Net-Centric Principles

Version 2.0

Abstract: In order to more easily identify attributes and characteristics of net-centricity and systems that support net-centric operations, those underlying and universally valid principles required are identified. The intent of these principles is in selecting attributes that reflect desirable net-centric characteristics.

Principal Authors

Hans Polzer, Lockheed Martin

Todd Schneider, Raytheon

Contributing Authors

Aaron Budgor

Terry Longstreth

Richard Larkin

Douglas Marquis

Approved for Public Release

Distribution Unlimited

NCOIC-NCP-v2.0-20120606

Table of Contents

Table of Tables

1 Introduction

The intent and goal of identifying principles of net-centricity is to simplify the task of distinguishing those attributes or characteristics of systems that are of value in net-centric operations and environments. In order to provide principles for the identification of attributes of value in a net-centric environment the most general and non-context constrained view must be taken in order to uncover these principles. Thus, in lieu of a constraining context, some principles listed herein may seem absurd or impractical if taken as an attribute in and of itself.

The impetus for developing these principles came from a request from the DoD Networks and Information Integration (NII) office to review their Net-Centric Attribute checklist. In the process of reviewing these attributes common and repeated notions underlying the attributes were recognized: principles. For the most part abductive reasoning was applied to identify the principle(s) underlying an attribute.

One problem in distinguishing those characteristics or attributes important to net-centric systems and operations is the confusion as to what constitutes net-centricity and the conflation of desired outcomes with the/an underlying reality. Consider the following definitions.

1.1 Definitions of Net-Centricity

The concept of net-centric or net-centricity is ill defined. The terms net-centric and net-centricity still have different meanings to different people and organizations, and as noted in some definitions, is inherently evolving.

Three definitions are provided here to help underscore the need for net-centric principles. Though there are different understandings of the extent, context and applicability of the terms ‘net-centric’ or ‘net-centricity’ we believe there are common underlying principles that provide a foundation for the variations on a theme.

NCOIC - Net-Centricity

Net-Centricity is a concept that allows authenticated, trusted and verified information to be shared among properly authorized users, applications and platforms in a flexible, timely, and seamless fashion on a ubiquitous inter-network without a priori knowledge of what information is available or needed.

NCOIC- Network Centric

Related to systems and patterns of behavior that are influenced significantly or enabled by current and emergent networks and network technologies. Often these center around IP-based internetworking, but the term is sometimes used to include any type of enabling network.

DoD Joint Capability Areas[1] Lexicon

(1) Net-Centric: The ability to provide a framework for full human and technical connectivity and interoperability that allows all DoD users and mission partners to share the information they need, when they need it, in a form they can understand and act on with confidence, and protects information from those who should not have it.

DoD Directive 8000.01

net-centric. Relating to or representing the attributes of a robust, globally interconnected network environment (including infrastructure, systems, processes, and people) in which data are shared timely and seamlessly among users, applications, and platforms.

[1] The **Joint Capability Areas** (JCA) are a standardized set of definitions that cover the complete range of military activities.

Wikipedia

> Participating as a part of a continuously-evolving, complex community of people, devices, information and services interconnected by a communications network to achieve optimal benefit of resources and better synchronization of events and their consequences.

Given the ambiguity of net-centricity and the number of 'check lists' for net-centricity, having a way to vet potential attributes or characteristics against core principles should

a. eliminate duplication,
b. provide a way to compare checklists,
c. allow refinement these check lists, and
d. provide characteristics for different views or communities that are consistent and mutually traceable.

1.2 Operational Effectiveness

The end goal of the concept of net-centricity is operational effectiveness, not rigid adherence to abstract ideas of network or information flexibility. However many definitions of net-centricity, net-enabled, or network centric are grounded in descriptions of desired outcomes rather than what it actually is. This paper and its *principles* are an effort to distinguish between these.

Net-centricity is more a way of conceptualizing and thinking about operational and technical realities rather than a specific set of system or operational attributes. As such, the principles have utility over and above their impact in selecting or characterizing system or organizational attributes. Conventional wisdom holds that there is a natural inertia in human institutions and societies which impedes paradigm shifts. To counteract this inertia the "weight" of emphasis needs to be on the net-centric side to reinforce the view of an alternative to enumerating attributes.

In systems development outcomes can be vary greatly depending on the initial perspective. Either starting from a narrow, internally focused perspective and then trying to expand outward to accommodate a net-centric environment that is evolving. Or starting from a larger and more global, net-centric environmental perspective, and then deciding how best to constrain the design to leverage and accommodate the global user. The end result is still constrained by pragmatism and a solid business case. The difference is that of aperture and breadth. What is considered in arriving at the business case is much wider. In general, starting from the larger global perspective fosters solutions that are more innovative, operationally useful, and extend beyond the immediate need at hand.

The belief is that these principles have utility over an above their impact in selecting or characterizing system or organizational attributes. They help foster a more abstract and wider view in thinking about net-centricity and the goal of operational effectiveness.

1.3 Principles vs. Attributes

The term and notion of 'principle' is used in various products from NCOIC with varying meanings and interpretations. Herein, the notion of 'principle' is used in a more constrained fashion with the intent of an interpretation more closely aligned with that of 'axiom' from formal reasoning systems (with the attendant ability to derive consequences from these axioms). In particular, 'principles' differ from 'attributes and characteristics' as described by the following definitions from WordNet 2.1.

> Principle -- A basic generalization that is accepted as true and that can be used as a basis for reasoning or conduct.

Where as an attribute or characteristic is described as

> Attribute, property, dimension -- a construct whereby objects or individuals can be distinguished;
>
> Characteristic, feature -- A prominent aspect of something; a distinguishing quality

Simply put 'principles' allow the selection of 'attributes' or 'characteristics' that are deemed useful. 'Characteristics' or 'Attributes' are used to distinguish or select systems.

Thus, in identifying the core principles of net-centricity the goal is that these 'principles' may be used to distinguish essential and relevant characteristics and attributes of net-centricity.

1.3.1 Contextual Constraints

Another distinction among principles and attributes is that of contextual constraints. In practice, for instance systems development, there are constraints as to what may be considered reasonable or feasible. These constraints derive from the operational context in force. Attributes are applied and used in such an operational context. The principles enunciated in the paper on the other hand should be understood as applicable (to some degree) to all possible contexts that involve a net-centric environment – The Core Net-Centric Principles are (application) context independent (i.e., free of specific practical constraints).

2 Net-Centric Environment

The domain of applicability of the principles enunciated in this paper and the attributes to which the principles may be applied for review is that of the net-centric environment. In order to provide the user with a basis for understanding the context of a 'net-centric environment' the following assumptions are made.

2.1 Net-Centric Environment Assumptions

1. An 'entity' is something that can be individuated; it has a recognized and demonstrable distinct, separate existence, though it need not be a material existence[2].
2. The net-centric environment comprises semi-independent or autonomous entities.
3. The network is itself made up of entities that both exhibit net-centric behavior and enable other entities to contribute to the net-centric environment with which they interact.
4. The entities in a net-centric environment have different scales along different dimensions (when compared among each other).
5. The entities of the net-centric environment change over time, over different time scales and at different rates. This includes disappearing and coming existence in the net-centric environment.

[2] For example, a process running on some server connected to a network can be viewed as an individuated networked entity that exists on the network and exhibits some behavior via the network connection.

3 Net-Centric Principles

The following principles are *not* mutually exclusive. They are interdependent. The interdependencies are context sensitive. Depending on the context or situation or other overriding constraints in which they are applied there may be incompatibilities or tensions among different principles or groups thereof. This can be easily seen by consideration of the Pragmatism principle. In general this principle trumps or at the minimum severely constrains the use, impact, or effect of any other principle. This preeminence of the Pragmatism principle derives from the premise of net-centricity, operational effectiveness.

1. Explicitness

An entity should make all information about itself explicit

Attributes or characteristics should require no a priori assumptions, information, or knowledge for their use. Any assumptions and contextual information that would aid in the understanding or use of a networked resource should be made explicit in a fashion that is usable by network entities.

The goal of explicitness is to remove or at least minimize misinterpretation among the senders and receivers of information or data. The impact of this principle is demonstrated by considering the problem of searching on the web (or even a local hard drive). Most search engines and tools provide results that require a human to determine the relevance, if possible. Descriptions in general derive from natural languages, which provide great latitude for ambiguity. For the net-centric environment ambiguity and misinterpretation are anathema (i.e., the consequences can be deadly).

In the case of service oriented architecture (SOA), explicitness or the lack thereof impacts Dynamism (e.g. discovery). Current SOA technologies usually require either a human-in-the-loop or a priori agreements as to the meaning and purpose(s) of services: There is a lack of explicit description that is machine usable. Hence there is a lack of dynamism.

Another area of non-explicitness is assumptions. Assumptions are artifacts of context and necessarily constrain possible interpretations. The lack of explicit pronouncement or representation of assumptions or the context in which they are made can lead to faulty decisions resulting in additional costs, reduced reuse possibilities, and increased potential misunderstandings.

Legacy systems also provide examples of the lack of explicitness and implicit assumptions. Many of these systems were developed when operating in a net-centric environment was not a requirement. Thus constrained to a much smaller environment it was natural to make many assumptions that did not need to be made explicit in the system or elsewhere (e.g., documentation). Moreover, to the developers of these systems the assumptions were most likely to be so patently obvious (to them), they may not have noticed that they were assumptions.

Being absolutely explicit about everything in a system is most likely impossible. Theoretically, due to the non-well foundedness of languages natural or formal. Practically, due to the amount of time, hence cost. The Pragmatism principle impacts this principle by requiring conscious and rational choices, that would be made explicit, based on explicit delineation of a system's boundaries.

Examples:

- Simple communications

- Not using acronyms
- Including time zones when scheduling meetings (off planet meetings being harder to schedule).

- Making explicit operational or system architecture assumption and dependencies
- Frames of Reference
  - Specifying a particular earth-centric geospatial coordinate system and datum.
  - Providing complete organizational affiliation information (e.g., United Nations) - Unit ID
  - Providing complete temporal and geospatial frames of reference and descriptions (e.g., units of measure)
- The NCOIC SCOPE model encourages explicit reasons for decisions made about the scope of a project or program and any requirements that support it.

2. Symmetry/Reciprocal Behaviors

Relationships among networked entities should exhibit symmetric characteristics and behaviors

Attributes should exhibit characteristics that are symmetric among involved net-centric entities (e.g., symmetric binary relations[3]).

In the net-centric environment many attributes and relationships among interacting entities are manifested in systems and technologies. In general, symmetry of these attributes and relations provides simplification. This simplification can be realized in the ability to reuse behaviors, architectural and design patterns, applications, and even source code. For instance if both consumer and producer must authenticate each other, then there can be a common pattern for this which leads to common design and code. In messaging, if both producer and consumer must behave in the same way, then this provides a common design pattern. In an operational setting symmetry can be expected to simplify training and actual operation.

The lack of symmetry can be used as a tool for analysis of use cases, architecture and design to identify assumptions or errors. Consider the case of a distinguished user role, say system administrator. In most systems someone or something needs to perform actions not allowed to most system users. Such a role is not symmetric with respect to the actions that can be taken (and impact of these actions). From an implementation perspective this means that additional fields must be included in the data model, additional code must be written to account for the data model, and operation of the system must deal with this distinguished role. This example also highlights the impacts on security issues when there is symmetry breaking: Some entities will have a larger security risk than other entities.

As described above, the principles defined in this paper may not be independent of each other. Their exact relationships may change depending on the context of usage. The Symmetry principle can have relations with, or can impact, several other principles, including Dynamism, Globalism, Explicitness, and Relationship Management. Examples of the [possible] relationships among these principles can be found in examining the implications of a priori asymmetric relationships among net-centric entities.

[3] A binary relation R(x,y) is symmetric if for any elements a in the domain of R and b in the range of R, if R(a,b) then R(b,a).

A priori asymmetric relations assert that particular net-centric entities have fore-ordained (i.e., not explicitly discoverable on the network) (asymmetric) relationships to other net-centric entities. These relations are usually not explicit and have some implicit scope of applicability. As example, the client-server architecture model embeds an asymmetric relationship between nodes designated as client types and other nodes designated as server types. This distinction is usually based on pre-agreed nodal attributes used to distinguish one from the other. The ability for a client to act as a server or vice versa without software redesign or hardware change are usually severely constrained. Such a priori relationships can preclude negotiation of new relationships with these and other net-centric entities of equal footing, thus violating the principles of Explicitness - implicit relations, Dynamism – an inability to renegotiate relations or their entailments, and Globalism – the scope of the relationship is not sufficiently global.

Another class of relations among the principles of Symmetry and Primacy are those of the attributes assumed by a net-centric entity in its various roles. In many cases there is an asymmetry among these attributes due to an asymmetry in the roles of an entity. These asymmetries can manifest themselves during the lifecycle of a system or capability by an implicit embedding of these asymmetries in the representation of the entities thereby exacerbating the asymmetry and conflicting with Entity Primacy.

Examples:

- Consumer/Producer – The behaviors and applicable attributes for the consumer/producer pattern/paradigm should be symmetric: If the producer must authenticate the consumer, then the consumer must authenticate the producer.
- Hierarchical or Subordinate Relationships – Network interactions should be independent of organizational, social, or other superior/subordinate relations (i.e. no a priori hierarchical or subordinate relationships among network interactions). When present, they should be dynamically formed and negotiated.
- Peer-to-peer (as opposed to a hierarchical model) – Inherently symmetric
- System administrator - In this role some attributes of ‘system administrator’ are not symmetric w.r.t. to other entities that are part of this role.

3. Dynamism

Networked entities should support dynamic behaviors

Design attributes of networked entities (hence the entities themselves) should support dynamic behavior (as manifested by minimal a priori assumptions about the environment).

The net-centric environment has multiple time scales of dynamic behaviors. Some changes might occur from one transaction to the next (e.g., banking, trading), while others may represent the start of a new mission or operation. Yet others might represent the introduction of new systems with new capabilities into the operational environment.

Applying the dynamism principle requires judgment and pragmatic tradeoffs regarding the types of changes networked entities should be capable of handling, over the time scales of each type [of change], how these potential changes or their mechanisms are represented, and costs for providing this capability. This contrasts with more general assertions that a given design is “flexible”, “adaptable”, or able to handle a “wide range

of" some set of inputs or outputs. Each of these qualifiers usually have implicit times scales and usually are not quantifiable.

To be useful dynamism must be accompanied with a metric since any real system will have several types of dynamic capabilities. For example, a networked entity may have the ability to bind to different service instances or to different services based on changes in the environment such as a link failure, congestion, or a new service coming on line. Or it may need to respond to changes in its own mission objectives, such as changes in rules of engagement, regulations, or operational objective weighting factors.

Dynamism has design complexity costs, run-time discovery costs, and adaptation latency costs. On the other hand, lack of sufficient and appropriate dynamism in a net-centric environment can lead to a networked entity no longer being able to perform its intended function/purpose: It's useful lifespan is shortened.

Explicitness mitigates this risk of unforeseen end-of-life by enabling other networked entities to understand what flexibility/adaptability a given networked entity is capable of and thus allows reasoning about what the entity can be used for and when it can be used. If an entity has made explicit its characteristics and attributes, then whether to make of this entity or use other networked resources for the required functionality and functional scope that a given networked entity does not address becomes possible. In a less dramatic setting, explicitness in the design can help to minimize costs when changes need to be made.

As a corollary to the dynamism principle, networked entities should be explicit about the types and degrees of dynamism they support. This might be manifested by providing an explicit range of values for each of an entities characteristics or attributes (e.g., text encodings, media formats, temperature ranges, security protocols, etc.).

The Globalism principle also drives the need for dynamism. The global and net-centric environments present networked entities with many potential dimensions of flexibility and associated scales, such as coping with different languages, currencies, institution types or numbers of users. Dynamism allows networked entities to discover and bind to specific subsets of the global environment for a given network interaction rather than requiring the interacting entities to expose and deal with the full range of global attribute values that might otherwise be necessary. For example, if a networked entity is dealing with someone from the United States (US), it would not need to ask if currency should be specified in Euros or Yen, or what the country code is for telephone numbers that might be entered. Note, however, that the networked entity would still need to be able to handle these globalism attributes and values in its design or advertise which ones it doesn't support.

Application of dynamism also entails use of another net-centric principle, Relationship Management. In traditional system design, and even at the "enterprise" architecture level, the relationship of networked entities is established at design time and sometimes even at requirements elicitation/specification time. This static determination of relationships among networked entities works against the dynamism principle. It reduces or eliminates many of the potential operational benefits of a network-centric environment. By contrast, dynamism requires that relationships among networked entities be formed and broken as the environment and operational needs dictate, rather than on the relationships defined by system designers and sponsors envisioned when the networked entity was created. In the dynamic environment relationships must be explicitly managed as active components (i.e., additional objects in the system) of the networked entities rather than having those relationships simply be implicit and static (i.e., "hard-coded").

Additional entailments of the dynamism principle are illustrated in the examples.

Examples:

- **Monitoring Behavior** – Adaptability and flexibility require networked entities to implement some level of monitoring of behavior to sense when the environment or their mission has changed sufficiently to invoke the appropriate adaption behavior. Some simple examples of this dynamic behavior is a laptop letting you know that it has detected a new WiFi node in the environment, or the OS agent on that same laptop periodically checking to see if any new updates are posted to the Microsoft Windows download site.
- **Discovery Behavior** – Discovery is often interpreted to include monitoring behavior, but the focus here is on actively searching for other networked entities that have certain attributes and attribute values that are important to a given networked entity's objectives. It typically gets invoked after monitoring behavior has determined the need for adaptation or for finding some resource on the network. Discovery behavior can be exhibited by system sponsors and developers when they search for other systems and information sources that they can leverage to achieve their target system goals at system design time, but it is more often associated with the technical process of scanning service and data/metadata repositories on the network for services and information sources required to achieve a given network entity's objectives. However, software complexity and assurance concerns have kept the latter form of discovery from being widely employed as yet, and usually restrict such uses to a well-controlled internal enterprise environment. A simple example of discovery behavior is the Microsoft operating system agent on home computers looking for and downloading update patches that are pertinent to its local environment. Another example would be a travel site such as Travelocity™ checking an industry registry to see if a given hotel chain offers a service for making room reservations.
- **Negotiation Behavior** – Once monitoring behavior sets in motion discovery behavior, the results of discovery may require negotiation behavior, depending on the business model relationship between the networked entities that are attempting to interact for some purpose. The negotiation may be as simple as examining a standard service level agreement offered by a service provider and deciding whether to accept that standard agreement, or as complex as an automated sequence of proposed agreements and counter-proposals revising and trading off various performance parameters in a service level agreement framework used by the interacting networked entities. A simple example most of us experience today is negotiating WiFi access in a new hotel room or some public hot spot. Note that even "free" WiFI access often requires agreement to stipulated terms and conditions (even if these are unread). More sophisticated examples might involve trading off service levels for price, like a stock quotation service that provides less frequent and less current stock prices at a cheaper rate (i.e., less dynamism at lower cost).
- **Resource Allocation** - virtualization and load management. Once a relationship is negotiated, the network entity providing the service typically has to ensure that it has allocated the resources necessary to satisfy the relationship agreement. While this may be an internal process to the service provider and involve only management of resources that the provider controls directly, it may also involve additional external resource providers as well, depending on the nature of the resources involved and the associated business model. Thus one managed relationship can lead to additional managed relationships among networked entities. Examples include internal data center virtualization to support various enterprise application customers, internal cloud computing services, public cloud computing services, software-as-a-service providers, and the like.

4. Globalism-Universality

There should be no bounds on the scope of applicability

Attributes should require or entail no a priori operational nor institutional domain limitations (de facto limitations imposed by cost/risk/utility considerations) - The attributes used in a net-centric environment should be universally applicable.

As written this principle may appear to be extremely impractical. However, it should be considered in the context of system(s) development. In particular, if the converse of this principle is applied, then there would be strict bounds on the scope of applicability during development with the resulting system likely having little or no extensibility. Moreover, who or what would determine the 'scope of applicability'?

Extensibility asserts that reusable value exists by developing more grandly than for the immediate application at hand: The scope of applicability should be extensible beyond the original use intended by the designer.

Applying the Global-Universality principle to the tenet of extensibility would produce the following: Attributes should allow an a-posteriori designer to extend systems beyond the original operational and institutional domain limits (those that might have been imposed by pragmatic considerations like cost, risk, or utility).

Application of this principle impacts temporal, spatial, cultural domains, etc., and almost all other aspects or attributes of net-centricity and entities in the net-centric environment. A restatement of this principle might be architects, designers, developers, or service providers should start with the assumption that "all the world is their stage - or audience", and then apply the Pragmatism principle to constrain what will be built. However, Pragmatism limits how extensible an original designer should be (e.g., don't try to make a microprocessor scale to be a flying airplane).

Examples:

- Security markings affixed in, or to, a document. The context of many security markings assumes a particular domain of use or circulation. Globalism would enlarge the domain of circulation to include the world.
- Allowing security mechanisms to include foreign coalition partners
- Allowing databases or UAV sensor tasking mechanisms to work in either synchronous or asynchronous modes
- Permitting data visualization systems to display data not within the original display framework

5. Omnipresent/Ubiquitous Accessibility

Entities should have omnipresent or ubiquitous access to resources

Ubiquitous is defined as existing or being everywhere at the same time, or constantly encountered[4]. Resources refers to those of the net-centric environment – the network, networked services, information,

[4] http://www.merriam-webster.com/dictionary/ubiquitous

collaboration tools or other technical capabilities. So that ubiquitous access to resources means that entities in the net-centric environment have access, everywhere at any time, to networked resources.

Resource constraints in a net-centric environment may be caused for many reasons, These include, but are not limited to, bandwidth limitations (itself a resource), access due to security protocols, requirements to maintain shared state consistency at the application level to maximize synchronicity of shared data, or simply operational costs.

Today's networked technologies permit, subject to some physical, organizational, and resource constraints, ubiquitous access to data and information. With this access and using collaborative applications (non-human), workspaces are being enabled that expand the ability to share information and knowledge from local and proximate environments to dispersed and geographically distributed virtual environments. To facilitate such access requires negotiation and discovery to meet allocation needs and an ability to discover resources and their constraints.

While this notion does not include a level of 'protection', it will certainly impact security considerations and risk-reward negotiation. As the number of networked nodes increase, with concomitant increase in disparate access devices, the difficulties to protect secured resources on a 'need to know' basis are magnified and must be addressed. The degree to which this and the other aforementioned constraints place limitations onto ubiquitous access is relatable to the pragmatism principle.

Pragmatism greatly impacts this principle. However, this principle, when combined with the Pragmatism and Explicitness principles, would imply that if ubiquitous access to one or more resources must be constrained, than the constraints should be made explicit (i.e., there should be a clear and demonstrable reason for constraining access to the resource).

The rise of cloud computing may provide some validation of this principle. In as much as virtualized processes will have greater access to compute resources with which to instantiate additional instances.

6. Entity Primacy

Entities have identity distinct from contexts in which they participate[5]

Entities can be part of multiple contexts simultaneously. In each such context entities have an identity with which they are identified or individuated in that context. This identity allows interactions with the net-centric environment and the establishment of relationships with other entities in the same context. The identity used in such contexts needs to be unambiguously attributable to the participating entity and its contexts.

As example, the relation of an entity in an enterprise context is distinct from the concept of its existence as an entity operating in the larger net-centric environment (context). The entity's existence is separate from that of the enterprise context and has primacy in the net-centric environment. In simpler terms, this principle means that a networked entity, such as a truck, may be referenced by other entities on a network using a multiplicity of identifiers (e.g., vehicle identification, government issued license or registration, leasing company inventory number, etc.). However, none of the networked entities can assume that the particular identifier they use to individuate that truck from other trucks or networked entities in their context is shared

[5] This principle is a manifestation of realism. *Realism* asserts that reality and its constituents exist independently of our (linguistic, conceptual, theoretical, cultural) representations thereof.

universally among all networked entities across all networks (and the associated contexts). Other examples of operational or enterprise domain context identifiers include network addresses, MAC identifiers, email addresses, URI's, asset IDs, user names, avatars, "handles", or customer numbers.

System users or developers should not assume that interacting entities have an identity specific to a given institutional or system context, or that such a collective context is the only source of identity attributes for a networked entity. Conversely, system designs/implementations should support hiding entity identities or attributes used as identities in some contexts from others only if that is acceptable or required for some relationships, per the Relationship Management principle below.

The impact of this principle is that no single or collective context has a priori primacy for naming or assignment of identities to entities in a net-centric environment, hence the name of this principle. However, the Relationship Management principle supports elevating an entity's specific context identity to primacy in a given relationship context, subject to negotiation. For example, in some context a set of networked entities may agree to use a truck's Vehicle Identification Number (VIN), as the most appropriate identifier when referencing the truck in their shared operational context, such as state vehicle mileage fee assessments. Another example of a lack of collective context is if a U.S. citizen is identified using their US Social Security Number. Here the collective context being used (for a Social Security number) to identify a U.S. citizen is that citizen as a federal taxpayer, a US citizen, and participant in the Social Security system.

The key pragmatic consequence of applying the entity primacy principle is that systems should always associate and make explicit the specific enterprise/operational context with any identifiers used (in their context); Systems, this includes both operational systems and development environments, should be able to provide both the entity identifier and the context to other networked entities per whatever relationships they may have established with each other.

Typically, systems should provide a mapping between entity identities used in their "native" contexts and at least one other identity for those entities in some external context important to system users. Ideally, this external identity would be as close to "universal" as possible (while still recognizing that it may not be truly universal or inherent to the entity in question). So in the truck example above, a system managing a fleet of trucks for some enterprise might identify those trucks through some enterprise-specific/local truck number or asset ID, but also include a VIN number and a URI for each truck (assuming networked-enabled trucks). The URI is an interesting contrast with the VIN in this example. The VIN includes no clues as to who the owner of the truck might be, but a URL for such trucks is likely to have a component associated with a domain name tied to the owning or operating enterprise. Yet, if the enterprise in question were to truly net-enable their trucks (i.e., make them accessible from the public Internet and advertise their existence via DNS), the URL identifier can be useful as a near-universal network identifier for the trucks in question. The problem, of course, remains that the enterprise context is still part of that URI identifier and the identifier becomes useless for identifying the truck if the truck is sold/transferred to another enterprise unless some third party DNS registry service is used to identify all trucks. This might suggest a novel business model for a net-centric identity management service that partners with the current VIN naming scheme for trucks and other motor vehicles! Indeed, such a service might become necessary to enable some "smart highway", "intelligent transportation", or similar initiatives involving inter-vehicle and vehicle-to-highway networked communications.

Examples:

- A person may have a name, several user IDs, multiple account identifiers, multiple email addresses, etc., but has an identity distinct from all of these, but coupled to them through various institutional or

operational contexts (e.g., birth certificate, employee number, credit card provider, etc.). This is the underlying issue surrounding the many attempts to use various national ID cards and similar identifiers such as the Social Security Number (SSN) in the United States as a "universal" identifier for individual people. While there are many pragmatic reasons to use context-specific identifiers for entities referenced in the net-centric environment, they also entail many pragmatic pitfalls (e.g., not everyone or everything has an SSN, customer ID, email address, track ID, unique name, etc.)

- A network node may have a host/node name, a unique instance identifier (e.g., serial number), a MAC address, an IP address (fixed or DHCP-assigned), an asset ID, a network configuration item ID, etc.

7. Manage Relationships[6]

Relations should be explicitly represented and provide for negotiation, creation, change, and termination

There are many relationships that occur among networked entities. These relationships are both internal and external to an entity and among processes and their overall environments (the net-centric environment being one such environment). The relation among an entity and the network itself (e.g., communications) is one particularly important relationship in the net-centric environment.

Relationships also exist within and among social, physical, and operational environments – all sub-environments of the net-centric environment in the context of systems development. The net-centric environment may help facilitate or manage these relationships. Or the lack of management of these relationships may impede operational effectiveness.

Relationships can exist among diverse individuals and networked entities independent of enterprise affiliation. Any networked entity has the potential to behave as its own enterprise in the net-centric environment. Therefore, individual entities may increasingly require the capability to manage relationships they have with other entities in their net-centric environment. Symmetry in relationships is thus a desirable starting point. It allows all relationship options to be explored in support of dynamism, with provisions to negotiate and agree to specific asymmetric attributes to the relationship when both parties see benefit in doing so. A priori asymmetry in relationships inhibits dynamism and adaptability. This contrasts to most current system designs and design paradigms which take relationship options for granted, generally assuming they are static and usually asymmetric (e.g., the collective entity can establish or terminate a relationship with an individual entity, but not vice versa).

Application of the Explicitness principle would facilitate managing relationships by exposing all entity descriptive and behavioral attributes that could impact interactions and relationships with that entity on the network. Thus making an entity available for negotiation in its relationships that it is capable of establishing over the network. Note that managing relationships overtly is itself an application of the explicitness principle. Of course, there are pragmatic constraints on managing relationships explicitly, especially in a large scale network-centric environment. So it is important to identify those attributes of entities critical to

[6] It should be noted that this principle is in sharp contrast to most information-intensive systems developed to date where relationships are largely implicit (e.g., hard coded) in the design and implementation. The relationships of these systems are not actively represented nor managed by the systems themselves or elsewhere on the network.

achieving their respective objectives in potential relationships and focus management of relationships on those key explicit attributes.

Managing relationships is entailed by the Dynamism principle. In a dynamic environment specific relationships should not be assumed to be static or implicit in the design or operation of a network entity or its interfaces. Conversely, managing relationships needs to be dynamic, allowing new relationships to be formed, and existing relationships to be altered and terminated.

Other net-centric principles also impact relationship management, such as ubiquity and universality. Relationship management can be viewed as a constraint on universality and ubiquity, or as a way to constrain and represent their application in specific contexts in a dynamic, alterable fashion.

Relationship management can be used to capture decisions by network entities regarding the degree to which they will implement the other network centric principles in various operational contexts with other network entities and make those decisions visible and potentially open to negotiation by their own stakeholders and with other network entities. Interestingly, this could include a decision not to manage relationships with any other network entities at all for some set of interactions.

Another example of relationship management is security. It entails a relationship between two or more networked entities, as well as with the network itself. That is, there is no security without relationships, but relationships can exist without security – just not very trustworthy ones. The type of relationship among networked entities determine the degree of trust appropriate or necessary for the relationship to persist. The type of relationship also determines the degree of confidentiality appropriate for a given exchange between networked entities.

Relationships also occur in larger contexts that drive both trust characterization and trust levels, as well as enforcement and consequence management mechanisms that might be invoked/applicable. Explicitness suggests that these attributes of a relationship be reflected in representations of relationships used by network entities to manage their relationship with other entities. As opposed to assuming that all communicating entities are operating on some "Intranet" and subject to the same security policies and governance sanctions.

The net-centric environment imposes another need for relationship management – the environment in which an entity operates. Because networked entities cannot directly sense when another network entity receives a transmission, whether the received transmission is in fact what the sending entity intended to send or how the receiving entity reacted to or perceived the transmission. Relationship management therefore includes consideration of, and mechanisms to support, appropriate means of determining whether the interaction over the network is functioning as intended by the parties to the relationship. Pragmatism suggests that managing this aspect of relationships might be facilitated by third-party services, such as we see emerging under labels like "Enterprise Service Management".

Examples:

- **Trust** - Mutual Distrust (i.e. Dynamic Trust Establishment – Authentication, Authorization, Certification). In a net-centric environment any two or more network entities should begin their relationship on the premise of mutual (symmetry principle) distrust (dynamism, explicitness, universality, entity primacy). Based on what they discover as part of the dynamic negotiation process, using identities and services from contexts appropriate to the proposed relationship purpose(s), including possibly relationships already established with third party entities (like an identity credential brokerage and reputation management services) or trial use of offered services/protocols (dynamism/discovery, explicitness), the entities establish a relationship and monitor its progress until the purpose of the relationship is accomplished or known to have failed. If it has failed for reasons that

can be remedied by altering the relationship in some way, another round of dynamic behavior ensues until a determination is made by at least one of the network entities to terminate the relationship. A third party could also terminate the relationship based on yet another relationship that may have been established previously between it and at least one of the network entities involved (e.g, an external web site gets blocked by an enterprise firewall based on certain behavior patterns it detects when that site is interacting with network entities inside the enterprise firewall).

- **Multi-party Trust Relationships** – The transport mechanisms themselves should be part of the trust relationships that are negotiated among entities. One of the key challenges presented by the net-centric environment is that the network itself (in whole or in part) is usually provided by parties distinct from the entities that are attempting to utilize the network to establish and consummate a relationship. This means that nearly every relationship among network entities is a multi-party relationship, with the network provider being a key participant in virtually all interactions. So entities on the network need to establish a relationship with the network provider as well as with each other, especially if they have any significant security concerns regarding their relationships and want to rely on security services provided by the network provider to protect their relationship with each other. This observation is behind the drive to implement and offer so called "Network Access Protocols" as part of Intranet border router services. Of course, entities on the network can attempt to operate on completely untrusted networks, and may be required to do so (i.e., on the open Internet) to achieve their intended operational/business objectives. However, doing so will require the terminal entity nodes to provide their own security mechanisms to authenticate the distant network entities with whom they want to interact and provide their own data in motion confidentiality mechanisms. Basically this is what happens when someone accesses a web site on the Internet using SSL-based http access ("https"). Yet another third party role in trust relationship establishment is that of identity broker or reputation management services provider. Such services reduce the burden on individual network entities having to manage relationships with all the other entities on the network with whom they might want to interact. In effect, they "outsource" major aspects of relationship management to such third party broker services. SAML and PKI based authentication services are common examples of third party identity management/authentication relationships. E-Bay uses reputation management services to help potential buyers and sellers establish trust relationships with each other.

- **Collaboration** – Sharing intentions, goals, information, and actions. In other words, establishing a shared context with other entities on the network and exploring each others' perspectives and insights regarding that shared context, usually with the intent to achieve some mutually beneficial goal. Collaborative relationships are usually more focused and ephemeral than traditional relationships among networked entities, although they can also be persistent over a long period of time and have broad scope. Likewise, such relationships typically involve some number of human agents, often from diverse organizations and functional domains. But they can also involve automated agents such as avatars and intelligent agent software and traditional mission or enterprise application software. The latter are most often used as data sources for the collaborative engagement and as "sinks" for the results of a collaborative engagement (e.g., a "plan" or "analysis" for some action or topic area). Most collaborative relationships include specific entities on the network as participants and often specifically exclude other entities from accessing the shared context without some admission protocol and associated admission criteria. If the collaborative context involves changing values in "enterprise" or "system of record" data sources, this admission protocol allows the collaborators to change such values in "what if" dialogues over the network without immediately exposing those new values to other network entities. In other words, this collaborative context is an "alternate reality" that is shared with

only the collaborating participants until such time as they decide to expose their context representation outside their collaborative context. Note the relationship of this example to the entity primacy principle. The shared collaborative context creates a new identity for a collective (e.g., the collaborative "team") that didn't exist previously and may provide additional identities for the collaboration participants to represent their participation in the collaboration context (e.g., team leader, logistics specialist, etc.)

- **Enforcement/Sanctions/Consequence Management**. While collaborative relationships convey a certain element of voluntariness, they may still entail a certain degree of social sanction and consequence management. And depending on the contractual nature of particular relationships among entities on the network, managing relationships may require some level of legal sanction and enforcement appropriate to the organizational/political context in which those relationships are formed. A key issue for the net-centric environment is the degree to which current legal frameworks in the physical world can be applied and enforced in the net-centric environment. Many relationships formed in the net-centric environment can lead to serious safety, operational, and financial consequences. Non-repudiation and accountability are important elements of human social and business relationships, and analogs for these attributes need to be addressed in the net-centric environment for managing relationships. This is still a very immature aspect of net-centricity but there are a growing number of serious papers and studies on legal frameworks for "Cyber Warfare" and "Cyber Security" that are beginning to address this aspect of relationship management.
- **[HTTP] Session Management** – The relation among the 'server' and the 'client'. Much of the success of the World Wide Web and the Internet itself is due to minimizing the amount of relationship management required and thereby support dynamism and coping with the open world scaling challenges. While a TCP virtual circuit is in fact a managed relationship between two nodes on the network, albeit minimally so and not centrally managed, an http exchange supporting the World Wide Web is atomic and includes no managed persistent state from one exchange to the next. This is both a "feature" and a "problem" in pragmatic usage of the web. On the one hand, it allows dynamic behavior such as redirection, changes in IP addresses, handling server failures, and the like. On the other hand, the "stateless" nature of the http protocol means that access credentials (if any) need to be passed with each request and web sites have no memory of your state. This has created a number of patchwork relationship management "work-arounds" outside the http protocol itself, such as browser cookies and browser-based html standard form field entry memory, which allow the web site to "remember" you from prior visits and the browser to "remember" common data element values you may have entered, such as your email address or telephone numbers. A browser cookie therefore acts as a relationship management mechanism for the web site server, and the browser html form data element memory (including the site passwords than the user allows the browser to save) acts as a relationship management mechanism for the browser user as the "client" in the interaction with the web site as "server". One reason for the influx of PC-based and smartphone-based network "apps" is in fact their ability to bypass these somewhat awkward workaround relationship management mechanisms and implement more sophisticated and explicit mechanisms directly in the application, which, in turn, has access to a significant amount of user context information on the user's platform (which the browser is not designed to access).
- **Autonomic Behaviors** – Both Self-Management and Aggregate Management;
    - **Self-Management** - A relationship among a entity and its internals. This is "bottom up" or self-directed management of network entities for some "system" using the network to coordinate behavior among its system components to achieve some desired state or objective. In this case

there is a pre-established relationship among some set of entities that manage themselves, typically with some domain knowledge and awareness of their own characteristics/roles needed to accomplish their goals and their current state with respect to those attributes, as well as some knowledge base for what to do when these attribute values change in a specific way. An example might be a "smart" truck examining its internal subsystems and based on "prognostics" determining what parts might need replacing and routing itself to a maintenance facility for that purpose. It might also include maintaining an "avatar" or agent on the network to represent the truck's most recent state information when the truck is either disconnected from the network or "asleep" or otherwise not capable of responding to a request from another network entity.

- **Aggregate Management** – A relationship among the entities that comprise the aggregate. This is a more "top-down" management of entities on the network, possibly as a system of systems, or at least as a set of entities that need to co-exist in order to achieve higher level operational objective of which the individual entities are not necessarily aware. An example might be truck fleet management in which some trucks are directed to specific locations even though they don't have a specific need to be at those locations. This might be done in anticipation of load requests at those locations based on overall historical demand (or other trigger information) and the current spatial distribution of the truck fleet with respect to all the different load demand locations that they might be serving.

- **Monitoring the Net-Centric Environment** – A relationship among a net-centric entity and its net-centric environment. As mentioned previously under Dynamism, entities on the network need to monitor their environment for the occurrence of things that might impact their ability to meet their objectives. One of those "things" is the net-centric environment itself. Arguably the net-centric environment is nothing more (nor less), than the collection of all the other entities that both comprise the net-centric environment and form the network itself. However, from a pragmatic perspective, a number of entities on the network act as surrogates or agents for the network as a whole. This includes network nodes that provide services such as DHCP and DNS, or SNMP services, or service directory services. Other network services which represent an aggregation of networked entities in some fashion can also be viewed as acting as a portion of the net-centric environment when viewed from the perspective of individual entities on the network identity management services and network access control services fit this description. Entities on the network manage relationships with such services as surrogates for the net-centric environment itself. An application might monitor the net-centric environment to see if an email address or URL for a trusted data source has changed, or whether network throughput/latency to an often used service has deteriorated or improved since it was last used.

8. *Open World*[7,8]

There is incomplete knowledge of the operational environment

[7] This does not refer to Hermann Weyl's 1932 book *The Open World: Three Lectures on the Metaphysical Implications of Science*, New Haven: Yale University Press.

[8] The open world assumption is incompatible with *Logical determinism* or *Determinateness*, the notion that all propositions, whether about the past, present, or future, are either true or false.

In a net-centric environment entities have incomplete knowledge of their environment: a priori knowledge of all valid states, current or in the future, of all entities and their relationships in a net-centric environment should be assumed to be unknown or incomplete.

The Wikipedia definition of net-centricity, that the net-centric environment is continuously evolving, by itself suggests that complete knowledge of the net-centric environment is infeasible: The whole is ever changing and greater than the sum of its parts. For systems and application development the Open World principle has very tangible impacts. Much of systems and application development relies on the categorization of system or context boundaries, which in a net-centric environment is problematic. There may not be a fixed community of participants or users (e.g., Wikipedia). Hence no single enterprise or context definition. Indeed, the reason for a capability in a net-centric environment may itself be an ephemeral collaboration formed for a specific purpose (e.g., a military coalition or natural disaster response). Consequently, the enterprise to be served by a net-centric environment is a dependent variable; derived *ad hoc* not *a priori.* There is minimal or no *a priori* context. Additionally, the view by other entities in the net-centric environment of a system or other entity may be difficult or impossible for the system or entity to know (i.e., lack of a feedback loop or self-assessment mechanism).

The open world assumption impacts both development and operational aspects of the life cycle of net-centric capabilities and systems. This contrasts strongly with historical (and perhaps current) systems development and engineering models which attempt to enumerate and precisely define all boundaries, constraints and [system] states to allow absolute control of the operational system.

Current systems engineering analysis and design may involve mixes of open and closed world models, without explicitly identifying them as such. In the closed world, we state what is possible, with the assumption (often implicit) that anything else is invalid. In the net-centric environment, service and data providers with a complete axiomatization of a particular subject area would be viewed as closed world enclaves with guarantees of content consistency included in their service or data profiles.

When combined with the Explicitness principle, the Open World principle would require such a level of description as to be impractical for a net-centric environment. Applying the Pragmatism principle would suggest an approach to system or service descriptions that would focus on describing the service or system independent of potential users. That is no bias towards a particular class of users would influence the description.

The Open World principle impacts all other principles. Consider enterprise models and architectures which vary over time or network access - negotiation is required to validate contingent credentials (for network or individual service accesses, for example). That level of flexibility and adaptability calls for open world models of engineering and organizational frameworks.

The open world assumption, as described here, appears in discussions of logic, knowledge representation, databases, and the Semantic Web[3]. In these domains the open world assumption can be understood by considering the example of an airline reservation system and how the inclusion or presence of a (particular) reservation (or entry in the database) in interpreted. First, either a reservation is in the database or it's not. The open world assumption would interpret the absence of a reservation in the database (i.e., the lack of an entry) as an admission of a lack of knowledge about the reservation (i.e., whether it exists or not): No conclusions about the reservation can be drawn. In contrast, the closed world assumption would interpret the absence of the reservation as evidence that it does not exist. The closed world assumption is exemplified by finite state machines where all valid states are explicitly described, only known facts about the world are recorded, and unknowns are presumed false.

Examples:

- System entities and service provisioning should not be constrained by execution platform capacities. Net-centric system and service design should support dynamic load distribution across multiple platforms and service instances, constrained only by the pragmatism principle.
- Number of users – A net-centric system should be designed for extensibility
- Geographical scale
- Search results – The lack of results from a search does not mean there aren't any.

9. Pragmatism

The ability to improve operational effectiveness is paramount

The intent of the principles described herein is to effect or improve operational effectiveness and capabilities in a net-centric environment. As such there may be situations of contention among the various principles in application. In those cases pragmatism can be applied to distinguish or otherwise clarify which of the principles has precedence or whether the principles are applicable.

It should be kept in mind that, as in all things net-centric, these principles are in no way orthogonal or independent of each other. They are interrelated, the specific interrelations depend on their realization which in turn depends on the context of application. The principle of Pragmatism can be used to help identify or analyze which of the principles is more important and the impact of not applying one or more principles.

Application of the Pragmatism principle would be similar to that of a trade study or analysis of alternatives. Undertaking such a study or analysis will require a clear relation between the principle(s) in question and the impact on the operational effectiveness of their realization. As example, if the symmetry principle is applied to authentication, then both parties in the interaction must authenticate each other. This impacts system development, security, performance, and operation/administration of the system. So any analysis of trading-off this application of symmetry would need to understand the impact on each of these aspects.

Beware the opportunity for the misuse of the Pragmatism principle. It may be presented as an excuse for not meeting net-centricity requirements or mandates. Or prematurely constrain the possible options. As in all things net-centric, there is no single formula or amount of net-centricity that can be preordained or uniformly applied.

Examples:

- Strategic Intent
- Operational Intent
- Costs/Risks
- Objective Alignment
- Technological Limitations – A current example is the use of semantic technologies to aid in realizing the Explicitness principle. Though current semantic technologies can be used to make systems and their information more explicit, pragmatically there are not enough experts to meet the needs. Thus introducing a risk.
- Service Description – In SOA services must provide a description of themselves so that potential consumers can determine whether to use that service. The current standards for service description

rely on natural language or a priori human intervention in service selection. This can viewed as a pragmatic response to the lack of sufficiently mature semantic technologies that could allow consistent machine interpretable descriptions.

- Security – For most systems organizational policy dictates conflict directly with the Omnipresent/Ubiquitous principle. Open and ubiquitous access must pragmatically be balanced with the policy dictates of an organization.
- System Architecture & Design – Pragmatically technology does not currently exist that would support the full implications of meeting the Open World principle (e.g., self-adaptive or evolutionary behaviours). However, as noted in the discussion of the Open World principle, too often at the out set of system development unnecessary constraints or assumptions (often implicit) are used.

4 Net-Centric Environment Emergence

Net-Centric environments and operations are complex, dynamic and contain multiple scales and interactions. With any sufficiently complex system **emergent behaviors** or **emergent properties** can appear when a number of simple entities (agents) operating in that environment form more complex relations and behaviors as a collective. If emergence happens over disparate scales, then the reason is usually a causal relation across the different scales. In other words there is often a form of top-down feedback in systems with emergent properties.

The processes from which emergent properties result may occur in either the observed or observing system, and can commonly be identified by their patterns of accumulating change, most generally called 'growth'. Why emergent behaviors occur include: intricate causal relations across different scales and feedback, known as interconnectivity.

The emergent property itself may be either very predictable or unpredictable and unprecedented, and represent a new level of the system's evolution. The complex behavior or properties are not a property of any single such entity, nor can they easily be predicted or deduced from behavior in the lower-level entities: they are irreducible[9]. The introduction of additional capabilities or properties (representing attributes or characteristics) may engender unanticipated consequences. These consequences may involve more than the entities that manifest the properties, that is the engendered consequences or emergent behaviours may involve other entities of the environment.

The net-centric environment differs from purely natural or physical environments where the ideas of emergence and self-organizing systems began. The major difference being that of the inclusion of machine mediated interactions and the control or constraints upon the non-human entities in this environment – They are not subject to the range of possible interactions or reactions that an entity in a natural environment is subject to. In the net-centric environment there are at least two distinct classes of entities. The physical infrastructure, routers, switches, servers (this class of entities could also include the power sources, communications hardware and paths, etc.) and the humans that (ostensibly) control and use this infrastructure for their purposes. It could be argued that a third class of entities is constituted by the information that resides in and traverses the net-centric environment.

Examples:

- Self-Organization

[9] From http://en.wikipedia.org/wiki/Emergence#World_Wide_Web_.26_Internet.

- Small-world network
- Use of networked resources in unanticipated ways
- Financial trading systems – The rise of global communications has changed the way commodities and equities are traded.
- Facebook, MySpace, Twitter – The use, prevalence and impact of social networking
- Routers - Small scale interactions produce large scale effects: Routers normally only look to nearby routers for a view of the network environment. But together these small views help the Internet work, providing the World Wide Web its basis of operation.

5 Relation to NCOIC Products

Various NCOIC products make use of the notion of 'principle'. In many cases this may be construed as misnomer with respect to the explanation and definition provided in Section 1.3. To aid the reader, some comparisons of 'principles' from other NCOIC products, to those described herein, is provided.

5.1 NCOIC Interoperability Framework (NIF)

The NIF was developed to provide an organizational construct and a repository for the enabling guidance being developed by the NCOIC. The information it provides is intended to complement reference architectures that are being developed via various government entities (departments, ministries, and services - defense and civil) and the systems engineering processes and tools already resident in engineering firms.

The NIF OverArching Framework (OAF) provides a set of assets that drive the content of the Specialized Frameworks by means of inheritance, traceability and relationships. As with all frameworks governed by the NIF, the OAF contains Concepts, Process, Principles, and Products. It contains a set of fundamental OverArching Principles that should be applied to any system or enterprise to achieve net-centricity.

Many network-centric principles implement a combination of a small set of general-purpose design tenets. These tenets help to understand the basis for achieving net-centricity and are listed below in Table 1.

Table 1 NIF vs Core Principles

NIF Principle	Underlying Core Principles
Enforce modularity - Modularity assists with replacement and upgrade, and often reduces coupling complexity (e.g: interfaces).	**'Modularity'** exhibits the principles of **Dynamism** and **Relationship Management**. If a modular or decoupling approach is taken, then this will allow dynamic behaviors and allow for more flexible management of the dependence relation among the components of the system.
Strictly enforce abstraction - Abstracting capabilities from the underlying technology is a fundamental IT principle that provides more robust interfaces.	**'Strictly enforce abstraction'** is an instance of **Relationship Management**, the relation among the capabilities and the implementation or technology.
Be explicit - Many net-centric problems come from implicit assumptions that are not documented or shared between networked parties.	**'Be Explicit**' is exactly the **Explicitness** principle, not to make, or to endeavor to minimize implicit assumptions or other information.
Apply a decentralized work scheme - Within a network of systems, decentralized schemas support keeping security domains independent and foster improved availability.	**'Apply a decentralized work scheme'** exhibits the principles of **Scaling** and **Relationship Management**. The relationship is that which exists among the entities that constitute the system. By decentralizing, a larger scale can be accommodated.

Preserve Autonomy - The orientation towards net-centricity emphasizes the need for autonomous behaviors to ensure mission continuity.	**Autonomous behavior requires both Entity Primacy and Relationship Management. An autonomous system needs to be able to individuate itself from its environment while also managing itself and the environments in which it may operate.**

5.2 Service Orientation Principles

The NCO Interoperability Framework (NIF) identifies the service approach as a key interoperability concept. The Net-Centric Services Framework is contained in the NIF overarching framework, consistent with the its structure requirements: Concepts, Principles, Patterns and Processes[10]. And complements NIF at a practical level. As stated in the Net-Centric Service Framework, version 1.0.24,

Principles are the overall requirements, goals, tenets or best practices that should be applied to foster Net-centricity. Principles are used as a basis to assess conformance of products and overall architectures.[11]

Thus the use of the term 'principles' differs from that adhered to in this document as described in Section 1.3.

Table 2 Service Orientation vs Core Principles

Service Orientation Principle	Underlying Core Principles
Service Description - The service description should, as far as possible, be explicitly expressed in a formal language, the service description language.	**'Service Description'** manifests the **Explicitness** principle, not to make, or to endeavor to minimize implicit assumptions or other information.
Service Interface - The service interface implements the contract between the consumer and provider. The service interface offers the capabilities of the service.	**Service Interfaces** are an example of **Relationship Management**
Service Contract - Service contracts, often referred to as Service Level Agreements (SLAs), define almost all of the primary parts of an SOA.	**Service Contracts** are an examples of **Relationship Management and Explicitness**.
Service Abstraction - Abstraction of underlying logic (also referred to as service interface level abstraction). It is this 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.	**'Service Abstraction'** is an instance of **Relationship Management**, the relation among the capabilities and the implementation or technology.
Loose Coupling – Loose Coupling is a condition wherein *"a service acquires knowledge of another service while still remaining independent of that service"*.	'**Loose Coupling**' represents both the **Entity Primacy** and **Relationship Management** principles
Service Interoperability - 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."[12]	**Service Interoperability'** is an instance of **Relationship Management**, the relation among the capabilities and organizations providing them.

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

[11] Excerpted from NIF version 2.

[12] 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

Service Composition - 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.	**Service Composition** represents the **Dynamism** principle**.**
Service Orchestration - 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.	**Service Orchestration** represents the **Dynamism** principle
Service Choreography - A SOA is an information system architecture for linking resources on demand. 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). Service objects can be combined by Choreography, enabling dynamic reconfiguration which, in turn, decouples Providers and Consumers of the System's Resources.	**Service Choreography** represents the **Dynamism** principle**.**
Discovery - Discoverability on the level of architecture is meant to refer to the technology architecture's ability to provide a mechanism of discovery. 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.	**Discovery** represents the **Explicitness** and **Entity Primacy** principles
SOA Governance - 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 a) Decision rights for the development, deployment and management of new services, b) Monitoring and reporting processes for capturing and communicating governance results.	**SOA Governance** represents both the **Relationship Management** and **Explicitness** principles
Location Transparency - 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.*[13]	**Location Transparency** represents both the **Relationship Management** and **Explicitness** principles
Security - 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.	**Security** reflects the **Pragmatism** principle

6 Application Guidance

As described in Section 1.3, one intent of these principles is to provide a basis for distinguishing attributes. One method to accomplish this is the process of derivation wherein an attribute is shown to be derivable or

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

an entailment of one or more of the core principles. The following section provides one example of how a derivation make be executed.

6.1 Derivation of Composability

The concept of composability has been proposed as another 'core' Net Centric Principle. The following will show that it can be derived[14] from principles of Dynamism, Relationship Management, and Explicitness in the context of Net Centricity.

Among the various system, software, service, or pattern attributes associated with net-centricity and net-centric service oriented architectures (SOA), is that of composability. Composability is itself a concept that has no one single meaning or definition that is universally shared, but generally includes a number of concepts manifested in net-centric systems and services. To minimize confusion, for this note the following definition will be adopted.

> **Composability** - the ability to connect or assemble two or more components into a more complex system or "ensemble" to meet a set of a prior purposes or goals that, by themselves, the individual components cannot fulfill.

Note that implicit in the understanding of 'composability' is that each of the components that may be involved in a composition provide a capability that meets a purpose.

In colloquial usage there is another implicit expectation that these connections or assemblages can be accomplished with no or minimal additional effort (e.g. software development) and due to this minimal effort, can be accomplished in short time frames. The key point being that 'composability' implicitly requires no development expertise and little, if any, system domain expertise. Ideally, the components should prevent incorrect composition much like differently shaped cable connectors are used to prevent the wrong signal types from being applied to specific inputs on electronic devices.

In a net centric context, composability takes on the additional notion or constraint of accomplishing the binding over a network connection, usually (but not necessarily) though some advertised/discoverable service oriented interface. Another aspect of net centric composability is that it does not assume knowledge of the implementation technology or execution environment of the component with which the composition binding is effected, aside from possible performance limitations such execution environments might entail.

6.1.1 Derivation

The description of composability (above), necessarily entails or requires

a. a pre-existing environment that allows or supports dynamic behaviors. For otherwise, components would be static and isolated (i.e. their inter-component time bindings would be infinite). Hence, as defined, composability is an instantiation of the principle of Dynamism.

b. at least 2 components (that may be composed) and some type of relation among them that permits composition consistent with the expected purpose of composition. In order for the composition to occur the compositor must be able to evaluate the candidate components for fitness. This action requires knowledge of the components and is itself an instance of relationship management. Thus representing both the principles of Explicitness and Relationship Management.

[14] Use is made of modal necessity and the possible world semantics of Kripke. A proposition P is necessary for Q if every possible world in which Q holds, P holds, P □ Q

6.1.2 Consequences and Entailments of Net Centric Composability

As the description of composability intimates, the realization or instantiation of net-centric principles of Dynamism, Relationship Management, and Explicitness drive systems towards allowing or supporting composability. This capability is a way to achieve a greater degree of net-centricity. The net-centric principles of Globalization and Scalability also an important enabler of composability. Ubiquitous network access allows net-centric compositions of components/services to persist even if component/service execution environments are mobile with respect to each other or have to operate in remote locations.

Use (i.e. instantiation) of the Dynamism principle forces systems to monitor their environment and respond rapidly to changes in user requirements or in the environment. For instance, by making network enabled connections to different and/or new components/services that happen to be available over the network, rather than by developing new code or modify existing code – which would generally take considerably longer than composition (days/weeks/months instead of seconds or minutes).

Composability is also a manifestation of the dynamism principle as an attribute or feature of an architecture or component/service set that facilitates greater dynamism – quick binding to alternate/variant services/components.

The Dynamism, Relationship Management, and Explicitness principles requires or entails that the relationship between components/services be explicitly represented and actively managed. In other words, relationships should be dynamically formed, monitored for changes in the relationship state or in the causes or purposes of the relationship, and overtly dissolved when conditions warrant such dissolution.

A net-centric service composition is precisely such a dynamically established relationship among networked entities, subject to changes in the environment and to the entities that are party to the relationship.

The explicitness principle supports/enables net-centric composability by making the comonent/service attributes important to a correct composition relationship to be themselves visible over the network. This enables the composition service to dynamically discover the attributes and value ranges that enable a correct composition relationship to form among a given set of components/services. Otherwise the composition service would have to be designed with information about all possible composition relationships among all the constituent/candidate components/services embedded in the code of the composition service or stored in a file store local to the composition service execution environment. That would limit the network accessibility of the composition service, and reduce its dynamism and scalability.

The globalism principle pushes composable component/service developers to support as wide a range of interface argument/output value types and argument value ranges as possible. This makes component services more correctly usable in a wider range of possible compositions. It also makes the job of the composition service simpler because it will have to do less value type and range checking to determine if a given composition will function correctly. Put another way, a component/service that is built to be more reflective of the globalism principle is more likely to function correctly when made part of a novel composition than a similar component or service that is built using the assumption it will be employed by a single enterprise, in a single operational domain, or in a specific country.

The scalability principle also enables net-centric composability by forcing developers of composable components/services to consider the possibility of an open-ended number of composition relationships any given component/service might participate in. In net-centric composability, a single instance of a component/service may participate in multiple composition relationships over the network. Although this can also happen in non-net-centric composability (e.g., via re-entrant code libraries), the scalability requirement is tempered by the execution environment/platform computing capacity. In net-centric composability, on the

other hand, the number of composition relationships is not inherently limited by individual execution platform capacity to host multiple service/component requestors.

Composability is a system/software architecture attribute that can be derived from the two primary net-centric principles of Dynamism and Relationship Management, and is further enabled by applying the net-centric principles of Explicitness, Globalism, and Scalability. The other net-centric principles play a tangential role as well. For example, the entity primacy principle enables dynamism in composability by separating service/component instance identity from composition membership (collective identity) and thereby enables dynamic reallocation of component/service instances to a composition because of server load saturation or network/component failures. The symmetry (entity autonomy or anti-hierarchy) principle encourages component/service developers to design them such that they their relationships with each other can be determined/established by the composition service rather than be implicitly built into their individual service interfaces (as is often the case in typical component libraries). Of course, this latter symmetry may be inherently constrained by the nature of the function that the component/service performs.

7 Summary

Name	Principle
Explicitness	An entity should make all information about itself explicit
Symmetry/Reciprocal Behaviors	Relationships among networked entities should exhibit symmetric characteristics and behaviors
Dynamism	Networked entities should support dynamic behaviors
Globalism-Universality	There should be no bounds on the scope of applicability
Omnipresent/Ubiquitous Accessibility	Entities should have omnipresent or ubiquitous access to resources
Entity Primacy	Entities have identity distinct from contexts in which they participate
Manage Relationships	Relations should be explicitly represented and provide for negotiation, creation, change, and termination
Open World	There is incomplete knowledge of the operational environment
Pragmatism	The ability to improve operational effectiveness is paramount