This appeared as a chapter in Eerlijk zullen we alles delen: verkenningen naar interoperabiliteit, Sander Zwienink and Pieter Wisse (eds.), Forum Standaardisatie, The Netherlands, 2008, pp. 346-62.

Interoperability as a semantic cross-cutting concern

Jeff Rothenberg

Introduction

This chapter focuses on two crucial and intertwined aspects of interoperability: the fact that it is a 'cross-cutting' concern and its reliance on semantics (i.e., the meaning of information and of the processes that are performed on it). Although these two aspects of interoperability may at first seem only tangentially related to each other, I will argue that they are in fact inseparable and can be seen as two faces of a single issue, which I refer to as a *semantic cross-cutting concern*. I will also argue that interoperability is unique among cross-cutting concerns in that it is not only 'internally' cross-cutting in the sense that it permeates the design of a given system, but it is also (by definition) a *cross-system* concern, which adds another dimension to all other cross-cutting concerns. This cross-system dimension is fundamentally semantic: the biggest challenge of making systems interoperable is to pervasively align the semantics of what they do and how they do it.

The nature of interoperability

A broad definition

Although there are many definitions of interoperability, the following encompasses most of them, so long as the term 'system' is interpreted broadly, as discussed below:

> Interoperability is the ability of distinct systems to share semantically compatible information and to process and manage that information in semantically compatible ways, to enable their users to perform desired tasks.

Here, systems can be computer (ICT) systems, institutional, organizational, or procedural systems, or any combination thereof. This broad interpretation enables the definition of interoperability to apply to groups, organizations, commercial enterprises, governments, military units, etc., as well as ICT systems. Throughout this chapter, I will use the term 'system to refer to any and all such combinations, distinguishing ICT systems explicitly only when the context is ambiguous.

The emphasis on *distinct* systems is implicit in the concept of interoperability. A system is by definition a coherent entity whose components are integrated with each other. As discussed below, integration is a stronger criterion than interoperability and therefore subsumes it. By implication then, interoperability applies only to distinct, disparate systems — which are not already integrated with each other.

We do not normally speak of the components of a system interoperating with each other, since their integration already implies that they interact in a more or less seamless manner.

Interoperability involves making multiple, distinct systems work together; but this is never an end in itself. The goal of interoperability is to serve some human purpose by sharing information or accomplishing some task — which presumably cannot be performed in its entirety by any subset of the systems in question. Interoperability therefore implies far more than simply getting ICT systems to communicate with each other. It also requires that such ICT systems — and the organizations that use them — employ compatible interpretations of the information they exchange, that they perform meaningfully compatible operations on this information, and that they enforce compatible policies and perform compatible procedures when handling this information.

Essential aspects of interoperability

Under this broad definition, two closely interrelated aspects of interoperability stand out: the fact that it is a cross-cutting concern and its reliance on semantics. These aspects of interoperability apply to systems in the above broad sense, as well as to ICT systems in particular.

From a perspective external to that of any single system, interoperability is a cross-cutting concern because it cuts across distinct systems. It is therefore typically outside the scope of any single system development effort or organization. In addition, systems are often designed and implemented before

there is a recognized need for them to interoperate, making interoperability all the more outside their original scope. However, interoperability is also an internal concern that must pervade each individual system if that system is to interoperate with other systems. Interoperability involves all aspects of a system, including its reliance on standards, its choice of information processing procedures and algorithms, the validity criteria surrounding its representation and processing of information, its interfaces to other systems and to its users, and its so-called non-functional or quality attributes, such as reliability, availability, security, privacy, and information assurance.[1]

In order to produce meaningful and valid results, interoperating systems must have compatible semantics. As a start, they must interpret the information they exchange and process in compatible ways. This implies that they must have compatible mechanisms for exchanging, representing, modifying and updating semantic descriptions of this information. Even more fundamentally, they must process the information that they exchange in ways that are meaningful to the other systems with which they interoperate. To the extent that the meaning and form of this information may be dynamic and may change over time, these systems must be able to modify their processing of this information — and possibly its representation — dynamically. Moreover, interoperating systems must employ compatible semantics in the interpretation and alignment of their policies with regard to validating, protecting, preserving, and disseminating the information and results that they exchange with each other.

The cross-cutting nature of interoperability and its fundamental reliance on semantics are closely interrelated. From a cross-cutting perspective, interoperability involves semantics at all levels, from the choice of compatible terminology, encodings of data, and communication protocols to the semantic alignment of processing algorithms and institutional policies. Conversely, semantic interoperability can be achieved only if semantics is considered a cross-cutting, pervasive concern, both within each system and across the collection of interoperating systems. In the following sections, each of these two aspects of interoperability is analyzed separately, but their interrelation will be emphasized throughout the discussion.

Issues surrounding interoperability

Potential benefits of interoperability

Interoperability offers a number of interrelated potential benefits; for example, it:

— increases flexibility, by allowing systems to be ‘mixed and matched;’
— facilitates the creation of new capabilities, by composing new functions out of existing ones;
— increases cost-effectiveness, by allowing the reuse of existing systems and capabilities;
— creates virtually integrated systems that are easier to use.

The ability to mix and match systems allows their users to perform unanticipated tasks that require new combinations of existing functions. This can be done on the fly to perform non-recurring tasks, or it can be done in a more principled manner, combining existing functions to create new ones that can be used repeatedly to perform recurring tasks. Similarly, interoperability can reduce the cost of creating new capabilities by allowing existing systems to be reused in multiple ways for multiple purposes (see the subsection titled Reusability, below).

One potential advantage of interoperability that is not often emphasized is that interoperating systems can create the illusion of being integrated with each other, even when they are not. Such virtual integration can make it easier for users to interact with an interoperable collection of systems by presenting a uniform user interface, uniform semantics, and uniform procedures and policies for use.

It is important to note, however, that these benefits may accrue to different parties. For example, developers may be able to sell the same reusable system to multiple customers for use in multiple markets, thereby increasing their return on investment, without necessarily providing any advantage to their customers. Conversely, customers may be able to purchase a reusable system only once rather

[1] Although these aspects apply most obviously to ICT systems, they occur in organizational systems as well.

than purchasing multiple variants of that system, thereby saving money, at the expense of vendor revenue. Similarly, the increased potential to mix and match interoperable systems and to create new capabilities by combining these systems in new ways may be utilized by developers to produce robust product lines with minimal development effort and cost: by combining interoperable components in different ways, developers can produce multiple products, each of which can be sold separately. Conversely, these same benefits may be passed along to customers by selling the interoperable components individually and allowing users to combine them to create their own applications. Finally, the virtual integration of interoperable systems can simplify the job of developers and maintainers of new capabilities as well as providing end users with a more consistent and coherent experience.

Costs and disadvantages of interoperability

Despite its potential advantages, interoperability does have some costs and potential disadvantages. In particular, it may compromise privacy and security, and it inevitably adds technical complexity to system design.

By linking together information that has previously been kept in separate silos, interoperability may make it easier to 'connect the dots' about users, customers, citizens, and others. This may be an advantage for commercial enterprises, governments, and law enforcement, but it poses an increased threat to privacy.

Similarly, interoperability may compromise security, making each system only as secure as its least secure interoperating partner. Isolated systems often operate within secure enclaves that ensure the security of their information; connecting such systems to each other creates an umbrella enclave that may be less secure than any of its constituents.

Finally, interoperability adds technical complexity to system design, since it imposes new requirements on a system. As discussed below, planned interoperability is more cost-effective than ad hoc interoperability, since retrofitting interoperability is both more difficult (and therefore expensive) and less effective than building it in from the start. In either case, however, the cost of a system design and development effort is likely to increase if the system is required to be interoperable. Moreover, this added cost is incurred by the system developer and/or customer, who may or may not reap the benefits of interoperability themselves. This 'sow versus harvest' problem arises when a number of independent development efforts are tasked with making their systems interoperable with each other: no one development effort may see much benefit for itself in making its system interoperable, whereas the added cost of doing so may be significant.

In many cases, the benefits of interoperability outweigh its costs and disadvantages, but these factors must be acknowledged and taken into account, along with any potential sow versus harvest problems, which may require inventive mechanisms for sharing costs and benefits across the organizations responsible for individual systems, thereby motivating them to make their products interoperable.

Systems of Systems

A collection of distinct, interoperating systems constitutes what is sometimes called a System of Systems (SoS). This term is often used to refer to any collection of systems that function together as a whole.[2] Any system is likely to be composed of components or subsystems, but unlike the subsystems of a system, each component system in a System of Systems is typically a standalone system in its own right, which performs some useful function independently of the other systems in the SoS. Furthermore, many SoSs are defined after the fact. That is, they are not designed and built as SoSs; rather they are defined as such only after their component systems have been independently designed and implemented. The degree to which a collection of systems deserves to be called an SoS is the degree to which the systems in that collection interoperate effectively with each other. This is analogous to the criterion for something being legitimately referred to as a system, which might be defined as the degree to which its components are integrated with each other. I use the term System of Systems in this chapter to refer to collections of interoperating systems; however, it must be kept in mind that such collections are often loosely defined and dynamically constituted and configured, often without having been designed as Systems of Systems initially.

---

[2] For a discussion of System of Systems issues, see Annette, J. Krygiel, *Behind the Wizard's Curtain*, Cooperative Research Program (CCRP), 1998.

Integration vs. interoperability

As noted above, true integration is best achieved by designing subsystems together, so that they function as part of an encompassing system, or in some cases by designing collections of systems together, to function as part of an encompassing System of Systems. In general, systems that are integrated with each other are more than just interoperable; we tend to use the term 'interoperability' to refer to the linking of systems that are *not* fully integrated. In this sense then, interoperability is always merely an approximation to the ideal of seamless integration. Nevertheless, when systems have been designed separately, interoperability may be the best we can achieve, short of redesigning a new integrated system ab initio.

Furthermore, even if a collection of systems is designed as an integrated whole, they will eventually be replaced or supplemented by new systems, with which they may no longer be integrated, since the new systems will be designed separately. Moreover, independently developed systems are unlikely to be integrated. All of this implies that interoperability is and must be a more open-ended approach than integration, since it enables new, unrelated, and unforeseen systems and capabilities to work together without redesigning the overall System of Systems that includes them. Interoperability is therefore more flexible and potentially more scalable than integration, even though it rarely achieves the same degree of seamlessness.

Because integration occurs within a system, whereas interoperability occurs between systems, interoperability tends to fall outside the scope and responsibility of any single system project or program, as discussed below (see the subsection titled Bringing interoperability within scope). However, interoperability is a more realistic goal than integration where independently developed systems are concerned.

Designing interoperability into a system versus retrofitting it

Interoperability can either be designed into a system from its inception or it can be retrofitted to the system after it is built. Designing interoperability into a system is generally more effective, since interoperability is a cross-cutting concern that must be implemented pervasively throughout a system in order to be effective. It is also generally far easier and cheaper to design interoperability in from the start than to add it after the fact.

Yet the need for interoperability is not always apparent when a system is designed and implemented. The desire to combine existing systems in new, unforeseen ways often demands that they be made interoperable after the fact. Although retrofitting interoperability in this way may often be worthwhile, it can be quite difficult. The cross-cutting nature of interoperability requires it to permeate many aspects of a system, such as its external and user interfaces, its use of standards and communication protocols, its internal processing, and the policies and procedures that it employs in its treatment of data and its provision of access to users. Examining, evaluating, and possibly modifying these aspects of an existing system in order to make it interoperable can require significant effort.

Similarly, because interoperability relies heavily and pervasively on semantics (as discussed below in the section titled Semantics), retrofitting a system to be interoperable can require major redesign of the system's representations and algorithms, as well as realignment of its policies. The cross-cutting nature of interoperability and its fundamental reliance on semantics make it challenging enough to implement when designing a system with interoperability in mind; when attempting to retrofit interoperability to existing systems, these factors can become serious obstacles to success.

The importance of system architecture for interoperability

Ultimately, the interoperability of a collection of systems depends on the architecture of each individual system, as well as that of the collection. The architecture of a system is a description of its overall design and structure, the underlying technologies and mechanisms used to implement it, its major elements and components, how these fit together structurally and logically, and how they interact with each other to produce the system's behavior. Inevitably, enhancing the interoperability (or integration) of a collection of systems or components must involve some kind of architecture, whether it is explicit, implicit, or ad hoc.

When pre-existing or independently designed systems are required to interoperate in the absence of a common architecture, linkages between individual pairs or subsets of these systems are often designed

and implemented in an ad hoc manner, producing what is effectively an ad hoc architecture. Such ad hoc architectures are unlikely to be coherent or scalable; greater interoperability among a wider set of systems can usually be achieved by designing and adopting an explicit architectural framework of some sort and ensuring that each system conforms to this framework. However, it may be impractical to design and enforce such a framework for pre-existing systems that are combined to create an ad hoc System of Systems.

In general, the architecture of a System of Systems (whether that architecture is designed in advance or is ad hoc) may enhance or reduce the ability of the component systems of the SoS to interoperate with each other. That is, some architectures may facilitate interoperability, while others may inhibit it. For example, Service Oriented Architecture (SOA), discussed below, provides a framework that can help its component services interoperate with each other, assuming they are designed in accordance with the SOA paradigm. Ideally, interoperability issues should be empowered to affect architectural decisions and should be used to help determine the choice of an architectural paradigm for any System of Systems that is intentionally designed as such. Yet this ideal cannot be applied in all cases, since many Systems of Systems are not intentionally designed but instead consist of ad hoc combinations of pre-existing, independently designed, standalone systems that were never intended to work together.

SOA as related to interoperability

One particularly important architectural alternative in the context of interoperability is the Service Oriented Architecture (SOA). This recent paradigm relies on an open-ended set of discoverable, autonomous, and interoperable services, each of which performs a distinct function, and which can be combined and invoked to perform a wide range of business processes. Services are typically remotely accessible over a network, such as the Internet. Despite the fact that it is not yet fully proven, SOA is rapidly becoming the architecture of choice across commercial, educational, government, and military domains.

The services provided by SOA are relatively coarse-grained in the sense that they provide fairly significant packages of capabilities, as opposed to fine-grained individual functions.[3] However, services are generally smaller than the standalone component systems of a System of Systems. Pre-existing components that were not designed with SOA in mind can be wrapped or packaged to function as services, but doing so can make it difficult to achieve the degree of interoperability required to make SOA work.[4] SOA is more likely to succeed when it utilizes new capabilities that are designed and implemented as services, rather than attempting to repackage and lash together pre-existing components.

The key to SOA is an infrastructure that supports the creation, publishing, discovery, remote invocation and orchestration of interoperable services, provides standard protocols for services to use when interacting with each other, describes the semantics of services, and stimulates the creation of an appropriate collection of such services to provide appropriate functionality. This infrastructure (derived from earlier approaches, such as CORBA,[5] DCOM,[6] and J2EE[7]) attempts to address many of the issues of interoperability, including semantics. If SOA can be made to work, it should therefore enhance interoperability; however, this is a circular argument, because in order for SOA to work, it must create pervasively and semantically interoperable infrastructure and services. Furthermore, it is unclear how well SOA generalizes to Systems of Systems, whose components are full-fledged

---

[3] If this were not the case, the communications overhead of using SOA would be prohibitive. For an overview of SOA, see *Service Oriented Architecture: Concepts, Technology, and Design* (Prentice Hall, 2005) by Th. Erl and the OASIS *Reference Model for Service Oriented Architecture V 1.0* (official Committee Specification approved Aug 2, 2006, available at http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf).

[4] See "DOA with SOA," Alex E. Bell, *CACM*, October 2008, Vol. 51, No. 10, p. 27, ff.

[5] See CORBA's *Common Object Request Broker Architecture: Core Specification, Version 3.03* (2004, available at http://www.omg.org/docs/formal/04-03-12.pdf).

[6] Microsoft's Distributed Component Object Model (see http://msdn.microsoft.com/en-us/library/cc201989.aspx) was a major competitor of CORBA; it has since been superseded by Microsoft .NET, described at http://msdn.microsoft.com/en-us/netframework/default.aspx).

[7] See *JAVA EE at a Glance* (Sun Microsystems, Inc., available at http://java.sun.com/javaee).

standalone systems in their own right that are unlikely to have been designed as SOA services and are therefore unlikely to interoperate with each other without substantial redesign and retrofit.

Reusability

Reusability is one of the key motivations for interoperability. The ability to compose a System of Systems out of reusable component systems has the potential to make the resulting SoS cheaper, more reliable, easier and faster to develop, more consistent, easier to maintain and support, and more flexible and scalable than it would be if it were designed as a monolithic whole.[8]

Reusability and interoperability converge in strategies such as Service Oriented Architecture, where the behavior and interfaces of components are formally defined, thereby allowing them to both interoperate with each other and be reused as components of multiple systems. A reusable component must provide a function that is of general value to other components. In some cases, this can be achieved by offering a single, well-defined capability that is of general use, such as looking up names in a directory or registry. In other cases, a component may need to provide a more general capability that has many variants and alternatives, such as a general database query function or statistical analysis. In many cases, each of the various other components that utilize a given reusable component will need it to perform a somewhat different function, or to perform its function in a somewhat different manner. This need for customization requires a reusable component to offer multiple options for its use, which complicates its interfaces, thereby making it more difficult to use. The need to balance the generality of reusable components against their ability to be customized (often referred to as their customizability) is a key challenge, and no general strategy is known that can ensure that components will achieve this balance. Each component must attempt to find its own 'sweet spot' that combines generality with customizability in ways that depend on its function and its relationship to other components.

Moreover, as noted above, the services in SOA are typically designed (or at least packaged) as reusable, interoperable components rather than as standalone systems in their own right. These services are envisioned as components of a larger system, rather than existing systems that are to be combined into a System of Systems. It is therefore an open question whether the reusability of SOA services translates into the reusability of interoperable systems in a System of Systems.

Semantics

Dimensions of semantics in interoperating systems

In order to work together, organizations and their systems must be able not only to exchange information but also to ensure that the information they exchange is meaningful to all parties. That is, their concepts, terms, processes, and policies must be aligned with each other so as to be compatible. Semantic interoperability involves at least:

— data definitions, units of expression, and the computational basis and assumptions that are used to derive data;
— terminology and controlled vocabularies;
— computational methods and assumptions used to produce results;
— conceptual and functional models;
— business process models;

[8] There are advantages to implementing monolithic systems, since they tend to be better integrated and more internally consistent and coherent. In addition, they have the potential (though not always achieved) to be more responsive and better tailored to their users' needs. However, it is frequently necessary to modify a system after it has been built, whether to improve its performance, adapt it to new (or newly revealed) user needs, take advantage of new technology or new business opportunities, etc. Modifying a monolithic system without redesigning it tends to reduce its integration and internal coherence, whereas it may be easier to modify a System of Systems by adding more component systems to it or modifying one of its existing component systems. The result is that after repeated modification cycles, an interoperable System of Systems may be no less integrated, coherent, and tailored to its users' needs than an equivalent monolithic system.

— key policies, such as access, authentication, authorization, security, transparency, accountability, privacy, etc.

If terminology and data are not aligned in this way, exchanged information may be misinterpreted. Similarly, if processes or procedures are incompatible, it may be difficult to combine them to create new, integrated processes, as required when implementing new composite capabilities. If fundamental algorithms and information processing are not semantically compatible, results may be meaningless or (perhaps worse) misleading. Finally, if policies such as privacy and accessibility are misaligned, interactions may be infeasible or may conflict with some parties' policies. Semantics therefore provides the foundation for meaningful interaction among systems and organizations. Yet achieving semantic interoperability is a substantial undertaking that is likely to require considerable effort.

Defining and aligning semantics

In some cases, distinct organizations or systems have good reasons to maintain distinct semantics. In such cases, interoperability requires that these distinctions be made explicit and that, to the extent possible, mechanisms should be provided to enable systems to automatically map one set of such semantics to another.[9] In addition, organizations may maintain internal semantics that may be opaque to users of their systems, whether these 'users' are other organizations, businesses, or human end-users. In order to enable their users to understand and interact with them effectively, such organizations must either modify their internal semantics to match the external semantics of their anticipated users or map their internal semantics into those external semantics when required. Ignoring semantics is a fatal flaw for interoperability, since doing so may give the illusion that systems can work together while in fact they may produce meaningless — or worse, invalid and misleading — results.

However, developing explicit semantics for even a single organization or system is a challenging task that requires analyzing and making explicit core assumptions and knowledge that are typically unstated and implicit among a group of practitioners, who typically share an unstated common background, purpose, organizational culture, and mindset. These practitioners are rarely versed in the formal methods required to represent explicit semantics, yet they are the experts who must provide the knowledge and understanding that are needed to create such representations.

Communities of Interest and semantic contexts

It is currently fashionable to divide the semantics problem into subsets, each of which is to be dealt with by a community of interest (COI) that is responsible for that subset.[10] If the areas of concern of these COIs had no significant overlap, then each COI would be free to develop its own disjoint semantics independently, and there would need to be little, if any, coordination of these semantic efforts across COIs. However, COIs are rarely disjoint, since they cut across each other. For example, commercial, government, and personal sectors are often populated by the same people. Similarly, many such seemingly distinct communities are 'cross-cut' by common disciplines, such as economics, technology, communications, and the law. Practitioners in a given sector (such as healthcare) may include those trained in — and concerned with — any or all such disciplines, creating an overlapping network of semantic contexts. Conversely, partitioning the semantics problem by COI assumes that all members of each COI share a common organizational culture, whereas in practice, distinct organizations within a given COI often have widely divergent cultures and mindsets.

The semantics problem is further complicated by the fact that meanings vary across different contexts

---

9 For example, see *The Object Primer: Agile Model-Driven Development with UML 2.0* (New York: Cambridge University Press, 2004) by Scott W. Ambler and *UML's Sequence Diagram* by Donald Bell (2004, available at http://www.ibm.com/developerworks/rational/library/3101.html).

10 The term 'community of interest' was popularized by the U.S. Department of Defense but its use has become widespread. See http://metadata.dod.mil/mdr/ns/ces/techguide/community_of_interest_coi.html and *Data Interoperability Community of Interest Handbook*, by Michael M. Gorman (Whitemarsh Information Systems Corporation, 2006, ISBN 0978996801).

even within a given COI.[11] Moreover, meanings change over time, sometimes permanently and sometimes temporarily, in response to current situations and concerns. For example, a term such as 'offshore' may have a nautical, engineering, or commercial connotation, depending on whether the current focus is on maritime issues, oil exploration, or the use of foreign business partners and subsidiaries. Approaches for dealing with dynamic semantics of this kind have their origins in early work on Truth Maintenance Systems in Artificial Intelligence.[12]

Representing semantics

Considerable research has occurred in the area of semantic representation. Notable current efforts include the semantic web developments surrounding XML-based tools such as RDF (Resource Description Framework)[13] and OWL (Web Ontology Language),[14] the ongoing Cyc project at Cycorp, Inc.,[15] the work on Topic Maps in The Netherlands, Norway, and elsewhere,[16] and Metapatterns.[17] Nevertheless, these semantic representation tools do not in themselves provide a mechanism for developing semantics to support interoperability. Practical processes need to be developed to enable the creation of suitable interoperable semantics to support a given System of Systems. In order to facilitate this, each SoS should include a semantic interoperability development process model to facilitate the development of appropriate semantics in the appropriate contexts. These process models should enable organizations to develop explicit representations of their semantics and align these representations with those of other organizations with which they need to interoperate.

Interoperability as a cross-cutting concern

Dimensions of cross-cutting concerns

A cross-cutting concern is one that cuts across sub-system boundaries and other considerations.[18] A cross-cutting concern must be pervasive in order to be effective; it cannot be isolated or encapsulated in a single module or subsystem. Furthermore, it is very difficult to add a cross-cutting concern as an afterthought without significantly increasing its cost and reducing its effectiveness.
In addition to interoperability, a number of other cross-cutting concerns also affect systems, including:

— system security and information assurance (IA);
— privacy and confidentiality of data;
— Non-functional qualities, such as usability, reusability, reliability, scalability, flexibility, extensibility, accessibility.[19]

---

[11] See Doug Lenat and R. V. Guha, *Building Large Knowledge Based Systems* (Addison-Wesley, 1990); R.V. Guha, *Contexts: A Formalization and Some Applications* (Stanford University Computer Science Department, PhD thesis, Stanford, 1991); J. McCarthy, Notes on Formalizing Context, (in: *Proc. IJCAI 93*, Morgan-Kaufmann, 1993); D. Lenat, *The Importance of Context Space* (Cycorp, Austin, Texas, October 28, 1998); and P.E. Wisse, *Metapattern: Context and Time in Information Models* (Addison-Wesley, 2001, ISBN 0201704579).

[12] See J. Doyle, A truth maintenance system (in: *Artificial. Intelligence*, vol. 12, pp. 231-272, 1979).

[13] RDF has been developed as part of the W3C Semantic Web, see http://www.w3.org/RDF.

[14] OWL has been developed as part of the W3C Semantic Web, see http://www.w3.org/TR/owl-guide.

[15] Lenat and Guha, *op. cit.*

[16] ISO/IEC 13250, *Topic Maps: Information Technology, Document Description and Processing Languages* (Michel Biezunski, Martin Bryan and Steven R. Newcomb, ed., 3 Dec 1999, available at http://www.y12.doe.gov/sgml/sc34/document/0129.pdf); see also http://www.empolis.com/downloads/empolis_TopicMaps_Whitepaper20030206.pdf

[17] Wisse, *op. cit*.

[18] Cross-cutting concerns are one of the fundamental issues addressed by Aspect-Oriented Programming (AOP), see Aspect-Oriented Programming by Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier and John Irwin (in: *Proceedings European Conference on Object-Oriented Programming (ECOOP),* Finland, Springer-Verlag LNCS 1241. June 1997) and *Introduction to Aspect-Oriented Programming* (O'Reilly, 2004) by Graham O'Regan.

[19] These are referred to as non-functional or quality attributes, since they do not directly affect the functional performance of a system (because most of the terms (in English) end in '-ility' they are sometimes

Like interoperability, each of these concerns must be designed into a system and must be pervasive if it is to be effective. Every such cross-cutting concern is internally cross-cutting, since it affects many aspects of a system. However, interoperability is unique among cross-cutting concerns in that, in addition to being internally cross-cutting, it also cuts across system boundaries. Just like any other cross-cutting concern, interoperability must permeate many aspects of a given system in order to be effective, but to a far greater degree than any other cross-cutting concern, interoperability *inherently involves cross-system issues*. If a given system is required to interoperate with other systems, all of its other cross-cutting concerns may become cross-system issues, as well. In this sense, interoperability *extends* all other cross-cutting concerns to be cross-system concerns.

Interoperability must therefore be charged with coordinating all of the other cross-cutting concerns among interoperating systems. For example, if two systems are to interoperate, they must not only implement privacy as an internally cross-cutting concern, but they must also have compatible privacy policies. Seen in this light, interoperability becomes a uniquely *cross cross-cutting-concern concern*.

Because of its extended cross-cutting nature, interoperability must be dealt with consistently and coherently by a wide range of organizations and parties. For example, designers and providers of ICT systems used in different agencies at different levels of government and representing different sectors need to adopt consistent standards, protocols, data representations, and semantics to ensure that eGovernment systems can interoperate. This requires coordination across institutional and sector boundaries, as well as across different organizational cultures.[20]

The cross-cutting nature of interoperability has profound consequences. In particular, it implies that interoperability cannot be implemented piecemeal and cannot easily be added to a system after it has been built. Furthermore, because interoperability is the cross-cutting concern that must take responsibility for ensuring that all other cross-cutting concerns are respected in a cross-system context, it has a unique and crucial coordination role.

Designing-in versus retrofitting cross-cutting concerns

Ideally, interoperability should be designed into a system from its inception. Precisely because it is a cross-cutting concern that must permeate many aspects of a system in order to be effective, interoperability is much harder to address after a system has been designed and implemented. As previously discussed, the need for interoperability affects an ICT system's data representations, its processing algorithms, its need to explicitly represent the semantics of its data and its processing, its policies for validating, protecting, preserving, and disseminating the information that it stores, generates, and shares with other systems, and its interfaces to its users and to any external systems. Changing these aspects of an existing system to be more interoperable may require intensive redesign and reimplementation, which are far more difficult, expensive, and error-prone than addressing such aspects initially.[21] Moreover, simply creating a wrapper around an existing non-interoperable system

---

collectively called 'ilities'). See B. W. Boehm et al., *Characteristics of Software Quality* (Amsterdam: North-Holland, 1978); V. R. Basili and J. D. Musa, The future engineering of software: A management perspective (in: *IEEE Computer*, vol. 24, pp. 90-96, Sept.1991); S. E. Keller et al., Specifying software quality requirements with metrics (in: *System and Software Requirements Engineering*, R. H. Thayer and M. Dorfman, eds., IEEE Computer Society Press, tutorial, 1990, pp. 145-163); and T. P. Bowen et al., *Specification of software quality attributes* (Report RADC-TR-85-37, Rome Air Development Center, Griffiss Air Force Base, NY, Feb. 1985).

[20] See J. Rothenberg, Maarten Botterman and Constantijn van Oranje-Nassau, *Towards a Dutch Interoperability Framework: Recommendations to the Forum Standaardisatie* (TR-552-FS, 2008, available at http://www.forumstandaardisatie.nl/fileadmin/OVOS/RAND_rapport_def.pdf.

[21] Retrofitting in general requires significant resources: maintenance is responsible for 60-80% of a typical product's total software lifecycle cost and more than 50% of programmer effort. See *Measurements to Manage Software Maintenance* by George E. Stark (The MITRE Corporation, Colorado Springs, Colorado, 1997); B.P. Lientz and E. B. Swanson, Characteristics of Application Software Maintenance (in: *Communications of Association for Computing Machinery*, June 1978, pp. 466-471); S.S. Yau and T. J. Tsai, A Survey of Software Design Techniques (in: *IEEE Transactions on Software Engineering*, June 1986, pp. 713-721); V. Gibson and J. Senn, System Structure and Software Maintenance Performance (in: *Communications of the Association for Computing Machinery*, March 1989, pp. 347-358); and B. Boehm, *Software Engineering Economics* (Prentice-Hall, New York, 1981).

may be insufficient: although this may enable the system to communicate with other systems, it may remain far from meaningfully interoperable with them, especially if the internal semantics of its data representations and processing are not examined to ensure that they are compatible with those of the other systems in question.
Unfortunately, it is not always apparent when designing a system that it may eventually be called on to interoperate with other systems, some or all of which may not yet exist. In such cases, realistic estimates of the effort and cost required to retrofit a pre-existing system to make it interoperable should be weighed against the alternative of redesigning and reimplementing the system in question.

Bringing interoperability within scope
Because interoperability involves two or more systems, it tends to fall outside the scope and responsibility of any single project or program. It therefore often 'falls through the cracks' between projects. Except in endeavors that are intentionally designing a System of Systems, no one is likely to have responsibility for thinking about interoperability. Even in SoS design, where interoperability may be recognized as important, it tends to be relegated to an overall integration task, which is often not supported very well by individual projects. Unless resources are allocated by each project (or added to their budgets and timelines) to enable them to focus on interoperability, they are likely to assign it very low priority, if not to ignore it altogether. Conversely, if a system is not designed as part of an intentional SoS effort, it is unrealistic to expect it to devote significant resources to making itself interoperable just in case it may need to interact with other, unforeseen systems in the future.
The semantic aspect of interoperability similarly falls outside the scope of most individual projects. An individual, standalone system can often get by without explicitly formalizing its semantics, since it need not share its internal meanings with other systems. When a system functions as part of a System of Systems, however, it must ensure that it understands the data, processing, and policies of other systems and that they understand its data, processing, and policies. This creates a need to explicitly represent semantics and share these representations with other systems. As with all other *cross-system* aspects of interoperability, this need is generated by the System of Systems context and so is rarely acknowledged or resourced within individual system design efforts, especially for pre-existing systems that were designed without any requirement to participate in a System of Systems. This lack of responsibility for semantics is particularly troublesome because achieving a consensus on semantic issues can require considerable effort by a broad range of overlapping communities of interest, as discussed above; this may be as difficult, expensive, and time consuming as building the systems themselves.
As noted above, it is hard enough to implement interoperability when designing a system with interoperability in mind. We can distinguish three cases that involve developing new systems (whose interoperability can therefore be designed-in) plus two cases that require retrofitting interoperability to existing systems.

— When new systems are developed as part of the intentional design of a new System of Systems, some overarching authority should be empowered and resourced to perform SoS integration across all of the systems in the SoS, and each individual system development effort should be required and resourced to adhere to the interoperability guidelines mandated or recommended by this authority.
— When new systems are designed to work with an existing intentionally designed SoS, they should similarly be required and resourced to adhere to whatever interoperability guidelines exist for that SoS.
— When new systems are developed as part of an existing SoS that was not intentionally designed as such, that SoS may not have any interoperability guidelines. In this case, new guidelines may have to be developed retroactively by whatever authority controls the SoS, if such an authority exists, or — if there is no such authority — by collaboration with the developers, maintainers, and users of the other systems in the SoS.
— When existing systems that were not intended to be part of any SoS are subsequently required to interoperate with other systems in an intentionally designed SoS, the existing systems must be retrofitted to conform to the interoperability guidelines of the SoS.

— In the fifth case, an existing system that was not intended to be part of any SoS may be required to interoperate with other systems in what is essentially an ad hoc SoS. Here, not only was the SoS in question never designed to be an SoS, but no new system development effort is underway that can incorporate interoperability as a new concern. Instead, all of the existing systems in the ad hoc SoS must be made to interoperate, with minimal redesign and modification. This is a very demanding and risky undertaking, that requires retrofitting pervasive, cross-cutting, semantic interoperability to multiple independent, existing systems that were not designed to interoperate with each other or to be part of an SoS. This characterizes many real-world situations, in which existing systems are asked to interoperate in ways that were never envisioned by their designers. Without some degree of formal SoS integration, coupled with significant, resourced redesign and reimplementation of existing systems, such endeavors seem unlikely to succeed.

To summarize, the first two of the above cases are demanding but straightforward, since they involve the design and implementation of a new system that is to be made interoperable with an intentionally designed SoS. The last three cases, however, are increasingly problematic, since they involve various combinations of retrofitting existing systems and achieving interoperability within ad hoc, non-intentionally designed Systems of Systems. Many real-world endeavors exemplify the last — and most challenging — of these cases.

Conclusions

Interoperability is the ability of distinct systems to share semantically compatible information and to process and manage that information in semantically compatible ways, to enable their users to perform desired tasks. I have argued that interoperability is a unique concern that is not only internally cross-cutting but is also inherently a *cross-system* issue and therefore adds a *cross-system* dimension to all other cross-cutting concerns. This means that interoperability is a *cross cross-cutting-concern concern* that must coordinate all of the other cross-cutting concerns among interoperating systems.
Furthermore, interoperability is fundamentally semantic: getting systems to talk to each other is much easier than ensuring that what they say is mutually meaningful. Ensuring semantic interoperability among systems requires that their interpretations and manipulations of data are compatible and that their procedures and policies for handling their data and their results are aligned. Ignoring semantics is a fatal flaw for interoperability, since doing so may give the illusion that systems can work together while in fact they may produce meaningless or, worse, invalid and misleading results.
These two aspects of interoperability are inextricably intertwined. Semantic interoperability must be cross-cutting, and the cross-cutting nature of interoperability is fundamentally semantic. The preceding discussion demonstrates the operational equivalence of these dual aspects of interoperability by showing that both of them must be pervasive if they are to be effective and that both are therefore much easier to build into a new system design than to retrofit. In addition, both of these aspects are more easily achieved when intentionally designing a new System of Systems, as opposed to configuring existing, independently designed systems as an ad hoc SoS. Finally, semantics as well as all other cross-system aspects of interoperability are System of Systems concerns that tend to fall outside the scope of individual system design and maintenance projects.
There appear to be five distinct situations in which interoperability may be incorporated into a system. In increasing order of difficulty, these are:

— developing a new system as part of the intentional design of a new SoS;
— developing a new system to work with an existing intentionally designed SoS;
— developing a new system to work with an existing SoS that was not intentionally designed as an SoS;
— retrofitting an existing system to interoperate with a new or existing intentionally designed SoS;
— retrofitting an ad hoc collection of systems–which were not designed as an intentional SoS–to interoperate with each other.

Above, the subsection titled Bringing interoperability within scope has suggested some possible strategies for each of these situations, but the implementation of interoperability remains challenging, especially in the last three cases.

In summary, interoperability is often used as a surrogate for integration when independently designed, standalone systems are connected to each other to provide new capability in a System of Systems. In many cases, such Systems of Systems are not intentionally designed as such but are rather constituted and configured as needed or even dynamically. Ad hoc Systems of Systems of this kind can create useful new capabilities, but they place significant stress on their component systems, which were in general not designed to work with each other. Even when Systems of Systems are intentionally designed as such, it is challenging to make their component systems interoperate; when ad hoc Systems of Systems are created out of existing standalone systems, this challenge is considerably harder. Nevertheless, interoperability is easier to achieve in such cases than seamless integration, while offering some of the same advantages and potentially greater flexibility and scalability. In all cases, in order to make systems work together meaningfully, it is essential to recognize that interoperability is a semantic cross-cutting concern, which must be implemented pervasively *both* within *and* across systems if it is to be effective.

Jeff Rothenberg, a Senior Computer Scientist at RAND, Santa Monica, has worked on numerous projects concerning interoperability and semantics. He was technical lead on a recent RAND-Europe project that developed recommendations for a Dutch Interoperability Framework. He has also developed an Adaptive Decisionmaking paradigm that helps revise decisions using semantic representations of dependencies among related decisions, assumptions, and constraints.