Evolving Enterprise Architecture

Richard Martin[1], Sandeep Purao[2], and Edward Robertson[3]

[1] Tinwisle Corporation
Bloomington Indiana
richardm@tinwisle.com
[2] College of Information Sciences and Technology
Penn State University
spurao@ist.psu.edu
[3] School of Informatics and Computing
Indiana University
edrbtsn@indiana.edu

Abstract. Architecture artifacts evolve over the course of an enterprise life cycle through a series of transformations. A significant side effect of these transformations may be implicit change in abstraction levels. Prior research as well as practice shows that these meta-ness concerns exist throughout a life cycle but tend to be predominant in early phases. Lack of alignment during architecture creation and lack of understanding about how the transforms provide utility to different stakeholders is caused by the failure to distinguish meta- from non-meta- aspects of the target architecture. This paper examines 'meta-ness' and other characteristics of transformations, so that we are able to align transformations to construct artifacts which properly represent the system of interest and provide value for stakeholders. Also, we may evaluate whether the meta-architecture adequately supports the transformations and their use.

1 Introduction

Enterprise architecture, its representation, and its uses all evolve; but they evolve in different phases and for different reasons. Difficulty and sometimes even disaster ensue when these evolutions are misaligned. Controlling these evolutions requires understanding the motivations and the mechanisms for evolution.

Architecture is a metaphor from the realm of office towers and bridges, intended to capture the use-oriented, as opposed to construction-oriented, aspects of the design of those structures. An enterprise architecture is a means to understand an enterprise by organizing and presenting artifacts that conceptualize and describe the enterprise. Architectures, particularly enterprise architectures, will be considered more in Sect. 3. In the following, the word "architecture" used alone always indicates enterprise architecture.

Researchers have argued and practitioners have demonstrated that the value of architecture is not in its representation but rather in its use. Yet a poorly represented architecture is of little use. Hence our concern with *architectural descriptions*, the vehicle for these representations.

Several studies have contributed to our understanding of the evolution of architecture as it gets expressed and used by different stakeholders.[13] Similar reports from the community of practice have described hierarchies of detail and of "meta-ness" in architectures and in their representations.[4] GERAM, Annex A of ISO 15704:2000,[9] identifies two aspects of evolution for enterprise architecture. The first concerns progress along a particular life cycle and the second concerns the re-occurrence of architecture artifacts as part of the chain of life cycles that take an organization from business concepts to production and service of products. At the INCOSE Architecture Workshop in 2008, participants stated the necessity to "distinguish the life cycle through which architecting happens from the life cycle of the architected system," respectively the *creation* and the *application* life cycles.

Transformations provide the mechanisms through which architectures – more precisely, the artifacts representing architectures – evolve. The meta-architecture drives the transformations though which architecture artifacts evolve often with increasing detail (Sect. 5 discusses how detail and "meta" are two distinct hierarchies). Much software engineering research shows that such transformations are a part of prescriptive methodologies for software design.[2] In this domain, each successive generation of the various kinds of models (programs, data flows, objects) has led to facilitating the transformations required to move from early phases to later, implementation-dependent ones. The introduction of generalization to software engineering has also resulted in the introduction of greater abstraction levels. These concerns, which may be described as one aspect of meta-concerns,[10] suggest that the meta-ness of a model is likely to be an important consideration that drives the evolution of artifacts.

As an architecture goes through its life cycle from specification to use, it has utility for multiple stakeholders who interpret and use it for their purposes. What constitutes utility for groups of stakeholders tends to be different, and as a result, the primacy they accord to different perspectives, viewpoints, slices or aspects of the underlying set of models also differ. Although the core architecture survives, it may retain a relatively weak existence dominated by the strong perspectives from various stakeholder groups. An architecture description is a *boundary object* [12] that serves as an interface between different communities of practice. Because architecture means different things to different people, that boundary object is the focal point for the perception of architectural utility.

As artifacts transition from conceptions to operational reality, their qualities of structure, behavior, and purpose transition from architectural in nature through design oriented to qualities of operational mechanisms. This transition is characterized in [3] with respect to intension and locality. Architecture is intentional and non-local, design is characterized as intentional but local, and implementation is extensional and local. This use of locality of impact offers a way to distinguish transitions between architecture and design from transitions

[4] Since we are encouraging precision in dealing with meta-ness issues, it behooves us to observe that the architecture representation discussion is meta-meta with respect to an actual system.

within the architecture as it evolves. Unfortunately, transitions often are not distinguished with respect to locality of impact and are melded into one intentional grouping where meta-level distinctions related to architecture and architecting are lost.

Whatever mechanism or process is used to create system architecture artifacts, that mechanism or process is meta- with respect to the artifact created. There is a more general architecture, either explicit or implicit, associated with the created architecture artifact. This is the enterprise architecture, within which human execution produces new or revised architecture artifacts for a system-of-interest.

With these considerations as motivation and foundation, the paper examines the facets of "meta-ness" in Sect. 2. Section 3 more carefully defines architectures and architectural descriptions. Section 4 examines time and the rules that constrain the chronology of activities. Next, Sect. 5 distinguishes the spectrum of meta-levels from other dimensions that are significant for architectures. The stage is now set to examine, in Sect. 6, the transformations which interact within "meta" and the other dimensions to create and evolve the architecture artifacts utilized by various stakeholders across the life cycle.

2 "Meta"

Properties related to "meta" are fundamental and pervasive to careful consideration of architecting. These properties are sometimes ignored because they are obscure, but the consequence of such ignorance is truly obscured analysis and deficient systems. Hence our first topic must be meta-ness.

The dictionary gives several definitions of the prefix "meta-", the relevant one being "more comprehensive, transcending – used with the name of a discipline to designate a new but related discipline designed to deal critically with the original one (meta-mathematics)."[15] A more syntactic definition is "A prefix meaning one level of description higher".[5] In the original Greek, "meta-" meant "behind" or "after", as in the anatomical term "metacarpal". The current use arose because Metaphysics came after Physics in Aristotle's work.

"Meta-" is relative (an order relationship), "meta-ness" is the related property. This is validated by the occurrence of phrases such as "meta-meta-data".[5] When used with a single term, "meta-" is reflective, with a sense of "aboutness"; "meta-X" means "X about X". Meta-data is data about data, meta-language is language about language, meta-media is journalists writing about journalists,[1] and meta-models are models about models.[6] Due largely to Hofstadter's popular

[5] Because "meta-" is directional, it would be convenient to have a term that inverts this direction. Since "meta-" is Greek for "after", we might consider the Greek for "before", which is "pro-" (akin to the Latin "pre"). Unfortunately "pro" already carries way too many connotations.

[6] This even fits with metaphysics (consistent with current use of "metaphysics" but not with the origin of the term), if we recall that physics was once also called natural philosophy, and thus metaphysics is philosophy about philosophy.

Gödel, Esher, Bach: the Eternal Golden Braid[4], the reflexivity of "meta-" has entered the vernacular meaning explicit self-reference and thus is associated with paradoxes from Epimenidies to Russell and the paradox-based proofs of Gödel and Turing.

"Meta-" often has a sense of abstraction. This is related to the fact that reflection often involves abstraction. Thus a meta-model is an abstraction of a family of models.

A final, general use of "meta", which certainly harks back to the notion that metaphysics is obscure and incomprehensible, applies "meta" to anything that is complicated and difficult to understand. Given the importance of the other meanings of "meta", we certainly hope that it avoids this last meaning in our context.

Where an organization provides architecting as its business product, it has an internal architecture describing how it delivers that product which is meta to the architecture of the client's system of interest. At the Penn State EA workshop in 2009, participants identified two corporate enterprise architecture communities that participate in the application life cycle, one related to company structure and governance and one related to services for customers. These communities work together to evolve their practice.

3 Architectures and Architectural Descriptions

The term "architecture" dates back to the early Greeks[7] and its expression was first codified by the Roman Marcus Vitruvius, whose classic work *De Architectura* was the standard reference for 1500 years, wherein he states that a structure must have *firmitas*, *utilitas*, and *venustas* - strength, utility, and beauty. More recently, the advent of automated information processing has given rise to a more expansive notion of architecture and the architecting by which it is created. While there is still considerable debate about the exact nature of architecture in the domain of automation-based enterprises, two International Standards provide definitions. The revision draft of ISO 42010, re-titled as *Systems and software engineering – Architecture Description*,[8] defines architecture (of a system) as the "fundamental concepts or properties of a system in its environment embodied in its elements, their relationships, and in [sic] the principles of its design and evolution." The revision draft for ISO 15704, re-titled as *Automation systems and integration - Framework for enterprise architectures and models*,[7] amplifies the definition of architecture as the "[enterprise] conceptualization of the form, function, and fitness-for-purpose of a system in its environment, as embodied in the elements of the system, the relationships between those elements, the relationship of the system to its environment and the principles guiding the design and evolution of the system."

[7] "Architect" derives from the Greek prefix "archi-", for principle, applied to "tektōn", craftsman or builder.

Each of these definitions of "architecture" seeks to concisely capture meaning for terms that now are used widely in many domains of practice, conveying a slightly different meaning in each domain.

Whatever the definition of "architecture", it is the artifacts that represent an architecture, the architectural descriptions, that are truly important. Indeed, the approach taken by the drafts of ISO 42010 and ISO 15704 focus on the architectural descriptions rather than the architecture itself. We consider the manner in which those artifacts evolve and provide utility to those who use them.

The various stakeholder concerns are addressed by an architecture description that must be understandable both to those stakeholders, so that validation is possible, and to other users of the architecture. ISO 42010:2006 defines architecture description as a "work product used to express an architecture."[8] At any particular point in the maturity of an architecture description, it takes the form of a boundary object and contributes to different stakeholders understanding and using the models that it contains. Thus the architectural description has presentations tailored to the perspectives of different stakeholder communities.

Smolander, with refinements by Purao, identified four different ways in which the architecture description is used in the domain of software engineering, which can be extended to enterprise situations.[13] Some practitioners use the description as a blueprint specification for implementation. Some use it as literature for current and future users. Some use it to communicate with others for achieving a common understanding. And, still others use is to make decisions about implementation. While all of these uses are inter-related, each makes a different demand on the architecture description. How we create architecture descriptions to serve these diverse needs throughout the enterprise life cycle is far more art than science.

Consider that each of these uses has an interface with the object that is the architecture description – analogous to four different object type interfaces. Then to provide common understanding for all communities of practice, there will need to be ways of transforming the architecture description content into meaningful information, appropriate to the interfaces that serve the various communities. As the description and the architecture it describes evolve, more transformations of intention and description will also occur.

Because there are at least two efforts occurring with respect to the architecture description, *i.e.* its creation and its application, various kinds of transformation may predominate within each of those efforts. The transformations necessary to meet the needs of stakeholders at the boundary of the architecture description where utility is made available are different than the transformations that occur as the architecture description evolves to maturity. These transformations are considered at length in Sect. 6.

4 Time

Time is of course present in any evolution. Evolution is, in turn, intimately intertwined with processes in either the creation (architecting) or the application life cycles. Time of the first variety is a context of the architecture. Time of the second variety is a subject of that architecture. This clearly appears in Zachman's Architecture Framework,[16] where the "When" column is all about time in the target system. The "When" column is thus the realm of scheduling. The architecture provides for operational scheduling but a change in the schedule does not perturb the architecture.

These processes involve interactions between the varying levels of abstraction (meta-ness), the life cycle phases and the different stakeholder communities.

A distinct dimension of *Causality* is closely associated with time; in fact, it is too closely associated, in that a dangerous confusion between the two is common. Causality reflects the fact that all enterprise architectures have a dimension which spans life cycle phases from initial conception to realization. It is typically drawn with conception toward the top and implementation toward the bottom and reflects purpose in that objectives of artifacts in upper phases are realized by artifacts in lower phases – it is thus the "purposive" dimension. Although this dimension constrains[8] the temporal order of the architectural process activities, it reflects only the order of these processes and not the time at which activities occur. In operations research terms, it is the dimension of PERT charts as opposed to the schedule derived from those charts. GERAM[6] wisely distinguishes process *stages* (time) from architectural *phases* (dependency).

The specific focus in this paper is the concerns that arise from an interaction of different meta-levels and transformations that are necessary across different phases.

5 Dimensions

Causality is one of four dimensions relevant to enterprise architectures. Time is not one of these four dimensions, as they occur in a static architectures, "snapshots" during the architectural process. There may be yet further dimensions beyond these four, dimensions which are designated by some framework or methodology, such as Zachman's Interrogative dimension.[16] But these four dimensions are always present and always relevant. In addition to Causality, there are three distinct scale dimensions, shown with the extremes of their respective scales:

[8] Or should constrain. The old software engineering wisecrack "You folks start programming and I'll find out what we are supposed to do." reflects the occasions when purpose is not properly established.

Abstractness:	abstract	$\leftrightarrow$	concrete
Generality:	general	$\leftrightarrow$	particular
Granularity:	course	$\leftrightarrow$	granular/detailed

These scales are often confused, causing troubling entanglements; hence it is valuable to examine and carefully distinguish the three scales.

First, consider the words. These three scales are named by one of their extremes; that the chosen name is the top extreme for the first two and is the bottom for the last indicates that we most commonly "look up" Abstractness and Generality while we "look down" Granularity. Abstractness is the dimension of 'meta-ness'; it is obviously inadvisable to use such an overloaded name. The label "Detail" is used in place of "Granularity", but adding detail does not always change Granularity. For example, we may add detail to the description of an automobile engine in (at least) two ways: supplying missing facts, such as specifying an engine's displacement, or exposing sub-components, such as block, head, pistons, *etc.* Adding facts does not increase Granularity, exposing sub-components obviously does.[9]

These three scales are independent. Processes may be decomposed in a Data Flow Diagram (DFD) and entities may be generalized in an Entity-Relationship (ER) model, but these steps do not change the level of abstraction of the DFD or ER model. Nonetheless, it is common to have co-occurrence at the extremes of the scales (a module is concrete, particular, and fine grained).

Figure 1 illustrates Abstractness as distinct from Generality. Granularity does not appear, as any one diagram only has static detail. A more detailed version of Fig. 1 could be obtained by adding attributes to the relations.

Contrasting the order structure of the three scales, Granularity has arbitrary but often many levels with tree-like structure (one-to-many along decomposition), Generality has a few levels with a slim many-many structure (building-type and building-style are distinct generalizations in the realm of civil architecture), while Abstractness has fixed levels with a many-to-many structure that is quite narrow at its upper levels.[10] In tree-structured hierarchies, Granularity and Generality often seem to have an obvious inverse relationship, but these must be carefully considered when intertwined with meta considerations. For example, "all zip codes in HR must be nine digits" is fine grained (highly detailed) in the information model but the implementation of this decision applies widely across the organization.

6 Transformations

Enterprise architecture artifacts evolve through human-mediated transformations. We suggest six kinds of transformations: *projection*, *instantiation*, *specialization*, *refinement*, *derivation*, and *linking*. Each has different decisions and constraints, which of course vary across enterprise life cycle phase and meta-level.

[9] However, "highly detailed" is synonymous with "fine grained".

[10] This seems to distinguish systems architects, who use a restricted set of carefully crafted abstractions, from philosophers, who abstract at will.

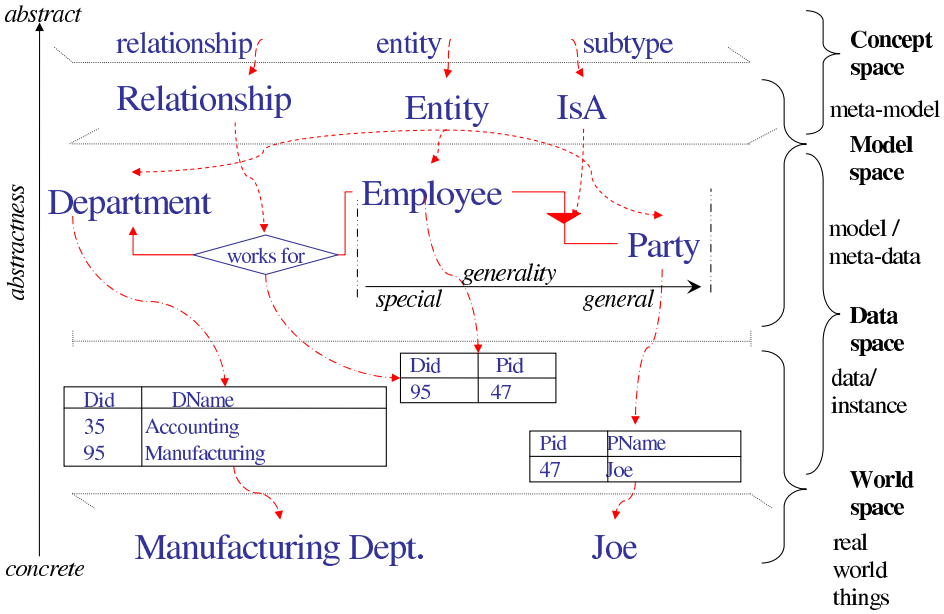

Fig. 1. Example with Abstractness and Generality

The above classification is not intended to partition the transformation space into non-overlapping regions. Indeed, examples below will discuss when a transformation of one kind may be thought of as a different kind. Rather, the intention is to distinguish how the kinds of transformations impact practice. An example of such interaction is addressed at the end of Sect. 6.4 below.

Transformations may be characterized by the four dimensions discussed above (Abstractness, Generality, Granularity, Causality) and by the distinction between *extractive* and *constructive* transformations. Some transformations dominantly occur in one of the creation or application life cycles while others are broadly relevant to both. Because time is ubiquitous in transformations, it is little use in distinguishing them.

6.1 Projection

Projection is the ultimate extractive operation. It never goes up any of the four dimensions but may move the focus of attention down. It is used by many stakeholder groups, albeit in different ways by different groups. There is a rising recognition of the importance of views and viewpoints in enterprise architecture

frameworks and standards (*e.g.* [8]). Projection is the mechanism to create such views, hence the growing importance of clearly specified projections for both the creation of architecture artifacts and for their use.

Projection is the most common way of extracting some portion of a model or set of models for use in a specific context. An SQL query on a relational database to produce a new relation is a widely used data projection method, often encapsulated in an SQL `VIEW`.

In the realm of architecting, the operational phase of an architecting enterprise may project, choosing from a catalog of models, to select those most suitable for a new enterprise.

Because projection typically looses context, care must be taken when applying additional transformations following a projection.

6.2 Instantiation

Instantiation is a constructive step that adds detail "down-meta", that is, to the more concrete form of some abstracted element. Figure 1 illustrates the results of an instantiation during creation (`Party` as an instance of `Entity`) and one during application (tuple `<47,Joe>` of `Party`). The final step from `<47,Joe>` to a living being is in the scope of epistemology and, coming full circle, Aristotelian metaphysics.

Instantiation is not only constructive but additive. During the creation life cycle, that addition is almost always human-mediated. During application, instances are often added by non-human actors, as when a monitoring instrument adds a temperature reading in a process control application.

6.3 Refinement

Refinement can occur in two different ways: decomposition and elaboration. A refinement may do only one of these or it may do both.

Pure decomposition happens most naturally to sets, because sets have no details other than membership.

Pure elaboration achieves refinement by adding details to an existing construct without creating something different. For example, adding attributes to an entity. When the added components come from a known source, this transformation is readily supported by tools. For example, a report generator provides a check list of attributes that are available for a report. The more attributes that are checked, the more information appears in the report. Refinement adds more attributes to the check box list.

The two ways combine when an element is decomposed into constituent parts. and each of those parts is then elaborated in more detail. When the sub-elements are of the same type as the original element, a hierarchy naturally results. The *part of* transformation is such a refinement.

Refinement is a constructive precursor of certain projections. That is, structures and sub-elements detailed during refinement are often used later to "drill down" to sub-elements or to their instances.

6.4 Specialization

Specialization constructs variations of abstract entities. It always happens above the bottom meta-level and never changes meta-levels. It adds detail at its meta-level and provides for additional instance data as well.

Figure 1 shows `Employee` as a specialization[11] of `Party`. The recognition that the `Joe` tuple corresponds to to the `Employee` entity is merely an instantiation (of those attributes that appear in the representation of `Employee`). The specialization that constructed the `Employee` (sub)entity occurred long before.

Specialization of an element is formally equivalent to first visiting the set of all potential instances of the element, then creating one or more subsets of those instances, and finally abstracting back up to new elements which are specializations of the original element. That equivalence is useful for understanding and even occasionally explaining Specialization and, while not directly useful to practice, does suggest a specialize-by-example tool. Also, it explains why Specialization is not meaningful at the lowest meta-level.

6.5 Derivation

Derivation can occur within almost any dimensionality. In its pure sense, derivation changes the form of element or elements without changing content.[12] In practice, derivation during the creation life cycle is also augmentative.

Because derivations are often complex, they work best when supported with tools. Tool support, in turn, requires precise characterization. An example derivation, during the creation life cycle, is the transformation from an ER model to a relational database schema. While this could be fully automated, the resultant schema is often improved with small amounts of human guidance, augmenting the transformation with pragmatic considerations.

Derivation during the application life cycle commonly appears in decision making or monitoring. For example, the calculation of total sales from individual transactions or of temperature variation over a stream of readings.

An example of the richness of derivation is when an element of one model is derived from a function applied to elements of another model. Figure 2 depicts elements from a Zachman Enterprise Architecture Framework description being mapped into a ISO 19439 life cycle integrated model description.

6.6 Linking

Unlike the previous transformations, whose motivations and mechanisms are well-formulated, linking is a great catch-all. A link, that is a direct connection, may be placed between arbitrary architecture elements. A link is always constructed but may, in principle, cross any dimensional boundaries. There, of course, is the power and the danger of linking.

---

[11] The word "specialization", used before as a verb, is here used as a noun.

[12] Our use of 'derive' is thus closer to the Latin '*transformare*', to change in shape, than to the current use of 'transform'.

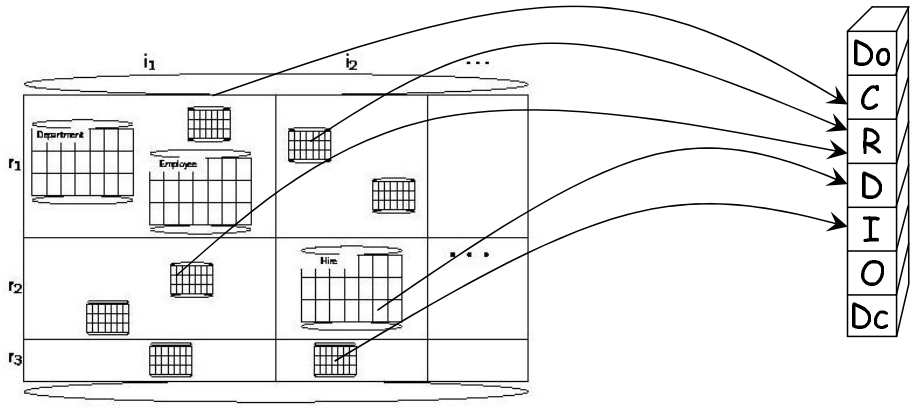

Fig. 2. Example of Derivation

The link is ubiquitous because it is the fundamental, indivisible unit of knowledge representation, as observed by Peirce in 1885.[11] Peirce also suggested that meaningful links are labeled (or typed), a recognition behind the development of RDF to add "Semantic" to the Web.[14] The message for us is that architecture frameworks should discourage and tools should prohibit untyped links.

Linking here is an explicit transformation. Links are often created in the process of other transformations as well, as seen in the example of Fig. 2.

Where an abstraction of a link exists, typically manifest as a Relationship in an ERM, the linking transformation is also an instantiation of the link's abstraction. That is, the linking that assigns `Joe` to `Manufacturing` in Fig. 1 is in fact instantiation. Such instantiation is the most common kind of linking during the application life cycle, which is indeed fortunate because the presence of a framing abstraction (as in an ERM) means that instantiations are safe.

7 Impact

The goal of understanding issues of "meta" and process is of course to maintain alignment of artifacts as the architecture evolves. First, it is essential that the meta-architecture be stable over the life of the project. Such stability is best achieved by using standards for architecture frameworks. Second, the evolution of the architectural abstractions - its meta components - must be traced. Again, standards and reference models facilitate such tracking, although more work must be done here. Finally, mechanisms to validate meta-alignment are required. These are the most difficult since they involve both understanding the practice of architectural evolution and developing the formalizations required to support validations.

References

1. Barringer, F.: A death is noted in the meta-media family. New York Times (June 5, 2000)
2. Carver, J., Williams, B.: Characterizing software architecture changes: A systematic review. Tech. Rep. MSU-081216, Mississippi State University, Department of Computer Science and Engineering (2008)
3. Eden, A., Hirshfeld, Y., Kazman, R.: Abstraction classes in software design. IEE Software 153(4), 163182 (2006)
4. Hofstadter, D.: Gödel, Escher, Bach: an Eternal Golden Braid. Basic Books (l979)
5. Howe, D.: The Free On-line Dictionary of Computing, http://foldoc.org, accessed Feb 26, 2008
6. IFIP-IFAC Task Force on Architectures for Enterprise Integration: GERAM: Generalised Enterprise Reference Architecture and Methodology (1999)
7. International Organization for Standardization: Automation systems and integration - Framework for enterprise architectures and models (ISO WD 15704), draft version of Jan. 2009, currently titled Industrial automation systems - Requirements for enterprise architecture methodologies.
8. International Organization for Standardization: Systems and Software Engineering - Architecture Description (ISO/IEC CD1 42010), Draft verson of Jan. 2010, currently titled Recommended Practice for Architecture Description of Software-Intensive Systems, `www.iso.ch`.
9. International Organization for Standardization: Industrial Automation Systems - Requirements for Enterprise–Reference Architecture and Methodologies (ISO 15704:2000) (2000), `www.iso.ch`.
10. Martin, R., Robertson, E.: "Meta" matters. In: International Conference on Information Resources Management. Niagara Falls, Ontario (2008)
11. Peirce, C.S.: On the algebra of logic. Amer. J. of Math. pp. 180–202 (1885)
12. Sim, S.: A small social history of software architecture. In: Proceedings of the 13th International Workshop on Program Comprehension (IWPC05). St. Louis, Missouri (May 2005)
13. Smolander, K., Rossi, M., Purao, S.: Software architectures: Blueprint, literature, language or decision? European Journal of Information Systems p. 114 (2008)
14. W3C: RDF Model Theory (2002), `www.w3.org/TR/rdf-mt/`
15. Webster's dictionairy, 7^{th} edition. online version
16. Zachman Institute for Framework Advancement: The Zachman Framework, various pages at `www.zachmaninternational.com`; the "Concise Definition" is particularly relevant.