3.1.13 Security

Identifier	Nif.Ncsf.principle.service-orientation.security
Source	SOA for NBD: Principles and Considerations, FMV Document ID: 33477/2006, Issue: 3.0.
Version	V1.0

To implement SOA in a distributed environment there are unique security principles that must be addressed. Primary motivations for SOA are flexibility and dynamics, a security challenge. Flexibility implies that the security principles must be universally defined and applied throughout the system. These principles, are not unique for SOA, but are stated here because they are important for the overall goal. Security must be universal over the entire system because flexibility and dynamic configurations requires that there is no predefined configuration dictating what systems should be coupled to other systems.

Service consumers and service providers must be able to identify themselves using a common method for authentication of users and services. There must be a common method and implementation of authorization for using services and data. Service providers must check that consumers are authorized to use the service and obtain the required data before a service request is granted. Information about availability of data and services which users are allowed to mutually use must be possible to change dynamically. These changes are themselves subject to authorization.

There must be a universal method to establish integrity. A service consumer must be able to check that the data sent from the intended source party is not changed by a third party.

There must also be a universal method to guarantee the confidentiality of the information exchanged, the most common of which is encryption. Non-repudiation, achieved through several means as indicated in the Information Assurance Design Framework is a key aspect for the user. Without non-repudiation, users become inclined not to use a given fielded system. This is complicated by identity theft or, on the other hand, signature forgery. Defense in Depth provides a layered defense to the Attacker. With provisions such as Prevention, Protection, Containment and Recovery, such a layered defense is established. Finally, in a federated environment where National Statutes may prevail over operational considerations, it is vital to recognize an overriding need for multiple security levels.

The above description is not exhaustive. Security principles make a tighter coupling between services, since the service provider and service consumer must implement the same methods for authentication, authorization, integrity and confidentiality to exchange information. The Security principle for Services is invoked by the Business_Goals and Policy entities as shown in Figure 2. For a more complete description, the reader is referred to the NCOIC Information Assurance Design Framework (IADF).

Context / Pre-conditions	Service Orientation, Service Oriented Architecture, Net-centricity, Swedish NBD
Results / Post-conditions	• Service consumers and providers capable of authentication through a common authentication method

	• A common method and implementation of authorization for the use of services and their data • Service providers that check that consumers authorized to use the service and requested data before the service request is granted • The ability to dynamically redefine the service access policy in runtime with these changes being subject to authorization themselves • A universal method for service consumers to check the integrity of data received from a service provider • A universal method guaranteeing confidentiality of data exchanged between service consumers and providers • A layered defense against attack, providing for recovery is warranted

3.2 Net Centric Service Principles

Net-centricity[29] focuses on enabling operations among systems across domain, context, and organizational boundaries. It also focuses on the potential for doing this dynamically and at low cost. Service orientation describes an architecture that asymmetrically provides services to arbitrary service consumers, and thus supports to some degree the open-ended aspects of Net-centricity. Service orientation is enabled by network technology, but to date has focused primarily on connecting systems regardless of the platform execution environment. Net-centricity by itself has not yet addressed the issue of establishing the means to share services across organizational or domain or context boundaries except in isolated and constrained situations (e.g., partnering with eBay or Amazon in which the major partner dictates the service framework). Service orientation by itself does not drive systems to be dynamic or agile at run time, but services can be built to support such dynamic service selection and adaptivity.

There are key characteristics that services and SOAs require to become more Net-centric. Net-centricity business model constraints limit what is achievable and operationally effective in a given institutional context (e.g., DoD, Department of Homeland Security, the Intelligence Community, North Atlantic Treaty Organization (NATO) Response Force, and The Global War on Terror). The following set of characteristics is derived from observations on key conceptual elements of Net-centricity and service orientation in preceding sections. The characteristics also draw on the SCOPE model work underway in the NCOIC. The Net Centric Service Principles identified to date are:

- Execution Platform Independence
- Explicit Service Business Model
- Environment Monitoring, Assessment, & Adaptation
- Reuse
- Dynamic Configuration
- Explicit Service Scoping
- Autonomicity
- Robustness (Graceful Degradation)

[29] The Essence of Net-centricity, AFEI DS3WG, October 12, 2006, Hans Polzer, Lockheed Martin Corp.

- Net-Centric Data

3.2.1 Execution Platform Independence

Identifier	Nif.Ncsf.principle.net-centric-service.execution-platform-independence
Source	The Essence of Net-centricity, AFEI DS3WG, October 12, 2006, Hans Polzer, Lockheed Martin Corp.
Version	V1.0

It shouldn't matter to either the provider or consumer what the system architecture of the other party is. Most industry SOA frameworks based on Web Services standards have this characteristic. Being explicit pays dividends, however. Execution platform independence facilitates crossing system and enterprise boundaries. This is because the likelihood that two systems or enterprises employ identical execution platform environments or versions of those environment (including virtual environments such as J2EE and Microsoft .Net) for all service instances gets smaller as the number and breadth of services and systems/institutions involved gets larger. All services should be accessible over the network via protocols that are architecture independent. All of the above are embodied in the NCOIC Interoperability Framework model. Platform Independence is often referred to in industry as "virtual machines".

The intent of SysML diagrams such as Figure 2 (or UML) is to provide a platform independent model for Services.

Context / Pre-conditions	Service Orientation, Service Oriented Architecture, Net-centricity
Results / Post-conditions	All services accessible over the network via protocols that are system architecture independent

3.2.2 Dynamic Configuration

Identifier	Nif.Ncsf.principle.net-centric-service.dynamic-configuration
Source	SOA for NBD: Principles and Considerations, FMV Document ID: 33477/2006, Issue: 3.0.
Version	V1.0

Available services in a Situation Adapted System should be dynamically configurable at run time. This means the service consumer must have the ability to be choreographed in to match a business demand, defined in the business process. By using the service concept a defined business process can more easily be implemented and changed when new business demands occurs. The principle is:

"Service implementations must have the ability to be dynamically configured at run time."

Figure 2 Illustrates that the Dynamic Reconfiguration Principle is a key motivator for decoupling service Providers and service Consumers. Without decoupled service Providers and Consumers, dynamic reconfiguration would also require dynamic remapping of Provider/Consumer relationships, identified in the coupling_parameters element in the Service_Interface entity.

Context / Pre-conditions	Service Orientation, Service Oriented Architecture, Net-centricity, Swedish NBD
Results / Post-conditions	Service implementations with the ability to be dynamically configured at run time

3.2.3 Explicit Service Business Model

Identifier	Nif.Ncsf.principle.net-centric-service.explicit-service-business-model
Source	The Essence of Net-centricity, AFEI DS3WG, October 12, 2006, Hans Polzer, Lockheed Martin Corp.
Version	V1.0

The business/institutional relationship between most service providers and service consumer/users should be explicitly defined in the service description accessible over the network, and the service interface definition should include parameters necessary to execute the business model.[30] If the service is free to all consumers for any purpose and frequency whatsoever, the service description should so state. If there is a usage cost, licensing fee, or other usage restriction, the service interface should include parameters necessary to verify and implement the specified business model. Security domains, digital rights management, or similar multi-lateral agreements are examples of service usage restrictions that may not be strictly monetary in nature. A more net-centric SOA makes more aspects of the service business model explicit, and provided for in the service interface and supporting systems management services.

As shown in Figure 2, Services support the Business Goals of the enterprise. An enterprise's Business_Goals create Policy which, when enforced by System_Management, instantiate the Explicit Service Business Model.

Context / Pre-conditions	Service Orientation, Service Oriented Architecture, Net-centricity
Results / Post-conditions	Accessible service descriptions with explicitly defined business relationship between the service provider and the service consumer and service interface definitions including parameters necessary to execute the business model

[30] However, a service provider and consumer may have an indirect relationship that is part of a business process. For example, the discovery service has no explicit business relationship with its clients.

3.2.4 Explicit Service Scoping

Identifier	Nif.Ncsf.principle.net-centric-service.explicit-service-scoping
Source	The Essence of Net-centricity, AFEI DS3WG, October 12, 2006, Hans Polzer, Lockheed Martin Corp.
Version	V1.0

The service description should include explicit definitions of the applicable scope of the service in terms of institutional domain, mission capability space, and operational context scope. Additional service scope dimensions should be specifiable (e.g., geospatial area of coverage for a particular service instance). Service scope specifications should be discoverable and readily handled by potential service requestors over the network. Service naming conventions may be used to facilitate explicit service scoping, but the use of explicit scoping parameters in the service interface definition is a preferred approach. Both have business model and service provisioning[31] implications. There could be a tradeoff between complexity of the service description and use of Service Management to ascertain details contained in the service definition.

The “supports” relationship between the Service and Business_Goals elements in Figure 2 illustrates Service Scoping. An example of Explicit Service Scoping is shown in Figure C-1 in Annex C. There are three example explicit scope relationships between the Service and the Business_Goals.

Context / Pre-conditions	Service Orientation, Service Oriented Architecture, Net-centricity
Results / Post-conditions	Accessible service descriptions with the applicable scope of the service explicitly defined and the possibility to specify additional service scope dimensions if appropriate

3.2.5 Environment Monitoring, Assessment, and Adaptation

Identifier	Nif.Ncsf.principle.net-centric-service.environment-monitoring-assessment-adaptation
Source	The Essence of Net-centricity, AFEI DS3WG, October 12, 2006, Hans Polzer, Lockheed Martin Corp.
Version	V1.0

[31] Service provisioning refers to the "preparation beforehand" of IT systems' materials or supplies required to carry out a specific activity. The Service Provisioning Markup language is the open standard for the integration and interoperation of service provisioning requests.

Service consumers may wish to devote more resources to monitoring the availability of services, the service versions on the network, and the current service performance, rather than assuming this is done at design or system configuration time. They may need to be prepared to select new service instances or versions, or even new services that may become available that are more appropriate to their own mission/enterprise objectives. Service consumers may need to be more aware of network performance and latency, and not only convey any issues to their immediate customers/users prior to run time but also to adapt for these conditions at run time. They may also be less trusting of service providers and sensitive to any certification parameters or trust mechanisms offered in the SOA. Conversely, service providers may be explicit about service certification state and versioning information, exposing version information in the service interface itself. Service management services may support this type of discovery, assessment, and adaptation interaction over the network.

Service interfaces in a net-centric system must be robust enough to allow consumers need to adapt to their environment so as to accommodate network bandwidth constraints. Service interfaces in a net-centric system should allow consumers to adapt to a provider's power, consumables and environmental considerations.

The Service_Interface entity in Figure 2 embodies the attributes and operations required to implement this principle.

Context / Pre-conditions	Service Orientation, Service Oriented Architecture, Net-centricity
Results / Post-conditions	Cautious service requestors that monitor the availability of services, the service versions, and the current service performance in runtime. Also the ability to select new service instances, versions or even new services at runtime.

3.2.6 Autonomicity[32]

Identifier	Nif.Ncsf.principle.net-centric-service.autonomicity
Source	The Essence of Net-centricity, AFEI DS3WG, October 12, 2006, Hans Polzer, Lockheed Martin Corp.
Version	V1.0

Autonomicity describes the Net-centric principle of self governing systems. Information about services and SOAs, such as design, configuration, network and geospatial distribution, security domain, certification state, composability, current performance, and functional/organizational scope dimensions, need to be accessible over the network via, and for, service management services. Historically, this kind of information was compiled in design documents and configuration management systems not accessible

[32] Autononomicity refers to self management differing from Autonomy which refers to self governance.

to run time software and services. The net-centric model, with its emphasis on crossing system boundaries and dynamic adaptivity, suggests that this kind of information needs to be available to the service instances themselves so that these instances can appropriately monitor and adapt to changing network and operational mission and context conditions. For example, a net-centric system should allow resource minimization through deference of state information dynamically based upon policy. Basically, services must be able to adapt to their underlying run time execution environment and offer levels of service to applications that allow for dynamic operation in a disadvantaged communications environment.

System_Management enforcement of Policy in order to govern Service use is illustrated in Figure 2. Enforcement can be accomplished in many ways. To accomplish Autonomicity, specific enforcement of self-governing Policy is required. This will be one topic of the planned System Management Specialized Framework (Figure 1).

Context / Pre-conditions	Service Orientation, Service Oriented Architecture, Net-centricity
Results / Post-conditions	Information about a SOA and its services available to the service instances themselves so that these instances can appropriately monitor and adapt to changing network and operational mission and context conditions

3.2.7 Reuse

Identifier	Nif.Ncsf.principle.net-centric-service.reuse-of-infrastructure-components-code
Source	SOA for NBD: Principles and Considerations, FMV Document ID: 33477/2006, Issue: 3.0.
Version	V1.0

Reuse has elements that should be considered within Service Principles as well as more specialized elements that address Net-Centric Principles. In this document, the discussion of Reuse is contained in the Net-Centric Principles in this subparagraph. A planned NCOIC pattern will address this in more detail.

Service Reuse   Reusability[33] is a generalized principle which relies on some of the other principles to reuse a service. Service-orientation encourages reuse in all services, present and future, regardless of whether immediate requirements for reuse exist:

- Design Standards - Applying design standards make each service potentially reusable
  - Eases the implementation of application changes to accommodate new requirements
  - Reduces development effort
- Wrapper Services
  - Inherently reusable services reduce need for creating wrapper services
  - Eliminates a generic interface over less reusable services that inhibits performance.

[33] Reuse in the SOA context differs from reuse in a software engineering context where one can reuse a particular software component or software configuration item at design time vice having that software object visible and available over a network.

Service orientation facilitates inter-application interoperability, composition, and the creation of cross-cutting or utility services. Because a service is simply a collection of related operations, the logic encapsulated by the individual operations must be deemed reusable to warrant representation as a reusable service.

Separation of Infrastructure & Higher Level Components: The purpose for reusing infrastructure components and code is cost reduction. To make reuse possible, the information exchange description should be written on a business logic abstraction level. This provides important advantages: The lower level components of the information exchange will be similar or identical in different providers and consumers, thereby making component use possible. Another reason, perhaps of equal importance, should be mentioned. If an implementation of the information exchange is reused, it is up–to-date, and its quality will be improved. A third reason is that there will be a clear interface in the program between the business logic code and the information exchange code (reused code). This interface is essential as a boundary between the business dependent part and the infrastructure part of the implementation, thus providing the possibility of different lifecycles of the business logic code and the code for information exchange.

There should be an explicit interface between business logic code and information exchange code to allow separated lifecycles. Implementations of the information exchange should, for cost effective reasons, be reused as much as possible.

Context / Pre-conditions	Service Orientation, Service Oriented Architecture
Results / Post-conditions	• Information exchange descriptions on a business abstraction level. • Explicit interfaces between the implementation of business logic and information exchange in order to enable separated lifecycles. • Reused information exchange implementations maximize cost effectiveness.

3.2.8 Robustness (Graceful Degradation)

Identifier	Nif.Ncsf.principle.net-centric-service.robustness
Source	SOA for NBD: Principles and Considerations, FMV Document ID: 33477/2006, Issue: 3.0.
Version	V1.0

The concept of Autonomy is implemented within this net-centric principle. There are many sources of failure in distributed systems. A service consumer must have adequate fault handling capabilities to cope with a situation in which the services the consumer is using may fail. It may be that the provider itself can fail, or the communication path to the provider can fail or degrade, so as to be unusable. In both cases the consumer must cope with the situation.

From the perspective of the consumer, service providers must be implemented in such a way that failure of a service provider does not result in failure of the consumer itself. Conversely, service consumers must be implemented in such a way that a connected consumer failure does not result in a provider failure. [34]

From an enterprise standpoint, a robust system will have alternate providers of a given service available to the consumer population. The contents of a given service may even vary between providers. For example, one provider may have a different version of a named service than another provider. This anomaly must be able to be resolved through the Service Level Agreement.

Another benefit of SysML diagrams such as Figure 2 (or UML) is to provide a robust model for Services.

Context / Pre-conditions	Service Orientation, Service Oriented Architecture, Net-centricity, Swedish NBD
Results / Post-conditions	Services robust enough to cope with situations where network or other services fail

3.2.9 Net Centric Data

Identifier	Nif.Ncsf.principle. net-centric-service.net-centric-data
Source	Adapted from Swedish NBD Design Rule Legacy Integration, FMV Document ID: 32912/2006, Issue: 1.0.
Version	V1.0

The purpose of making data net centric is to expose data to the network and enable available data from systems on the network. This includes existing data produced by the systems (the core functionality) e.g. sensor and track data. It may also include new types of data from the systems, e.g. data from other connected equipment on a platform.

- The data and functionality exposed on the network may be offered as a service consumable by other services. Figure 2 illustrates the "offers" relationship between the Service entity and the Data entity
- The services and data from the network are available to the designers and operators of the net-centric system. The level of integration may vary, from support functions completely isolated from current (legacy) applications (e.g. chat and email services), to functions tightly integrated with the existing legacy applications
- The data should be described with meta data to facilitate processing.

[34] The concept of Autonomy is implemented within this net-centric principle.

- Metadata for data made visible, discoverable, and accessible, should itself be visible, discoverable, and accessible over the network. This includes information models and schemas for metadata, whether included as embedded tags with the data or stored in a separate metadata repository. Metadata should include any frames of reference used for key data element values, especially identity or type attributes. If there is a reference value standard or naming/assignment service associated with a metadata element, it, too, should be a network accessible service. For example, in the case of UPC codes, the service for assigning or looking up or validating UPC codes should be a network-accessible service. Unit names or ship names, Radio Call Signs, or hull numbers might be a similar example. XML schemas should be visible, discoverable, and accessible over the network.

Context / Pre-conditions	Service Orientation, Service Oriented Architecture, Net-centricity
Results / Post-conditions	Data is provided in a Net-centric manner

4 Pattern Checklist & Topics

The baseline Pattern Model is described in the NSD-RM Overarching Framework, section A.1.7.1.3. Specialized Frameworks may extend this definition, but also inherit the requirements from the Overarching framework in the NSD-RM. Patterns are descriptions of mature solutions that solve similar problems and are constrained by a specific context.[35]

Adoption of service-oriented principles makes it easier to obtain interoperability and support Net-centric behavior. Service orientation alone doesn't guarantee interoperability. *It takes more than providing a bucket full of services to make truly interoperable systems with intrinsic net-centric capabilities.* Even services realized within a standardized SOA framework like Web Services (WS) may be non interoperable by differences in implementation, discoverability and transport bindings. Patterns serve as bridging vehicles between abstract service identification and specification in the design phase and final service realization. These patterns should be interoperable net-centric solutions within the scope of service orientation.

Systems using the same patterns and implementations should be interoperable, at least within the pattern scope. Pattern matching and selection is a pragmatic and efficient approach for interoperability and Net-centricity without dictating what and how to implement services. It does not guarantee interoperability outside the pattern scope; more than one pattern may be selected to cover a broader scope. Multiple patterns provide flexibility, scalability and modularity needed for the complete lifecycle since patterns can be added, dropped or exchanged over time. The construct of multiple patterns adds to complexity of the solution. A multiple pattern construct has an impact on net-centricity and interoperability of the end result, in this case the amalgamation of several net-centric service oriented patterns. So it is important to stress that pattern matching only provides net-centricity and interoperability assurance on an architectural level.

Consideration of more than a single service is important for interoperable and net-centric SOA solutions. The service is just a "citizen" of the "SOA state." All "citizens" of the "SOA state" should cooperate, just as "SOA states" occasionally might need to be able to federate with other "SOA states." This analogy highlights a quite complex and delicate problem stated simply as: How does one make sure "citizens" of the "SOA state" work well together and how does one enable "SOA states" to collaborate? The process should be employed to select the right combination of patterns, but should make sure that all elements are considered in a complex environment. There is no one correct answer to making sure all bases are covered, due to cost and performance. Limiting the scope too much might ease interoperability, but may hurt net-centricity.

[35] From NSD-RM

4.1 Pattern Topics

For the Net-Centric Services Framework, there will be specific topics that should be addressed by patterns which depict Net-centric and Interoperable Services. These are contained in the pattern matching check list below. Use of this list of specific topics is addressed in the section 5 (process) below. The checklist may be used as a pragmatic approach to at least cover the important topics for the SOA and its “citizens”. One should go through the list and make sure the selected patterns covers the requirements for a particular service. *Some topics might not be of relevance for a given scope*. An attempt should be made to see the bigger picture. Services need to be good SOA citizens; and keep in mind that net-centricity is not the same as interoperability.

There are degrees of interoperability that apply to communication, information and applications in ascending order of complexity. This interoperability enables net-centricity, dependent on the interoperability level required. Applying net-centric principles at each of these levels enables interoperability at degrees ranging from simple interaction to complex collaboration.

NCOIC patterns should be net-centric, but there may be alternative patterns to select from. Each pattern might offer interoperability within a specific scope and context. There may be an overlap and there may also be differences resulting in non interoperable solutions. Selecting widely used and generic patterns maximizes net-centricity in such a case. A trade study should be made to achieve the best combination of patterns to achieve both interoperability and net-centricity. A net-centric pattern checklist consists of the following major topics:

• Service Related Topics	
• Discovery	(see Section 4.2.4)
• Granularity	(see Section 4.2.8)
• Orchestration	(see Section 4.2.3)
• Choreography	(see Section 4.2.3)
• Service/User Trust	(see Section 4.2.8)
• Autonomy	(see section 3.2.6)
• Location Transparency	(see Section 3.1.12)
• Symmetry	(see Section 4.2.10)
• Dynamism	(see Sections 3.2.2, 4.2.11)
• Robustness	(see Sections 3.2.8, 4.2.12)
• Loose Coupling	(see Sections 3.1.5, 4.2.13)
• Platform Independence	(see Sections 3.2.1, 4.2.14)
• Net Centric Topics	
• Policies and Aspects	(see Section 4.2.1)
• Users and Applications	(see Section 4.2.2)
• Existing Systems	(see section 4.2.5)

• Integration	(see Section 4.2.6)
• System Management	(see Section 4.2.7)
• Net-Centric Data	(see Sections 3.2.9, 4.2.15)

Note that, as shown in Figure 1, additional planned specialized frameworks will greatly expand the discussion of some of these topics. For example, Section 4.2.7 will be the subject of an entire specialized framework. Each topic is further explained below.

4.2 Pattern Checklist

This checklist is used to show that the pattern selection covers relevant aspects of each topic. Assess how the current selection of patterns affects interoperability and net-centricity to find if there is a better match. Keep in mind that the result will be limited to assurance that the architecture will be net centric, assessment of Net-centric Service maturity and confidence in the service oriented architecture. Note that there is an overlap between the topics in the pattern matching checklist and the principles discussed earlier. There is no conflict between the principles and checklist; in fact there is complementary use of items such as orchestration.

Annex B shows a mapping of NCSF principles to the Pattern Check List.

4.2.1 Policies and Aspects

Web services are commonly selected based on their functionality. For example the web service description provided in the WSDL[36]-format does not support explicit quantitative requirement such as constraints based on Quality of Service (QoS) and management statements, classes of service, access rights, pricing information, SLAs and other web services contracts. Explicit specification of quantitative requirements by a service provider in a precise and unambiguous way to allow composition and integration of services in an intelligent and customized manner must be stated in a effective service. In this way, applications can specify criteria considered at service selection time to integrate services that best fit the application requirements. The mechanisms enabling such a behavior should be governed by policies.

The policy topic addresses capabilities required to govern services by policies and aspects such as QoS, trust and security (such as Information Assurance, user authentication, access rights etc.), modes of operation, discoverability, and service selection. The policy topic is also related to all other topics since policies apply to them all (note that a non existent policy should also be considered a policy).

[36] Web Service Definition Language

4.2.2 Users and Applications

A service may be consumed either by a human user or a machine. A service pattern may be quite different for each of these consumer types; while having many common elements. Applications may have may be developed along differing patterns when the consumer is a machine rather than a human. How do users and applications interact with the service? For example, business processes, user interfaces, information representation, meta-data and semantic interoperability should be considered. Note that there may be an overlap between considerations of Users and Applications and the Policies and Aspects topic. As illustrated in Figure 2, Services enable Applications and a net-centric application uses one or more services.

4.2.3 Orchestration and Choreography

Services may be bundled through orchestration and choreography to act together as a single service. These composite services support specific use cases and business processes. A more extensive description of how Orchestration and Choreography are used in service composition is found in Section 3 along with other composition topics. Governance, in particular should be carefully considered in service pattern development from both aspects addressed in section 3.

4.2.4 Discovery

Exposed services can be discovered or be statically bound and then invoked. This topic also provides for the mechanism to take enterprise scale components, COI[37] specific components etc., and externalizes a subset of their interfaces in the form of service descriptions. Thus, the enterprise components provide service realization at runtime using the functionality provided by their interfaces. An extensive discussion of Discovery is found in Section 3.

4.2.5 Existing Systems

System or enterprise components in operational use are considered to be legacy systems. Characterization of these systems range from those in place prior to the advent of widespread acceptance of Internet Protocol (IP) based systems to systems modified through many different means to adapt to IP based systems. A pattern planned by NCOIC addresses processes to upgrade Legacy Systems to IP based systems. This topic addresses the enterprise-scale assets that are responsible for realizing functionality and maintaining the QoS of the exposed services, and as such, conformance to SLAs.

[37] Community of Interest

4.2.6 Integration

This topic represents the means for services to integrate with applications and other services. Not from an operational point of view (that is covered by the Business Process Choreography area), but from a technical point of view. It covers aspects such as binding and implementation components for making services available location independently.

4.2.7 System Management

This topic provides the capabilities required to monitor, manage, and maintain QoS such as security, performance, and availability of the services. NCOIC has planned to develop a Specialized Framework for this topic. Several government agencies such as U.S. DoD, NATO, and the U.K. MoD are considering core services, usually entitled “System Management Services” that address QoS in detail.

4.2.8 Services Design, Categorization and Aggregation

One of the key topics in Net Centric Services is the Granularity of Services. Net-centric services may place additional constraints on the service granularity. Granularity is one of the key factors in service performance. It is addressed in Section 5. It will also be addressed in the planned Legacy Systems Pattern.

4.2.9 Trust between Services and Service Users

This topic is the key to the successful implementation of services. Unless a community of practice develops standards and verification thereof for services so that users can trust their results over and above the SLA, this notion will surely fail. Questions to analyze when studying the selected pattern include the following:

- How can discoverable services be provided so that consumers know the service will operate as expected?
    - Can sufficient metadata, test cases & results be exposed to enhance trust?
    - How can enterprise policy be documented for the user in a way to enhance trust?
    - Are there other techniques to show that provider has done the work to make the service work as intended?
- How do SLAs and Contracts build trust and means for a consumer to interact with the provider if the service does not conform to the contract?
    - This might be effected by adding SLA terms including:
        - Depreciation period, support hours and other “soft” items
        - Hard metrics of response time and throughput.
- Can sharing of Service Level Metrics be used to show consumers how the service is performing?

- Are potential consumers allowed to access a testing endpoint of the service so that they can test and validate their service composition?

4.2.10 Symmetry

Services implemented in a net-centric pattern should exhibit characteristics that are symmetric among involved entities.

4.2.11 Dynamism

Services implemented in a Net-Centric Pattern should support dynamic behavior (as manifested by minimal *a priori* assumptions). To be useful dynamism must be accompanied with a metric: any real system will have several levels of dynamic capabilities. The net-centric environment will have multiple time scales of dynamic behaviors.

4.2.12 Robustness

Services implemented in a Net-Centric Pattern should operate in a way such that failure of a service provider does not result in failure of the consumer itself. Conversely, service consumers must be implemented in a way that a connected consumer failure does not result in a provider failure.

4.2.13 Loose Coupling

Services implemented in a Net-Centric Pattern should acquire knowledge of other services while still remaining independent of those services.

4.2.14 Platform Independence

Services implemented in a Net-Centric Pattern accessible over the network via protocols that are system architecture independent

4.2.15 Net-Centric Data

Available data from systems on the network should be exposed and be available to other systems on the network. This includes existing data produced by the system and new types of data or data from other connected equipment on a platform. The data (and functionality) exposed on the network may be offered as a service consumable by other services. The level of integration may vary, from support functions completely isolated from legacy applications (e.g., chat and email services), to functions tightly integrated with the legacy applications. The data should be described with metadata to facilitate processing.

The metadata should include information about the operational/institutional scope of the data and the frame(s) of reference used to describe/represent the data, such as units of measure, U.S. Army identification number, or commercial Stock Keeping Unit (SKU). Ideally, it should include a network address/URL for more information about the frame of reference or source of controlled attribute values (e.g., employee numbers, Vehicle Identification Numbers (VINs), etc). In some cases this may include actual data value validation or lookup services.

5 Process

Defining, selecting and using the right principles and patterns are important tasks for the framework user. Processes contain steps and relevant intermediate artifacts that help the framework user to develop Net-centric solutions. Framework Processes provide methodology that supports an approach for developing Net-centric solutions[38].

Most companies have their own internal engineering processes. The NCOIC Interoperability Framework and its ancillary documents such as the Net-Centric Services Framework, should be used as adjuncts to such processes for enabling interoperability. Many companies use an engineering process whose genesis was IEEE1220 or EIA632, with common process steps of *Requirements Analysis*, *Functional Analysis & Allocation*, and *Synthesis*, coupled to a *System Analysis* function for specific topics.

The NCOIC Net-Centric Services Framework fits well within such a process. An input to the Requirements Analysis process step will likely be a buyer's requirement for a Service Oriented Architecture. If not, the Requirement Analysis process itself will determine if such a requirement exists, and, if so, the process will document that requirement in the output of that step, having used the Concepts contained in this document to guide the architecture style requirements and service definitions.

Principles contained in this document are used in Functional Analysis in typical systems engineering processes. During the functional analysis process, the service versus non service content of the system will be determined, coupled with a Systems Analysis of modes of operation. Other aspects of SOAs such as service governance are addressed by the principles.

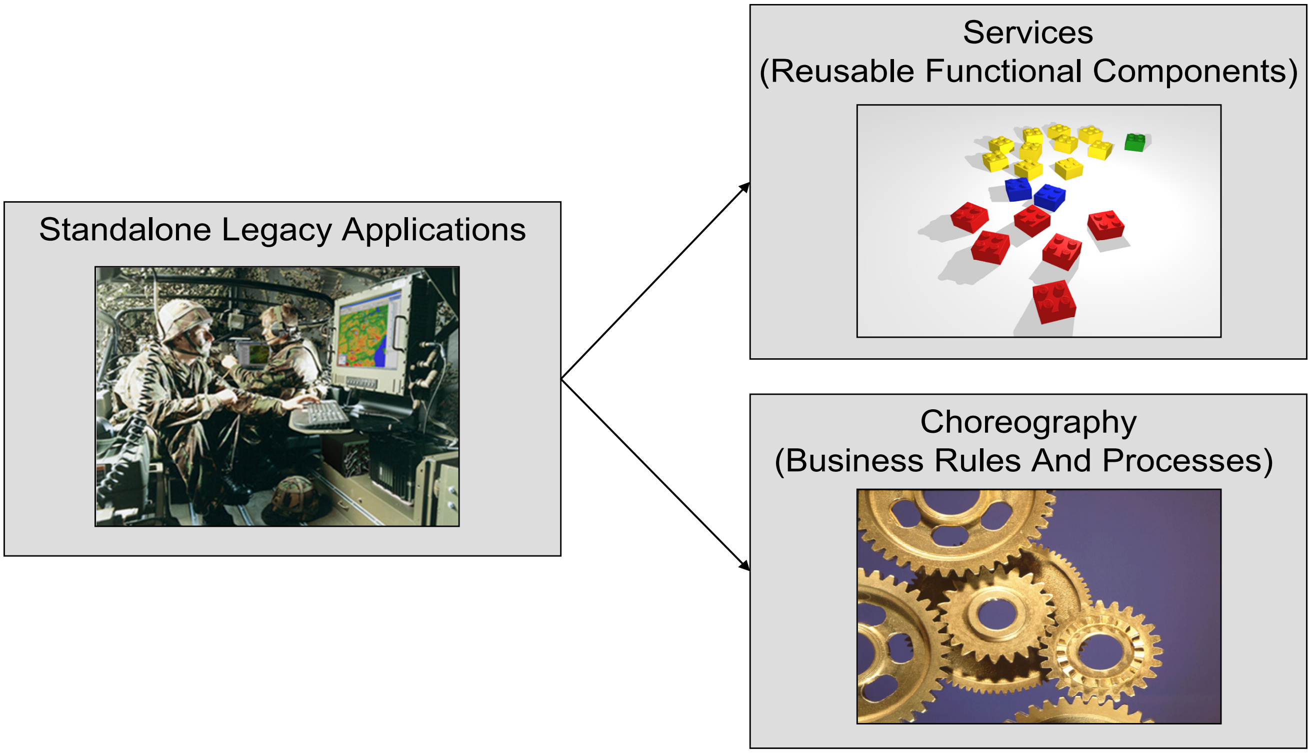
This process section develops a functional allocation to services. The concept of patterns was introduced in Section 4, important to functional service allocation. Within NCOIC, patterns are being developed at multiple levels which range from high level operational patterns to specific service patterns. This process section concludes by determining the service functional definition in terms of granularity. Patterns provide guidance, based upon this and other frameworks, such as the Information Assurance Framework, to the final system engineering implementation step for a system, System Synthesis.

This section covers a set of processes or practices that may be accomplished to obtain interoperability between SOA's (and its services) to promote net-centricity. A summary of the contents follows:

- Domain Decomposition
  - The selection of a Generic SOA Framework
  - Framework Interoperability Assessment
  - Top-down Decomposition
- Analysis of Existing Legacy
- NIF Pattern Selection
- Net-centric Service Litmus Test

[38] Exerpted from NIF version 2

SOA implementation rarely starts from scratch. Creating a SOA solution will almost always involve integrating existing legacy systems by decomposing them (top-down approach) into services, operations, business processes, and business rules as shown in Figure 4 and in the material below:



Services
(Reusable Functional Components)
Standalone Legacy Applications
Choreography
(Business Rules And Processes)

Figure 4 – Service decomposition: Source: NCOIC

- Existing applications are factored into sets of discrete services that represent groups of related operations (bottom-up approach).
- Business processes and rules are abstracted from the applications into a separate BPM, managed by a business choreography model.

There are several general frameworks in existence aimed towards the general task of service decomposition and SOA creation, such as NAF (NATO Architectural Framework) version 3 for NATO standardized frameworks, and frameworks by organizations such as OASIS and The Open Group. This document does not provide guidance when it comes to the general design or realization of a SOA. It is assumed that the user first selects a preferred general SOA framework and with the help of that framework, either provides a draft SOA as input to this framework, or produces a SOA in parallel with this framework.

The Net Centric Service litmus test determines how well a certain service meets a set of criteria. The general service litmus test is usually used to determine if the services are defined and described at a proper granularity level, as shown in Figure 5.

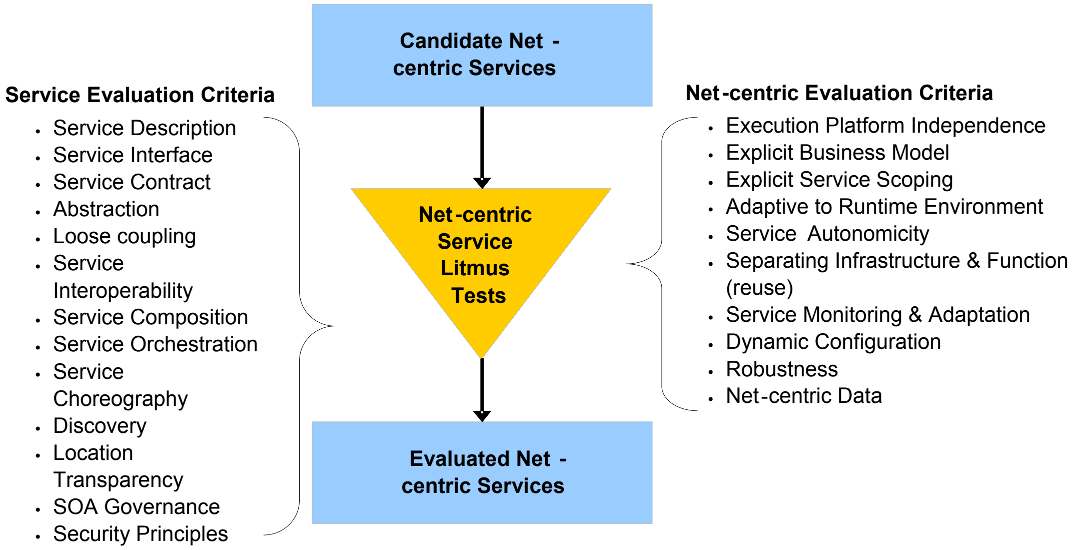


Figure 5 – Service and Net-centric Service Litmus test: Source: NCOIC

The Net Centric Service Litmus test is a structured method to determine what should and should not be a service from existing enterprise architecture components. The Net Centric Service Litmus test is used within this framework to assess general alignment with Net-centric SOA principles and additional net-centric & interoperability criteria for candidate services. The preferred source of net-centric criteria are SCOPE[39] dimensions, either contained in selected Net-centric patterns or from results of a SCOPE analysis. Criteria from NCAT™[40] may be used as criteria for a litmus test. Criteria used for litmus tests can be used in the NCAT™ for later assessment as predefined criteria that might be reused by others.

[39] Network Centric Operations Industry Consortium (NCOIC), Systems, Capabilities, Operations, Programs, and Enterprises (SCOPE) Model

[40] Network Centric Operations Industry Consortium (NCOIC), Net Centric Assessment Tool (NCAT™)

A service-oriented modeling approach provides modeling, analysis, design techniques, and activities to define the foundations of a SOA. It helps by defining the elements in each of the SOA layers and making critical architectural decisions at each level. It does so using a combination of a top-down, business-driven manner of service identification coupled with a stream of work that conducts service identification through leveraging legacy assets and systems. In this way, high-level business process functionality is externalized for large-grained services. Smaller-grained services, those that help realize the higher level of services, are identified by examining the existing legacy functionality and deciding how to create adaptors and wrappers, or componentizing the legacy to externalize the desired functionality often locked within the system.

A service oriented architecture framework contained in the engineering process of most firms describes a modeling and architecture method. A Net-Centric Services Framework can then be “overlaid” on top of that framework in order to assure a net-centric and interoperable architecture. This is done by pattern matching and selection from a future pattern repository provided by NCOIC.

Figure 6 represents a high level overview of the processes of interest for the Net-Centric Services Framework. The Overarching Process receives input from the Overarching Framework and from other NIF Specialized Frameworks. The Overarching Process is also the process that receives the output from the Net-Centric Services Framework. The top-down and bottom-up processes are processes specified by the Generic SOA Framework of choice by the user. It is complemented by this framework process to achieve a Net-centric SOA with interoperable services.

This framework process is broken down into two high level processes, Pattern Selection and the Service Litmus Tests. These processes rely on the pattern repository mentioned above and should also provide feedback to the repository such as rating of existing artifacts and requests for new artifacts.

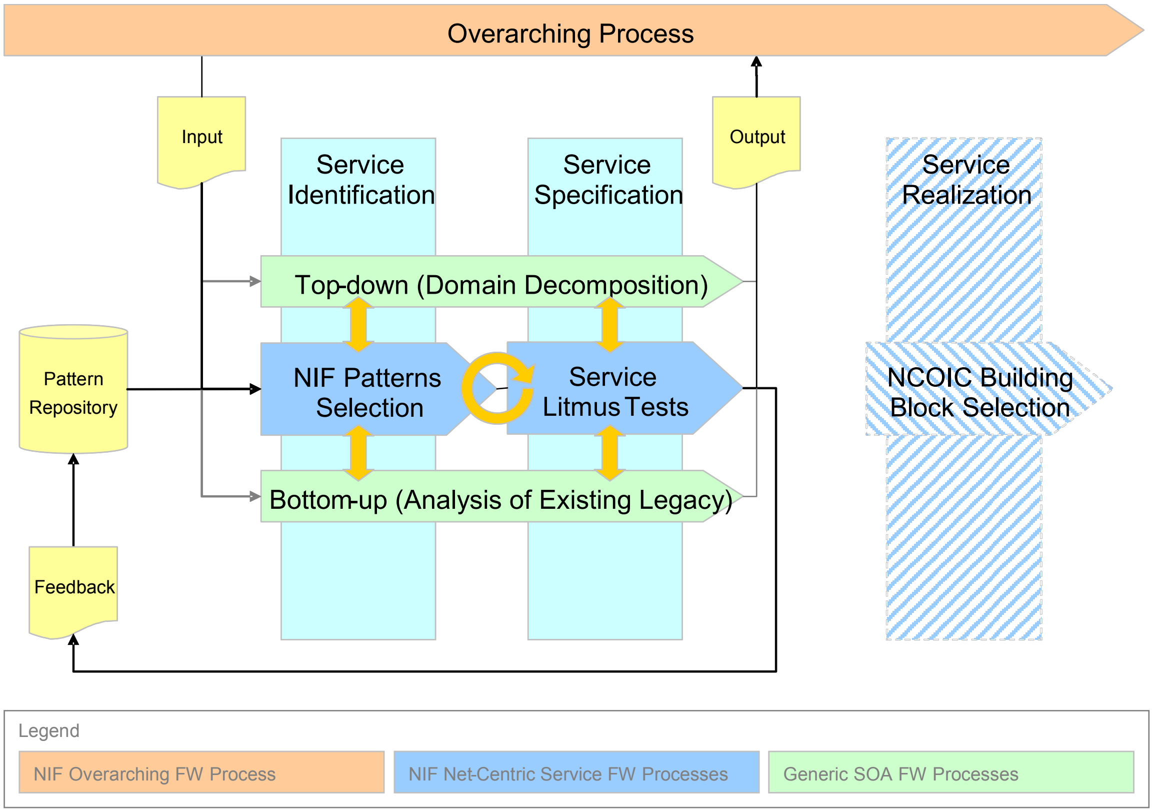


Figure 6 - High level process overview: Source: NCOIC

Figure 6 points out that the Service Realization phase is not covered by this framework. NIF is delimited to system architecture and the Net-Centric Services Framework only covers the relevant parts of the identification and specification phases.

5.1 Step 1 – Domain Decomposition

Domain Decomposition contains inputs and outputs listed in tabular form and described in detail below.

Select a generic SOA Framework, perform an assessment of interoperability for the framework, and complete a top-down decomposition.	
Input	Output
• SOA Framework • Operational Requirements • Scenarios • Key Performance Parameters (KPP) • SCOPE Analysis	• Compliant SOA Framework • Draft SOA with high level enterprise service candidates identified • Use cases for identified services • Applicable Operational Descriptions (OD)

5.1.1 Selection of a Generic SOA Framework

The Top-down Domain Decomposition should be carried out in accordance with a generic SOA Framework. The NIF user should choose a generic SOA Framework at this point.[41]. This generic SOA Framework becomes the selected generic SOA Framework, the output from this step.

There are several important reasons to decompose using a framework. First, such a framework serves as a communications tool. It is easier to explain the service if the listener is familiar with the context of the service. Second, it provides an industry vetted basis that lends credibility to the results of the service granularity determination. Finally, the decomposition should have improved consistency rather than starting from a clean sheet of paper.

5.1.2 Framework Interoperability Assessment

The selected generic SOA Framework is analyzed using Concepts and Principles described earlier in this document. The SOA Framework is considered a compliant SOA Framework as soon as it satisfies the conditions described in these sections. The output from this step is a compliant SOA Framework.

The generic frameworks input to this step will have more generality than needed. For example, if a service describes a fusion process, it may not need a heavy emphasis on messaging that a generalized framework will normally have. The compliant framework will focus in the interoperability of the particular service in question. In other words, the framework will have an appropriate scope.

5.1.3 Top-down Decomposition

Top-down Domain Decomposition is carried out in accordance with the compliant SOA Framework[42]. In the top-down view, a blueprint of business use cases provides specifications for business services. This top-down process is referred to as domain decomposition, consisting of decomposition of the business domain into its functional areas and subsystems, including its flow or process decomposition into processes, sub-processes, and high-level business use cases.

These use cases are candidates for business services exposed at the edge of the enterprise, or for those used within boundaries of the enterprise across lines of business. Resources for this work can be drawn from NIF and other NCOIC artifacts. Examples of such artifacts are results of SCOPE analysis, reusable pattern assets, or operational descriptions. A new operational description may be created if necessary. The outcome of this activity should result in:

[41] E.g. Zachman Framework, Gartner/Meta Group Framework, Open Group's TOGAF, vendors

[42] And if needed, in conjunction with traditional approaches such as BPM, EA, and OOAD

- A compliant SOA with high level enterprise services identified
- Use cases for identified services
- Applicable ODs.

5.2 Step 2 – Analysis of Existing Legacy

The analysis of existing legacy is carried out using the selected SOA Framework[43]. Existing systems are analyzed and selected to determine if they are viable candidates for providing better value solutions to the implementation of underlying service functionality supporting the business process. In this process, API's, transactions, and modules from legacy systems and applications are analyzed to determine if they can be leveraged. In some cases, componentization of the legacy systems is needed to re-modularize existing assets for supporting service functionality.

Step 2 – Analysis of Existing Legacy	
Input	Output
• Draft SOA with high level enterprise service candidates identified • Use cases for identified services • ODs • KPPs • Definitions • Human Machine Interface • Application Interfaces • Data Separability • Language	• Enterprise-scale assets inventory (systems, components etc.) • Additional service candidates (technical, infrastructural etc.) • API, Interface and Protocol specifications • Data and Information models • Input for modeling and simulation • Updated ODs

This step incorporates a "middle-out" (similar to the process of identifying derived requirements in traditional system engineering) modeling step to validate and unearth other services not captured by either top-down or bottom-up service identification approaches. It ties services to operational requirements, capabilities, key performance parameters and other relevant metrics. Parameters also to be considered are definitions, human machine interface, application interfaces, data separability and language. Available resources provided by the previous step is a selected SOA with high level enterprise service candidates identified, use cases for identified services & applicable OD's. The outcome of this activity results in:

- Enterprise-scale assets inventory (systems, components etc.)
- API, Interface and Protocol specifications
- Input for modeling and simulation
- Additional service candidates (technical, infrastructure etc.)
- Data and Information models
- Updated set of applicable ODs

[43] E.g. Zachman Framework, Gartner/Meta Group Framework, Open Group's TOGAF, vendors

5.3 Step 3 – Pattern Selection

Existing patterns are analyzed and matched against the SOA and additional artifacts such as ODs, Use Cases, KPP, Systems, Interfaces, Protocols, Information and Data Models.

Step 3 – NIF Pattern Selection	
Input	Output
• Draft SOA with service candidates • Use cases for identified services • ODs • SCOPE Analysis • Enterprise-scale assets inventory • KPP • API, Interface and Protocol specifications • Data and Information models • Input for modeling and simulation	• Refined SOA with draft selection of NCOIC Patterns and associated services • Input for test and verification

The pattern matching checklist may be used on the SOA and its services to assess interoperable functional patterns. Patterns need to be selected as answers to the following questions:

1. Are *policies and aspects* specifically addressed?
2. How are services offered to and consumed by *users and applications*?
3. How to compose services using *Orchestration and Choreograpy* to support business processes?
4. How are services *discovered and bound* with other services?
5. How to realize service functionality and maintain QoS with *existing systems*?
6. How are services *integrated* with other services and applications from a technical point of view?
7. How to provide *System Management* to monitor, manage, and maintain QoS?
8. How are services designed, categorized and aggregated?
9. Who can do what and how to build trust between services and service users?

New NCOIC patterns can be created or requested. The outcome of this activity results in a refined SOA with pattern candidates and input for test and verification.

5.4 Step 4 – Net Centric Service Litmus Test

The Service Litmus Tests are carried out to determine how well the identified set of services meets the criteria for the Net-Centric SOA Principles and SCOPE dimensions.

Step 4 – Service Litmus Tests	
Input	Output
• Refined SOA with selected NCOIC Patterns and associated services • SCOPE Analysis • Maturity Models • KPPs	• Assured net-centric SOA with interoperable services • Final selection of NCOIC patterns • Input for NCAT evaluation
Note: Criteria should at least cover net centricity; CMM is optional but should be utilized if one exists	

The result of a SCOPE-analysis and the Net Centric SOA Principles could make up the foundation for applicable criteria that the identified services could be measured against, e.g.:

- How execution platform independent is the service?
- How explicit is the service business model?
- How explicit is the service scoping?
- How adaptive is the service to its runtime environment?
- How autonomic is the service?

Litmus test criteria can be a valuable source as input for NCAT evaluation. Step 3 and 4 must be iterated until selected set of patterns and services pass applicable litmus tests. Litmus tests can be used to assess and assure interoperability as well as Net-centricity. The outcome of this activity should result in:

- Assured net-centric SOA with interoperable services
- Final selection of NCOIC patterns
- Input for NCAT evaluation

6 ANNEX A – Comparison of SOA and Service terms

The purpose of this annex is to demonstrate the diversity of SOA and Service terms. For terms and definitions in general, please refer to the NCOIC Lexicon.

Term	Definition	Source
(SOA)	SOA is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations. The SOA-RM specification bases its definition of SOA around the concept of "needs and capabilities", where SOA provides a mechanism for matching needs of service consumers with capabilities provided by service providers.	OASIS
SOA	SOA is an architectural style for a community of providers and consumers of services to achieve mutual value, that: • Allows participants in the communities to work together with minimal co-dependence or technology dependence • Specifies the contracts to which organizations, people and technologies must adhere in order to participate in the community • Provides for business value and business processes to be realized by the community • Allows for a variety of technologies to be used to facilitate interactions within the community	OMG
SOA	SOA is a form of distributed systems architecture that is typically characterized by the following properties: Logical view: The service is an abstracted, logical view of actual programs, databases, business processes, etc., defined in terms of what it does, typically carrying out a business level operation. Message orientation: The service is formally defined in terms of the messages exchanged between provider agents and requester agents, and not the properties of the agents themselves. The internal structure of an agent, including features such as its implementation language, process structure and even database structure, are deliberately abstracted away in the SOA. Using the SOA discipline, one does not, and should not, need to know how an agent implementing a service is constructed. A key benefit of this concerns so-called legacy systems. By avoiding any knowledge of	W3C

Term	Definition	Source
	the internal structure of an agent, one can incorporate any software component or application that can be "wrapped" in message handling code that allows it to adhere to the formal service definition. Description orientation: A service is described by machine-processable meta data. The description supports the public nature of the SOA. Only those details that are exposed to the public and important for the use of the service should be included in the description. The semantics of a service should be documented, either directly or indirectly, by its description. Granularity: Services tend to use a small number of operations with relatively large and complex messages. Network orientation: Services tend to be oriented toward use over a network, though this is not an absolute requirement. Platform neutral: Messages are sent in a platform-neutral, standardized format delivered through the interfaces. XML is the most obvious format that meets this constraint.	
SOA	Fundamentally, SOA is a design style for creating shared, reusable, distributed services. A service is reusable functionality that is exposed and invoked through a programmatic interface. The service's programmatic interface is a service descriptive "contract" that is abstracted from the service's internal implementation.	Burton Group
	SOA is software topology (architecture) that uses interactive business components designed to be meaningful, usable and useful across application or enterprise boundaries. Sometimes, event driven business components are also considered to be event driven services.	Gartner
SOA	A service oriented architecture is a collection of services that communicate with each other. The services are self contained and don't depend on the context or state of the other service. They work within a distributed system architecture.	Data Management Review
SOA	SOA is just that, an architecture. It is more than any particular set of technologies, such as Web services; it transcends them, and, in a perfect world, is totally independent of them. Within a business environment, a pure architectural definition of a SOA might be something like "an application architecture within which functions are defined as independent services with well defined invokable interfaces which can be called in defined sequences to form business processes.	IBM

Term	Definition	Source
	SOA is an architectural style for building software applications that use services available in a network such as the web. It promotes loose coupling between software components so that they can be reused. Applications in SOA are built based on services. A service is an implementation of a well defined business function, and such services can then be consumed by clients in different applications or business processes	SUN
SOA	In Service Oriented Architecture autonomous, loosely coupled & coarse grained services with well defined interfaces provide business functions & can be discovered and accessed through supportive infrastructure. This allows internal & external system integration and flexible reuse of application logic through the composition of services.	Malte Poppensieker, University of Trier
Service	A contractually defined behavior that can be provided by a component for use by any component, solely based on the interface contract. Service interaction models include Request/Response, Publish/Subscribe, Thread/Process and finally Stream.	NATO Network Enabled Capabilities Feasibility Study
Service	The means by which the needs of a consumer are brought together with the capabilities of a provider.	OASIS Reference Model for Service Oriented Architecture 1.0
Service	A collection of operations, accessible through an interface, that allows a user to evoke a behavior of value to the user.	Geographic Information Services ISO – 19119:2005, also in OGC Service Architecture
Service	Is a logical representation of a repeatable business activity that has a specified outcome (e.g., check customer credit; provide weather data, consolidate drilling reports), is self-contained, may be composed of other services, and is a “black box” to consumers of the service	Open Group Wiki, 08-Jun-2006

Term	Definition	Source
Service	A definition of a service is broken apart into distinct segments: service outcome: defining the intended real world effects or information provided by the service; service identification: identifying and uniquely naming a service; describing the set of functionality offered; describing responsibilities; identifying the service's consumers and the information consumed and provided; service properties: identifying specific properties of a service that may differ from one instance or implementation of a service to another. This includes quality of service properties, such as: • performance • security • availability • reliability • maintainability • latency • confidentiality • integrity. Service interfaces: specifying the interfaces through which the service consumer may exchange information with this service; Service interaction: showing the allowed interaction between the service consumer and the service provider. This can be done using sequence diagrams, activity diagrams or state diagrams; service policies: specifying the policies regarding security, commercial conditions, applicable laws, etcetera, under which the service is provided.	NATO Architectural Framework (NAF) version 3

7 ANNEX B – Mapping principles from source documents to the NCSF

The following matrix provides a single point of reference to trace the development of the NCSF principles, from the NIF (NSD RM) to the Chapters of the NCSF. Since an over-arching set of Net-Centric principles are being developed and discussed within the NCOIC Working Groups, the matrix informally traces to these principles as well.

Key to using this matrix is the understanding that the NSD-RM includes Overarching Principles in Section 2.3. These principles apply to any system or enterprise that strives to achieve net-centricity.

Details include Basic Tenets in Section 2.3.4, Architecture Principles in Section 2.3.5 and Architecting Principles in Section 2.3.6. The matrix maps Architecture Principles from Section 2.3.5.x.

This NCSF, as mentioned in the Introduction, describes concepts and principles from several viewpoints: Section 2 describes the concepts; Section 3 contains the principles, separated into Service Principles and Net Centric Principles; Section 4 describes pattern requirements; Section 5 contains the processes used when implementing a Net-Centric Services Framework solution.

The mapping matrix is intended as a guide to the reader to trace principles as well as highlight the distinctions among the viewpoints described in Section 2 - Concepts, Section 3.1.x – Service Orientation principles, Section 3.2.x – Net-centric Service principles and Section 4.2.x – Pattern Check List.

Principle	NSD-RM 2.3.5	NCSF Ch 2-concepts	NCSF Ch 3.1 SOA principles	NCSF Ch 3.2 Net-centric principles	NCSF Ch 4.2 Pattern Check List
1. Explicitness		visibility	.1 Description		.4 Discovery
	.10 QoS			.3 Explicit svc business model	.1 Policies
					.2 Users and Apps
	.4 NCEI			.4 Explicit svc scope	.4 Discovery
					.6 Integration
	.1 Standards				.8 Services Design
				.9 Net-centric data	.15 Net-Centric Data
2. Symmetry	.5 Autonomy	mutual sustainment			.10 Symmetry
				.1 Platform independence	.14 Platform Independence
				.3 Explicit svc business model	.1 Policies
				.6 Autonomicity	.1 Policies
3. Dynamism	.2 Interfaces				.11 Dynamism
			.10 Discovery		.4 Discovery
				.2 Dynamic configuration	.3 Orch. & Choreg.
		environment awareness		.5 Environment Monitor	.8 Services Design
				.8 Robustness	.12 Robustness
4. Globalism			.4 Abstraction		.8 Services Design
			.12 Location transparency		.4 Discovery
5. Ubiquitous access	.2 Interfaces		.2 Interface		.8 Services Design
	.7 Mediation		.6 Interoperability		.6 Integration
			.10 Discovery		.4 Discovery
	.5 Autonomy	autonomy		.6 Autonomicity	.1 Policies
					.4 Discovery
	.6 Resilience			.8 Robustness	.3 Orch. & Choreg.
6. Entity Primacy			.5 Loose coupling		.13 Loose Coupling
	.5 Autonomy			.6 Autonomicity	.1 Policies
7. Explicit Relationship	.4 NCEI	visibility	.3 Contract		.7 System Management
	.9 Governance		.8 Orchestration		.3 Orch. & Choreg.
			.9 Choreography		.3 Orch. & Choreg.
			.11 SOA Governance		.1 Policies
					.7 System Management
		trust	.13 Security		.9 Trust
				.3 Explicit svc business model	.1 Policies
				.4 Explicit svc scope	.1 Policies
8. Scale Independence			.4 Abstraction		.8 Services Design
	.3 Legacy integration		.7 Composition		.8 Services Design
				.1 Platform independence	.14 Platform Independence
9. Pragmatism				.7 Reuse	.5 Existing Systems
					all

.8, .11, .12 not addressed in
NIFv2 NSD-RMv1.0

Table B-1 Principle Mapping Matrix: Source NCOIC

Note that there can be multiple elements in the Pattern Check List that pertain to a given principle. Only the first or most obvious PCL element is listed in the table.

8 ANNEX C – Additional Detail for NCSF Entities

The figures in this Annex are derived from the basic SysML Block Definition Diagram in Figure 2. They provide detail and entity continuity and are referenced from the text in order to capture some more specific analyses that occurred during the development of this document, but whose inclusion in the normal text flow would possibly bog down the reader.

Figure C-1 shows several relationships between the Service and Business_Goals entities. This expands the basic "supports" relationship.

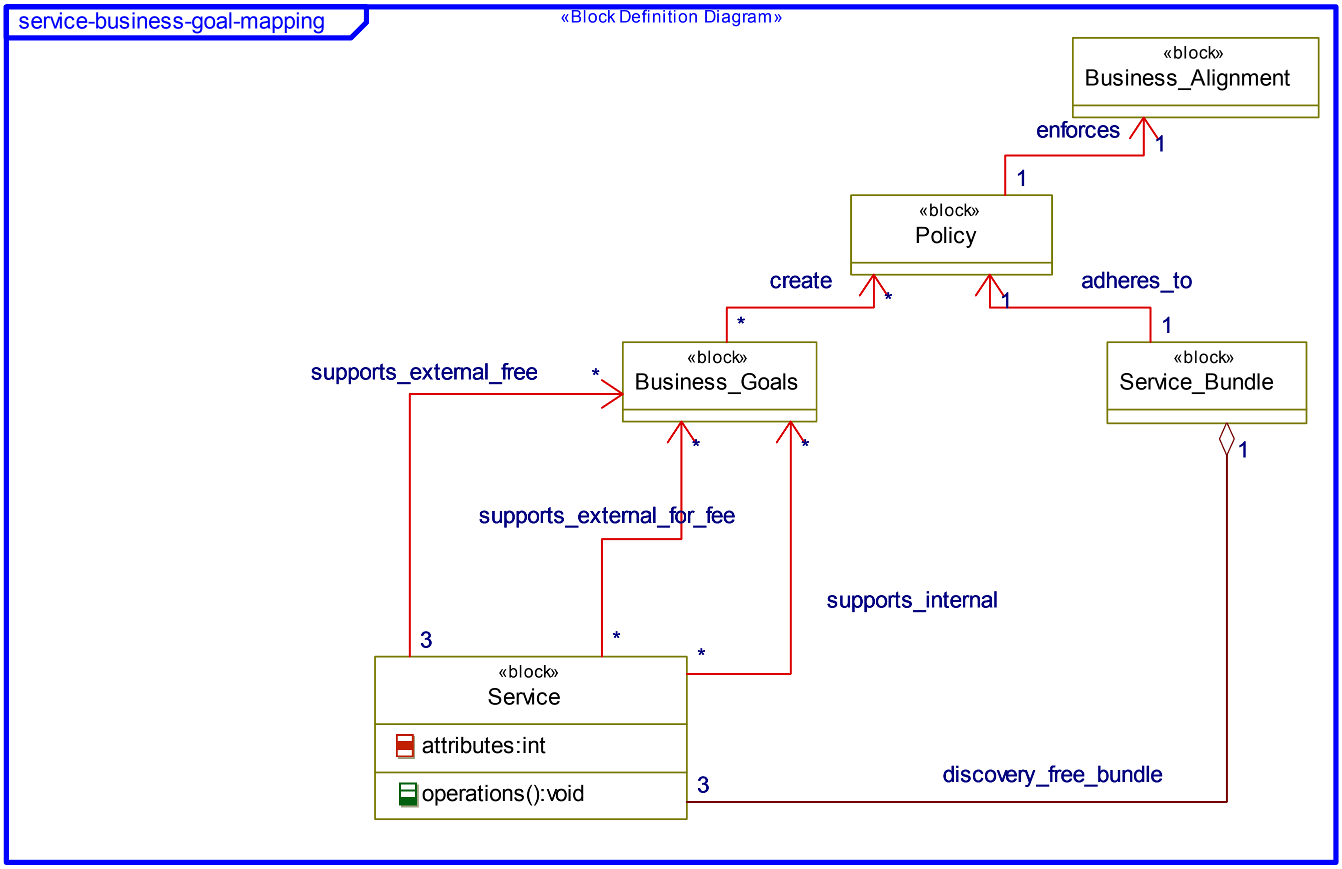


Figure C-1: Example Service-Business Goal Relationships. Source: NCOIC

Defining a range of relationships helps clarify the roles of Service and Business_Goals in an enterprise. One entity might offer services that can be used for free (in this example, only 3 free); some for a fee (buy as many as you like) and internal services. The object “Service_Bundle” is an aggregation of services that can be put together only according to the roles defined in the collection of aggregation relationships.

And a user only uses those Service_Bundle’s that adhere to its Policy.

A developer of an NCSF-based pattern would like to specify which business goals are supported by which services. Are the business goals associated with any specific enterprise types such as banks or service enterprises, versus manufacturing or distribution enterprises, versus government organizations, etc. If we say business goals create (and update/evolve) policy, and policy enforces business alignment, how do we specify the types and ranges of policies and policy attributes or options that are needed for some set of business goals, for some set of enterprise types? Services and service bundles are shown as supporting business goals and adhering to policy

Without a way to specify the scope of business goals and the degree to which services support them, and for what enterprise types, we have no idea whether a particular service or service pattern might be appropriate or applicable in a given situation. Figure C-1 only works if we have a specific enterprise/business in mind with specific goals and policies, and we design services with these scope assumptions in mind *a priori*. This is the typical system engineering scenario for which SysML was built.

The diagram does show that services could have potentially different business models for their consumption, but there are service attributes that suggest whether a particular business model might be appropriate for that service or not. For example, if the service is very domain specific and expensive to provide, a fee for service model might be appropriate. If it is not expensive to provide, and has wide applicability, a common services (centrally funded) model or a third party payer model (such as Google) might be appropriate. If the service is critical to the proper operation of the enterprise (e.g., regulatory or safety compliance), it might be free but mandatory to use, and the service provider centrally funded in line with the expected usage level of that particular service.
Figure C-2 shows the next level of Figure C-1 and attempts to address these questions. Three specific Net Centric Enterprise Services (NCES) entities inherit from the Service entity and comprise a NCES Content Discovery Service bundle. They each have a specific relationship to the DKO (Defense Knowledge Online) Business Goals.

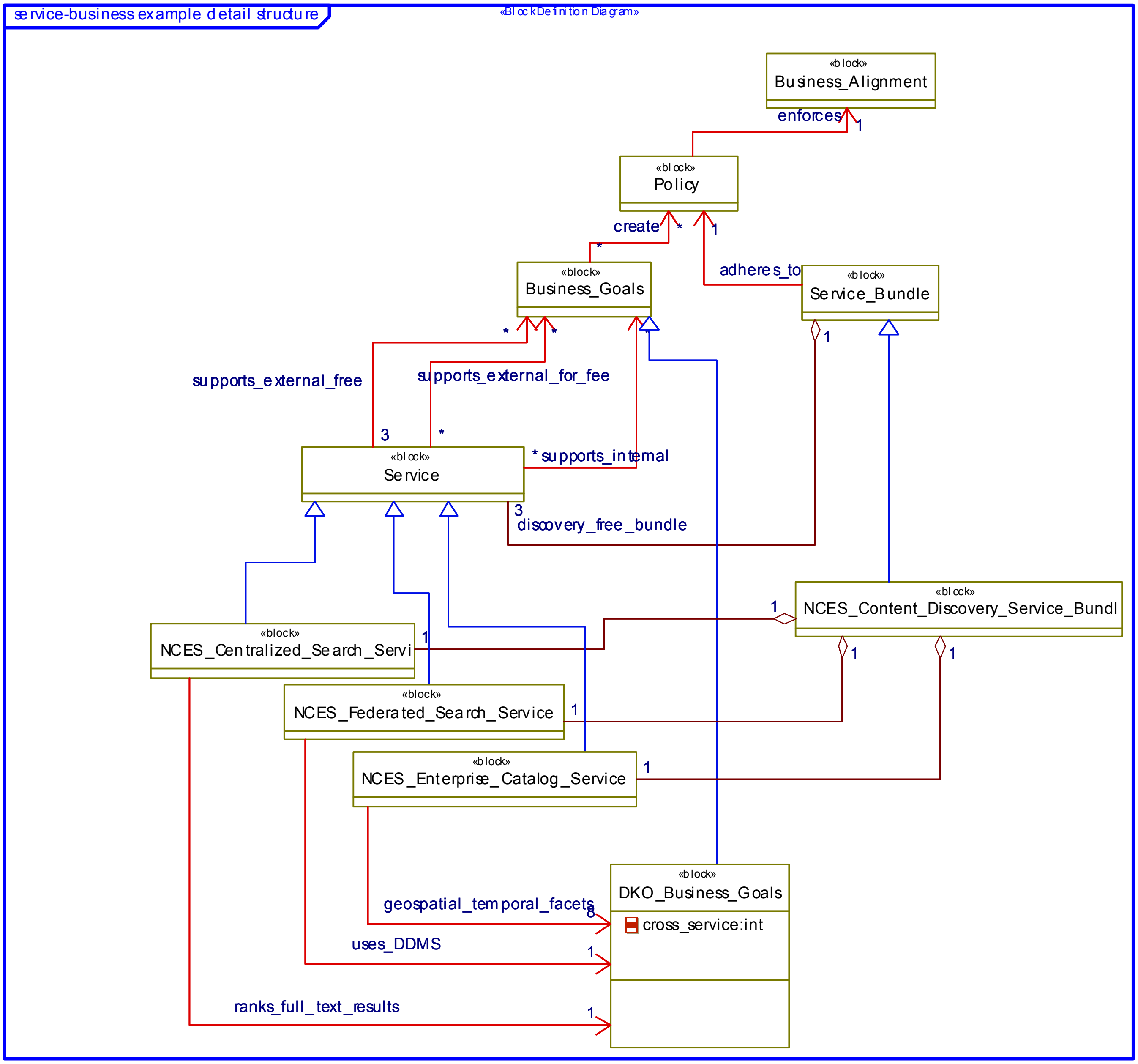


Figure C-2 Service and Business Goal Example detail. Source: NCOIC

This is an example only of the type of detail a NCSF-based Pattern might develop to illustrate its use of services to support Business Goals. This example illustrates the DKO enterprise's use of Services and Service_Bundles to meet Business_Goals.

DKO_Business_Goal would be a "real-world" instantiation of the generic Business_Goals. They might have an attribute such as "access must be cross-Service" (Service = Army, Navy, Air Force, Marine).

The real-world orchestration of several NCES services into a Bundle (this is a real bundle at http://www.disa.mil/nces/product_lines/content_discovery.html) might have the illustrated composite relationships (e.g., the catalog service can search on up to 8 facets for information discovery).