An Overview of Enterprise Architecture Framework Deliverables

Frank Goethals
Frank.Goethals@econ.kuleuven.be
Naamsestraat 69
B-3000 Leuven
SAP-leerstoel 'Extended Enterprise Infrastructures'

Abstract: A number of enterprise architecture frameworks do exist. In this paper, we differentiate between two classes of frameworks: classic enterprise architecture frameworks, and federated enterprise architecture frameworks. From each class, a number of reputable frameworks are presented. Conclusions are made concerning what these frameworks learn us for setting up an Extended Enterprise architecture framework.

1 Introduction

During the years to come, the research at the SAP-leerstoel will focus on the creation of an Extended Enterprise architecture framework. Not much research has been done towards this specific topic yet. However, a number of enterprise architecture frameworks are available. In order to develop a framework, useful for the Extended Enterprise *integration* exercise, we should first investigate the existing frameworks.

TOGAF (The Open Group Architecture Framework, [1]) defines an architecture framework as follows:

> "*An architecture framework is a tool which can be used for developing a broad range of different architectures* [architecture descriptions]. *It should describe a method for designing an information system in terms of a set of building blocks, and for showing how the building blocks fit together. It should contain a set of tools and provide a common vocabulary. It should also include a list of recommended standards and compliant products that can be used to implement the building blocks.*"

Besides this, TOGAF states that two of the key elements of any enterprise architecture framework are:
- a description of the method by which the architecting activity should be done, and
- a definition of the deliverables this activity should produce.

However, the majority of the existing enterprise architecture frameworks focus on the second of these only [1].

In the literature review at hand, we present a selection of existing frameworks. This selection contains a number of frameworks mentioned by TOGAF [1] and by van den Heuvel and Proper [2]. Please, note that we only present the *deliverables* that the frameworks suggest. In future papers, we will also discuss methods to produce these deliverables. In what follows, we first set forth some basic concepts and terminology. Next we present the different frameworks, and finally, we draw some conclusions that could serve as a basis for further research.

2 Basic concepts

Through the years, the idea behind Architecture Descriptions (ADs) has evolved, producing the IEEE 1471-2000 standard on 'Recommended Practice for Architectural Description of Software-Intensive Systems'. IEEE 1471-2000 defines an 'architectural description' as *a collection of products to document an architecture*, whereas 'an architecture' is defined as *the fundamental organization of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution* [3]. While we are convinced about the accurateness of these definitions, many authors do not use these terms in this sense! In this literature review, we will stick to the terminology used by the specific authors. The reader should keep in mind that in an overwhelming part of the literature the word 'architecture' is used when 'architectural description' is meant.

Describing a software-intensive system is not straightforward. Before one can start drawing up ADs, the scope of the architecture activity should be determined. TOGAF [1] mentions four dimensions on which the scope may be defined and limited:

- The *enterprise* scope, i.e., what is the 'enterprise' that will be described? This question is of particular interest in the Extended Enterprise (EE) setting: it has to be determined whether each company will develop ADs independently (while allowing the integration of the ADs later on), or whether one large centralised AD will be drawn up for the total EE.
- The level of *detail*. The description should be detailed enough to be useful, but too much detail makes the ADs confusing and costs money and time.
- The *time* horizon, i.e., is the effort aimed at the description of the as-is situation, or is it meant to describe a target-architecture (or some transitional architectures)?
- The architecture *domains*. This has to do with the idea of viewpoints. This topic is of particular interest to this paper, so we investigate it in more detail:
  IEEE 1471-2000 defines a 'view' as *a description of the entire system from the perspective of a set of related concerns.* As such, *a view is composed of one or more models.* A 'viewpoint' can be regarded as *a standard or template for constructing a view,* it says *where you look from*.
  The term 'viewpoints' attracted attention in 1996 in the Reference Model of Open Distributed Processing (RM-ODP, see Section 3.1.5).[1] The underlying idea was that for each system a number of roles could be identified that have an interest in the system. While each role is interested in the same system, their relative views of the system are different, and they have different requirements. Therefore, it seems interesting to describe the system from different viewpoints, each of which is chosen to reflect a set of concerns[2].
  The question is which viewpoints are relevant. ADs are not a goal as such, but are a means to realise other goals. The viewpoint selection should be driven by the knowledge of what the ADs will be used for when ready. IEEE 1471-2000 states that the stakeholder concerns should be used to justify the views [3], and Hilliard [4] talks about traceability of views. Consequently, before arbitrarily drawing up an AD, one should determine what the description will be used for.
  IEEE 1471-2000 itself does not tell which views are relevant. Van de Heuvel and Proper [2] state that many authors have introduced view models (i.e., a coherent collection of viewpoints, as those presented in the remainder of this paper), but that it is unclear when to use which view model. Van de Heuvel and Proper put forward the proposition that it should be possible to compose an optimal view model, aimed at a specific context. We conclude that there should be an optimal view model for the Extended Enterprise integration (EEi) exercise too. However, at this moment it is still being investigated what the contribution of ADs could be in handling EEi challenges. Consequently, it seems unrealistic to assume the optimal view model would have been defined already.

---

[1] Other authors (as Zachman for example, see below) discussed the *concept* of viewpoints earlier, but did not use the term 'viewpoint'.

[2] To clarify these concepts we may refer to the Model Driven Architecture. In the MDA [15] it is stated that a model that is based on specific abstraction criteria (suppressing irrelevant details) is referred to as a *model from the viewpoint defined by those criteria*, or in short as a *view* of the system.

The paper at hand focuses on the first and the fourth of the four dimensions proposed by TOGAF because existing frameworks primarily pay attention to these dimensions. In our opinion, all frameworks can be applied independently of the chosen scope with respect to the second and the third dimension.

3 Existing enterprise architecture frameworks

As stated, a number of enterprise architecture frameworks do exist. While some have been described extensively, most are only explained in a (number of) paper(s). In our discussion we do not pay equal attention to every framework. After all, some frameworks are regarded as being more important than others, and some frameworks do resemble each other a lot.

We distinguish between two classes of frameworks on the basis of the first dimension mentioned by TOGAF, i.e., the meaning of the term 'enterprise'. While frameworks in the first class focus (mostly implicitly) on one centralised enterprise, the second class contains frameworks that aim at describing enterprises that are more or less linked by a kind of umbrella organisation. We call all models in the first class 'classic enterprise architecture frameworks' (discussed in Section 3.1). All models in the second class are grouped under the name 'federated enterprise architecture frameworks' (Section 3.2).

3.1 Classic enterprise architecture frameworks

In what follows, we first present the ideas of John Zachman (Section 3.1.1), who is considered to be one of the pioneers in this domain. Another reputable framework is offered by Kruchten, who introduced the 4+1 View Model of Architecture (see Section 3.1.2). Other, similar models are the one of Soni, Nord and Hofmeister (Section 3.1.3), that of Tapscott and Caston (Section 3.1.4), and ISO's RM-ODP (Section 3.1.5). We end the discussion with a note on OMG's Model Driven Architecture (Section 3.1.6).

3.1.1 The Zachman-framework

John Zachman is regarded as being the person who introduced the idea of 'Information System Architecture' (ISA). He considered information system design by analogy to the work steps and the representations of the classical architect and producers of complex engineering products. When developing an IT system, it is obvious that many parties are involved. Businessmen have business requirements, which should be translated into ICT-requirements and next be transformed into a combination of software and hardware that fulfils the requirements. What follows are the ideas of Zachman and of Zachman and Sowa as presented in [5] and [6] respectively.

First and for all, it is noticed that there is not *an* information system architecture, but a *set* of such architectures. The Zachman framework relies on the fact that architecture is relative to the perspective from which you look at it, and to the question that is in mind when drawing the architecture. As such, the framework (as depicted in Figure 1) presents two dimensions along which architecture descriptions could be categorized. The first dimension (the succession of the rows in Figure 1) concerns the different perspectives of the different participants in the systems development process. The second dimension (the sequence of the columns in Figure 1) deals with the six basic English questions *what*, *how*, *where*, *who*, *when* and *why*.

	Data (What)	Function (How)	Network (Where)	People (Who)	Time (When)	Motivation (Why)
Scope (Ballpark View)	List of things important to the business	List of processes the business performs	List of locations in which the business operates	List of organizations important to the business	List of events / cycles significant to the business	List of business goals / strategies
Business model (Owner's view)	e.g. semantic model	e.g. business process model	e.g. business logistics system	e.g. work flow model	e.g. master schedule	e.g. business plan
System Model (Designer's view)	e.g. logical data model	e.g. application architecture	e.g. distributed system architecture	e.g. human interface architecture	e.g. processing structure	e.g. business rule model
Technology Model (Builder's view)	e.g. physical data model	e.g. system design	e.g. technology architecture	e.g. presentation architecture	e.g. control structure	e.g. rule design
Detailed Representations (Subcontractor)	e.g. data definition	e.g. program	e.g. network architecture	e.g. security architecture	e.g. timing definition	e.g. rule specification
(Functioning system)	(e.g. data)	(e.g. function)	(e.g. network)	(e.g. organization)	(e.g. schedule)	(e.g. strategy)
Descriptive model	Entity Relationship Entity	Input Process Output	Node Line Node	Organization Reporting Organization	Event Cycle Event	Objective Precedent Objective

Figure 1: The Zachman framework [6]

FIRST DIMENSION: different perspectives of different participants. Zachman noticed that the set of architectural representations that are produced during the process of constructing a building may be generic to the process of building any complex engineering product, including information systems. As such, five architectural representations are proposed, of which the second, the third and the fourth are considered to be fundamental (as they each present a view for a different player in the game):

- The *scope description* is a gross representation of size, shape and scope.
- The *business* model (owner's view) describes the product the owner needs to fulfil some goal. This model, containing the owner's perceptions and requirements is translated into
- the (*information*) *system model* (designer's view).
- The *technology model* (builder's view) is the adaptation of the designer's view to incorporate constraints of the laws of nature and available technology to make the product producible.
- The *detailed description* (out-of-context view) does not depict the final product in total but concerns parts or subsections (components) of the total structure. This description is more oriented to the actual implementation activities.

Each representation (and each perspective) presents a different set of constraints. These constraints are additive, and the constraints of one model should not be inconsistent with other models. This means that designers should search for gaps (inconsistencies) between different representations and should - when needed - modify the models in order to remove all inconsistencies. The five perspectives are presented as the first five rows in Figure 1. The sixth row in this matrix is sometimes presented as a *sixth perspective*. This row contains the functioning systems (and is – in our opinion – as such a totally different kind of architectural description than the other five perspectives, as it is not just a description on paper, but exists in reality).

SECOND DIMENSION: basic English questions. The architectures can be looked at from a different functional point of view. One may be interested in *what* entities the system works with, in *how* the system works, in the location *where* the system is placed, in *who* (which agent) does something with the system, in *why* the system does something or in *when* something is supposed to happen. According to these questions, different types of descriptions can be made, according to the different aspects of the object being described. For every column, i.e., for every question in mind, a basic descriptive model is proposed (the last row in Figure 1). The way this descriptive model is filled in depends on the perspective that describes the system.

To clarify the idea of perspectives, we walk top-down through the *data* column[3]. The scope description would then consist of a list of basic entities, important to the business. The business model shows these entities in a diagram, related with relationships which denote business rules, e.g., *product* A should be *shipped* from *warehouse* XYZ only. The information system model looks at entities like records that are related by a data relationship. At the level of the technology model, it is decided which system will be used (DB2 for example), and entities are (for example) regarded as rows, and relationships as keys. The out-of-context perspective, finally, presents Data Definition Language.

For every different type of description, there are different perspectives for each of the different participants. Consequently, the combination of the two dimensions results in a matrix containing 30 (5 x 6) cells. Each description (cell) stands alone and each is different from the others, although all the descriptions may pertain to the same object and therefore are related to one another. As such, the matrix offers a taxonomy for relating things in the real world to computer representations: "*The ISA framework serves as a convenient classification scheme or "periodic table" for information entities*" [6]. We can conclude that there is not *an* information system architecture, but a *set* of architectures that are additive and complementary. Changes in one architecture are likely to effect other architectures.

A corollary to the rule that each cell is unique is that a plethora of information system design formalisms have emerged as well as a plethora of methodologies, each applicable to some subset of cells. Zachman and Sowa argue for a single common language (conceptual graphs) to describe the subject of all the cells (and their interrelationships), rather than using an ideal, special notation for each cell separately.

---

[3] Please note that in each of the fundamental views, the descriptive model for the data column, entity-relationship-entity, is respected.

3.1.2 Kruchten's 4+1 View Model of Architecture

Philippe Kruchten's *4+1 View Model of Architecture*, presented in IEEE Software in November 1995 [7], is also considered to be an important model for architectural description. Kruchten's view model, as depicted in Figure 2, describes software architectures using five concurrent views. One of these views, the *scenarios*-view, is redundant with the four other ones. The scenarios, which are instances of important use cases, are used to show that the elements of the four other views work together seamlessly. Besides this, the scenarios can also function as a driver in searching architectural elements during the architecture design.

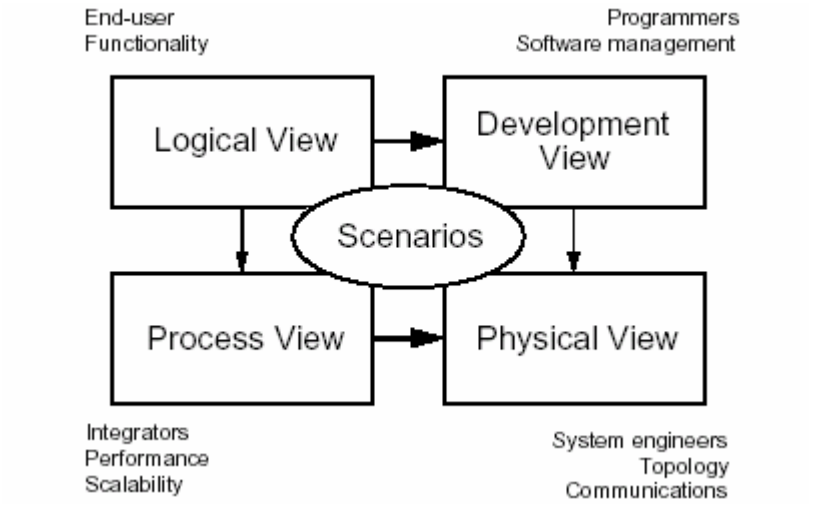

Figure 2: Kruchten's 4+1 View Model of Architecture [7]

The four fundamental views in this view model are:

- the logical view,
- the process view,
- the development view, and
- the physical view.

The **logical** view describes (in case of OO-design) the design's object model. To represent this view the use of class diagrams (to show the classes and their relationships) and state-transition diagrams (to define an object's internal behaviour) is suggested. The logical view is meant to describe the services the system should provide to its end users. Please note that Kruchten primarily talks about OO-design. In other cases, entity-relationship diagrams may be used in the logical view.

While the logical view primarily supports functional requirements, the **process** view takes into account some non-functional requisites, such as performance and system availability. This blueprint is all about concurrency and synchronization aspects: for each operation of each class identified in the logical view it specifies by which thread of control it is executed. The process view is used to estimate message flow and process loads.

The **development** view describes the software's static organization in its development environment. This is done through module and subsystem diagrams that show the system's export and import relationships.

The **physical** view shows the mapping of the software (the elements defined in the three other views) onto the hardware. Kruchten remarks that the physical blueprints may become very messy in large systems, and that these blueprints therefore should be organised in several forms, with or without the mapping from the process view.

Elements in one view are connected to elements in other views. Consequently, the various views are not fully independent. While the logical view considers each object as active and potentially concurrent, in the process view it is recognized that it is not practical to implement each object with its own thread of control. Also, the logical view and the development view may be very close (especially in small projects).

Please note that Kruchten states that '*not all software architectures need every view in the 4+1 View Model. Views that are useless can be omitted*'.

3.1.3 Soni, Nord and Hofmeister

The basic paper of Soni, Nord and Hofmeister [8] is all about software architecture, and its role in the design and development of large systems. After doing a number of case studies, the authors noticed that all structures within software systems fell into four broad categories: the conceptual structure, the module structure, the code structure, and the execution structure. Again, each of these categories is said to address different engineering concerns, and to describe the entire system from a different perspective. The authors attribute many of the successes in their case studies to this 'separation of concerns' and they state that some reported problems '*were a direct result of merging or intermingling these structures*'. The strength of this concept is that this separation allows the structures to develop independently, while maintaining the relationship between the structures. Besides this, combining structures into one structure reduces the understandability of the system.

The four categories should be interpreted as follows:

The **conceptual** architecture is a high-level and 'domain-specific' structure of the system, i.e., it describes the system in terms of its major design elements (as display service, report generator, database manager, and print service in a monitoring application) and the domain-specific relationships among these elements. This structure is independent of implementation decisions. It contains functional blocks, functional diagrams, etcetera.

The conceptual architecture is implemented by module structures. The **module** architecture is meant to support programming-in-the-large. As such, it reflects implementation decisions. It encompasses two structures: a *functional decomposition structure*, showing how the system is logically decomposed into subsystems, modules and abstract program units (and the components' interrelationships in terms of exported and imported interfaces); and *layers*, which reflect design decisions based on allowable import/export relations and interfacing constraints.

The modules of the module architecture and the components in the conceptual architecture are assigned to runtime elements in the **execution** architecture. This architecture is used for performance analysis, monitoring, and tuning. It describes the dynamic structure of a system in terms of its runtime elements, communication mechanisms, resource allocation, and assignment of functionality to runtime elements (i.e., hardware, and software like operating

system tasks). It also shows design decisions related to location, migration, and replication of a system's runtime elements.

Finally, the **code** structure describes how the source code, libraries, and binaries are organised in the development environment.

The authors noticed that there is frequently made a distinction between the *application software* architecture, and the *platform* software architecture. Besides this, not every structure was made explicit in every case (what often complicated the software development exercise). This remark especially applied to the conceptual structures, which were often implicit and embedded within the documentation of other structures.

The four structures contain the results of design decisions made at different moments in the development process, and consequently each is developed and used by different teams for different purposes. It is worth mentioning that the degree to which each structure reflects implementation decisions varies, and that the degree to which each structure is expected to change also varies from structure to structure.

Van de Heuvel and Proper [2] remark that the frameworks of Kruchten and of Soni, Nord and Hofmeister are very similar.

3.1.4 Tapscott and Caston

In the vision of Tapscott and Caston [9], an architecture comprises five interrelated viewpoints, which are depicted in Figure 3.

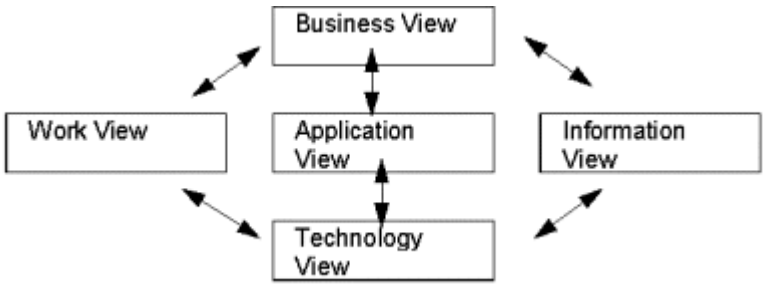

Figure 3: The five architectural views proposed by Tapscott and Caston [9]

The **business view** presents a network of *service functions* linking clients and servers. Information flows between these service functions trigger business activities and further interactions. This view is regarded as the starting point for developing the other views, emphasizing the fact that businesses should first be reengineered, before work processes are redesigned and IT applications developed.

The **work view** shows the work activities and the resources (people, information, …) that make up the service functions. The work view can thus be seen as a set of business process models. As such, this view is useful in building the architectural requirements with business managers and intended users.

The business view is not only the basis for the work view, but also for the **information view**. In this view the fundamental information resource requirements are modelled, which can be derived from the basic service functions of the business.

The **application view** links the work view and the information view. After all, applications, which support the work activities of business processes, vouch for the creation, update, access and deletion of information.

Finally, the **technology view** provides the required technology platforms (hardware) and application environments (to build or develop specific functionality). This view is the enabling infrastructure for work, applications and information.

3.1.5 ISO's RM-ODP

The ISO Reference Model of Open Distributed Processing (RM-ODP, [23]) was defined to deal with distributed systems. It defines five different viewpoints (with respective models):

The **enterprise** viewpoint reveals the high level enterprise requirements and policies and is aimed at the needs of the users of the information system. It looks at *what* the system is required to do for the enterprise.

The **information** viewpoint identifies information elements, manipulations that may be performed on these elements, and information flows in the system.

The **computational** viewpoint deals with the definition of application components, their interfaces and the interactions between components.

The **engineering** viewpoint looks at the organization of the underlying distributed infrastructure, which should give support (i.e., offer services) to applications. As such, it can be considered as '*an extended operating system, spanning a network of interconnected computers'*. Note that the distribution – which is transparent in the computational model – is not transparent in the engineering model. Nevertheless, this model still does not look at real computers, nor does it consider the *implementation* of mechanisms or services identified in this model.

The **technology** model identifies technical artefacts for the earlier defined structures: engineering mechanisms, computational structures, information structures, and enterprise structures.

Once more, each viewpoint is used to describe the system from a particular set of concerns. The viewpoints '*should not be seen as architectural layers*', but rather as different abstractions of the same system, which should all be used to completely analyse the system: the combination of the five viewpoints is supposed to provide a complete description of the system. The purpose of the viewpoints is to '*position services relative to one another, to guide the selection of appropriate models of services, and to help in the placement of boundaries upon ODP'.*

3.1.6 OMG's MDA

The Model Driven Architecture (MDA) [10] defines an approach to IT system specification that distinguishes between the specification of *system functionality* on the one hand (a Platform-Independent Model, PIM) and the specification of *the implementation* of that functionality on the other hand (a Platform-Specific Model, PSM). One motivation for making this distinction is the fact that it allows the same functionality-specifying model to be realized on multiple platforms through mapping.

The MDA provides guidelines on the architecture of models, but does not deal with viewpoint selection: '*the choice of viewpoints to be used in a system specification is a modelling choice*'. We may say that the MDA only provides two viewpoints: the PIM and the PSM. These views could contain more and less abstract models; the PIM could for example contain a *computation independent business model*, and a *platform independent component view*. Please note that the MDA insists on it that specifications should be formal[4]: '*a specification that is not formal [...], is not a model*'. After all, one of the key features of the whole MDA approach is the notion of mapping, i.e., '*using a set of rules and techniques to modify one model in order to get another model*'. These mappings are used to (sometimes automatically) transform for example one PIM into another PIM or into a PSM. The MDA puts forward the Unified Modelling Language (UML) to model systems. The adequacy of UML for supporting architecture semantics itself has, however, been widely questioned [11].

3.2 Federated enterprise architecture frameworks

Bearing in mind the doctoral research on Extended Enterprise architectures, the concept of federated enterprise architectures seems very interesting, as it deals with creating integrated or integratable architectures. The TOGAF [1] mentions three federated enterprise architecture frameworks, which will be discussed in the remainder of this paper: the Federal Enterprise Architecture Framework (Section 3.2.1), the C4ISR Architecture Framework (Section 3.2.2), and the Treasury Enterprise Architecture Framework (Section 3.2.3).

3.2.1 Federal Enterprise Architecture Framework

The Federal Enterprise Architecture Framework (FEAF [12]) was developed by the Chief Information Officers (CIO) Council to promote interoperability, the shared development of common federal processes, and the sharing of information among agencies of the federal government and other governmental entities. To come to an integrated architecture the CIO council did not suggest the use of one centralized architecture, but opted for a 'segment architecture' approach. This approach allows critical parts (architectural segments) of the overall federal enterprise to be developed individually within a structured enterprise architecture framework. *'A segment is considered to be an enterprise within the total Federal Enterprise'* [12]. The Framework itself is '*a place-holder for the content once developed*'.

The FEAF was built on the enterprise architecture model of the National Institute of Standards and Technolgoy (NIST). The NIST model allows for organizing, planning, and building an integrated set of information and IT architectures. It contains five layers which

[4] With 'formal', the MDA means that the language should have a well-defined form (syntax), meaning (semantics), and possibly rules of analysis, inference, or proof for its constructs.

are defined separately, but are interrelated: the business architecture, the information architecture, the information systems architecture, the data architecture, and the delivery systems architecture (hardware, software, and communications)[5].

We note that the FEAF is supported by federal enterprise architecture *principles*, which govern and represent the criteria against which all potential investment and architectural decisions have to be weighed. The principles concern (among other things)

- the establishment of federal interoperability standards to come to an open system architecture,
- compliancy mechanisms to ensure that investments are funded by business and architectural decisions,
- data standardization (including a common vocabulary and data definition), and
- opting for proven market technologies.

The FEAF is depicted in Figure 4. Basically, the FEAF shows how the architecture should move, under the impulse of architecture drivers and a strategic vision, from the current architecture to the target architecture. Architecture descriptions basically contain two layers: a business architecture and a design architecture. The latter is actually composed of three architectures: a data architecture, an applications architecture, and a technology architecture. So with 'design' is meant data, applications, and technology.

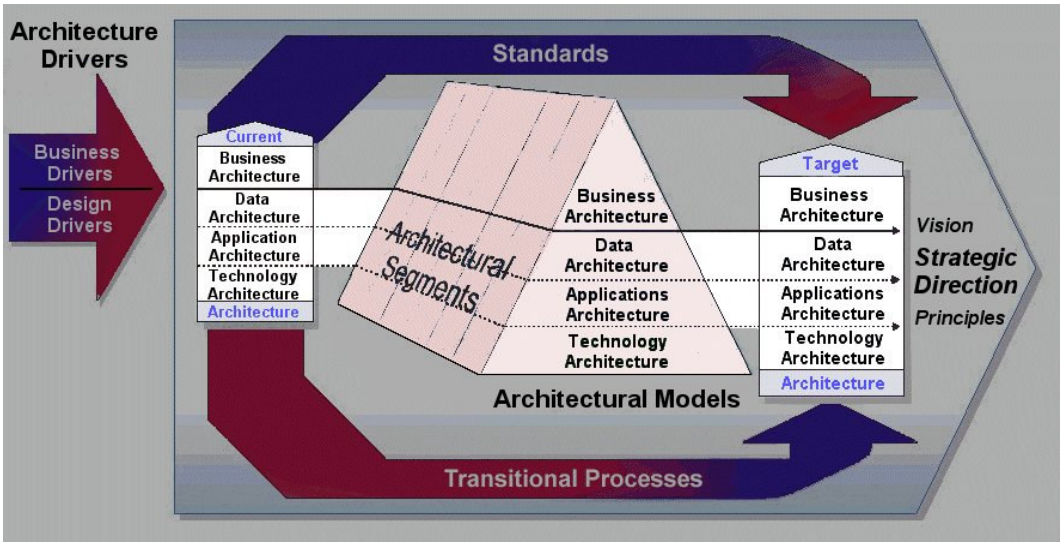

Figure 4: The FEAF [12]

When looking at the FEAF at a lower level of detail, we can represent the FEAF as a matrix, representing a part of the Zachman framework. The business architecture is expressed in the top two rows of the Zachman framework, while the models contained in the third, fourth, and fifth row define the design architectures. Besides this, the three names of the design

[5] Please note that LEAP, the LMI (Logistics Management Institute) Enterprise Architecture Practice, which is based on the FEAF, utilizes five architectural layers (just as the NIST): business, information, application, data, and technology [16]; whereas the FEAF consolidates the information architecture with the business architecture. Further information on why this is so is currently not publicly available.

architectures appear as three column headings in this matrix. The detailed FEAF is depicted in Figure 5. It is clear that only three of the six columns in the Zachman Framework are being incorporated. The three remaining columns (i.e., *who*, *when* and *why*) will be considered for incorporation into the FEAF in the future. In the FEAF, the rows are called 'perspectives', the columns 'focus'.

	Data Architecture (entities = what)	**Applications Architecture (activities = how)**	**Technology Architecture (locations = where)**
Planner's View Objectives/Scope	List of Business Objects	List of Business Procesess	List of Business Locations
Owners'sView Enterprise Model	Semantic Model	Business Process Model	Business Logistics System
Desiger's View Information Systems Model	Logical Data Model	Application Architecture	System Geographic Deployment Architecture
Builder's View Technology Model	Physical Data Model	System Design	Technology Architecture
Subcontractor's View Detailed Specifications	Data Definition "Library or Encyclopedia"	Programs "Supporting software components (i.e., Operating Systems)"	Network Architecture

Figure 5: The FEAF (more detailed) [12]

The models which define the business architecture of the federal enterprise are considered essential and must be completed to develop a segment architecture description that can be commonly understood and integrated within and across the Federal Enterprise. There are three important things to note. First, the constraints of each perspective are additive, i.e., the constraints of higher rows affect the rows below (but the reverse is not necessarily true). Secondly, one is not obliged to model all the cells, but if a cell is not made explicit there is a risk of making invalid assumptions. This could result in increased costs and a rescheduling of the implementation. Thirdly, the concept of slivers (and slices) as a portion of a cell or of several cells is important to realise the segment architecture approach. It provides a way to relate the segmentation of the federal enterprise to understandable parts without losing the definition of the total integration. Please note that if cells are not made explicit enterprisewide, other slivers in the same cell may not relate to or integrate with the previous slivers unless by chance, or unless steps are taken to pre-integrate following efforts.

3.2.2 C4ISR Architecture Framework

C4ISR (Command, Control, Computers, Communications, Intelligence, Surveillance, and Reconnaissance) has emerged as a successor of the TAFIM, the Technical Architecture Framework for Information Management. C4ISR is a framework ought to give architectural guidance for a number of related domains of the Department of Defense (DoD) in the United States. The project was started up after the US DoD realised that DoD organizations were developing architectures representing specific contributions and relationships with respect to overall DoD operations, but that significant differences in content and format were inhibiting the ability to rationalize or compare different architecture descriptions. These disparate and

unrelatable architecture descriptions brought along non-integrated, non-interoperable, and non-cost effective capabilities in the field [1].

C4ISR considers three viewpoints, namely, an operational viewpoint, a systems viewpoint, and a technical viewpoint. C4ISR is meant to ensure that different architectural descriptions are interrelatable among each organization's (operational, systems, and technical) architecture views, *and* integratable and comparable across organizational boundaries. It not only provides uniform methods for describing information systems, but also for describing their performance in context with mission and functional effectiveness [13]. The three views and their relationships are shown in Figure 6.

The **operational** view is defined as '*a description of the tasks, activities, operational elements, and information flows required to accomplish or support a military operation*'. This view could for example be used to examine business processes for reengineering or technology insertion, or to investigate doctrinal and policy implications. Operational views are generally (but not always) independent of organization or force structures and of technology, and are generally driven by doctrine.

The **systems** view is '*a description, including graphics, of systems and interconnections providing for, or supporting, warfighting functions.*' As such, this view describes how multiple systems (from sensors and information systems to shooters) link and interoperate, and it may show the internal construction and operations of particular systems. At this level, system and component performance parameters are specified. The primary purpose of this view is thus to enable/facilitate operational tasks and activities through the application of physical resources. Please note that the operational architecture is generally not systems-dependent. The systems architectures are, however, based upon and constrained by technical architectures.

The **technical** view is '*the minimal set of rules governing the arrangement, interaction, and interdependence of system parts or elements, whose purpose is to ensure that a conformant system satisfies a specified set of requirements.*' This view thus provides the technical systems-implementation *guidelines* (technical standards, conventions, rules and criteria) upon which engineering specifications are based, common building blocks are established, and product lines are developed. Well-planned and comprehensive technical views facilitate integration and promote interoperability across systems and compatibility among related architectures.

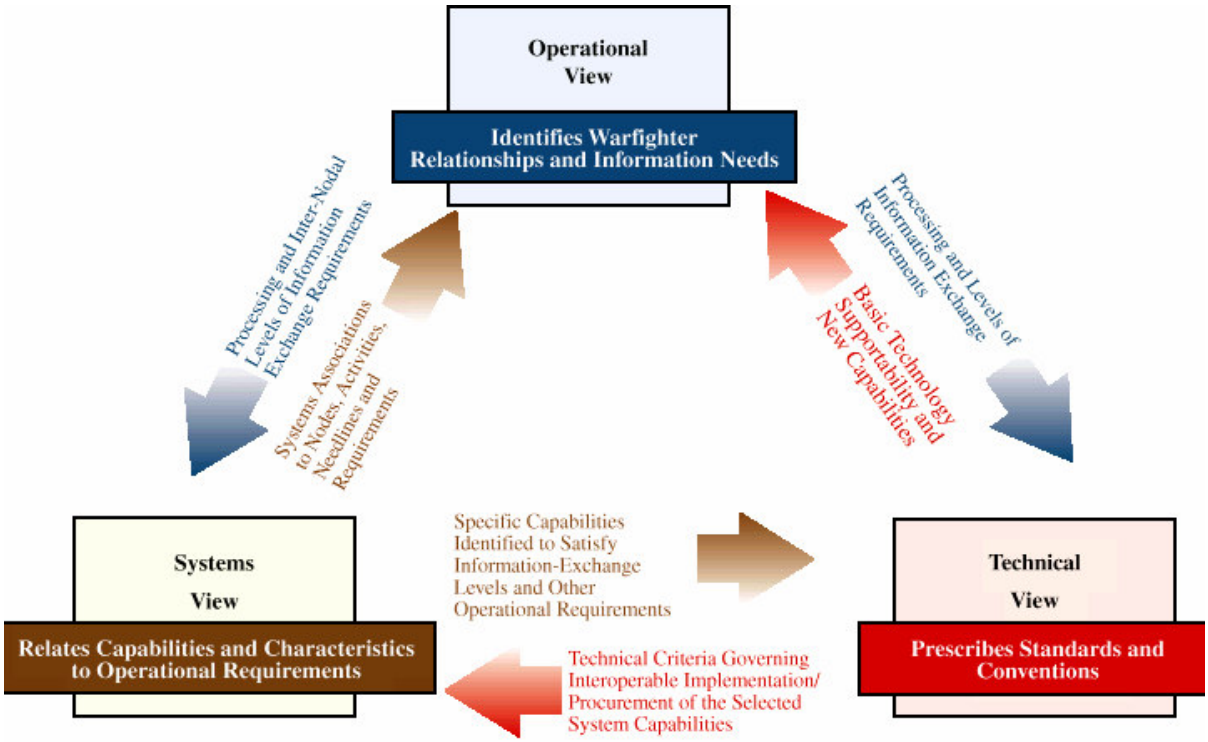

Figure 6: C4ISR views and their linkages [13]

Please note that the framework also presents a number of rules architectures should obey to be compliant with the framework. Each architecture should for example use the common terms and definitions specified in the framework and provide a minimum set of 'essential products'. It is believed that the latter is imperative to come to architectures that are comparable and integratable. Please note that the framework distinguishes between 'essential products' (which are indispensable if one wants to develop integratable architecture descriptions), and 'supporting products' (which are developed depending on the purpose of the architecture). An overview of the essential products is given in Table 1.

Table 1: C4ISR essential products (adapted from [13])

Applicable Architecture View	Architecture Product	General Nature
All Views (Context)	Overview and Summary Information	Scope, purpose, intended users, environment depicted, analytical findings of the architecture.
All Views (Terms)	Integrated Dictionary	Definitions of all terms used in all products.
Operational	High-level Operational Concept Graphic	High-level graphical description of operational concept (high-level organizations, missions, geographic configuration, connectivity, etc.).
Operational	Operational Node Connectivity Description	Operational nodes, activities performed at each node, connectivities & information flow between nodes.
Operational	Operational Information Exchange Matrix	Information exchanged between nodes and the relevant attributes of that exchange such as media, quality, quantity, and the level of interoperability required.
Systems	System Interface Description	Identification of systems and system components and their interfaces, within and between nodes.
Technical	Technical Architecture Profile	Extraction of standards that apply to the given architecture.

3.2.3 Treasury Enterprise Architecture Framework

In July 2000, version 1 of the Treasury Enterprise Architecture Framework (TEAF, [14]) was published. The Department of the Treasury consists of a number of bureaus and offices that are considered to be individual enterprises. Nevertheless, it is recognized that an integrated view of the total enterprise is needed to effectively manage IT investments. An enterprise architecture program is established to identify, document and manage the interrelationships among the business organizations, their operational processes, and their supporting information systems. The TEAF aims at the establishment of a common enterprise architecture structure, consistent practices, and common terminology; what should facilitate integration, information sharing, and exploitation of common requirements across the department.

The TEAF is a revision to the Treasury Information Systems Architecture Framework (TISAF). Just as the C4ISR framework, the TEAF includes descriptions of a number of work products for documenting and modelling enterprise architectures. It is explicitly stated that '*these work products align with FEAF models and with Department of Defense (DOD) Architecture Framework products*'. Please note that the TEAF also includes a list with ten core architecture principles (fundamental beliefs). Besides this, the TEAF requires each organization to formulate architecture principles appropriate to their mission and situation.

The TEAF describes 'views' (comparable to the columns in the Zachman framework) and 'perspectives' (comparable to the rows in the Zachman framework), creating a matrix (comparable to the one suggested by Zachman). The TEAF matrix is presented in Figure 7.

Although this matrix contains fewer cells (16) than the Zachman framework (30), one is still not required to fill in all sixteen cells. Besides this, it is stated that to create a work product for one cell, information is needed from other views and sometimes from other perspectives (see below).

	Functional view	Information view	Organizational view	Infrastructure view
Planner	Mission & Vision Statements	Information Dictionary	Organization Chart	Technical Reference Model + Standards Profile
Owner	Activity Model + Information Assurance Trust Model	Information Exchange Matrix (Conceptual)	Node Connectivity Description (Conceptual)	Information Assurance Risk Assessment + System Interface Description Level 1
Designer	Business Process/System Function Matrix + Event Trace Diagrams + State Charts	Information Exchange Matrix (Logical) + Data CRUD Matrices + Logical Data Model	Node Connectivity Description (Logical)	System Interface Description Levels 2 & 3
Builder	System Functionality Description	Information Exchange Matrix (Physical) + Physical Data Model	Node Connectivity Description (Physical)	System Interface Description Level 4 + System Performance Parameters Matrix

Figure 7: The TEAF matrix: Views (columns), Perspectives (rows) and Work Products (in the cells) [14]

The four perspectives are the same as in the FEAF matrix (and the Zachman Framework), except that the TEAF collapses the 'Builder' and 'Subcontractor' perspectives into one perspective named 'Builder'. Also, three of the views correspond to the columns presented in our discussion concerning the FEAF: the FEAF Data Architecture corresponds to the TEAF Information view, the FEAF Applications Architecture corresponds to the TEAF Functional view, and the FEAF Technology Architecture corresponds to the TEAF Infrastructure view [14]. The TEAF Organization view is not present in the FEAF we presented above, but might be included in it later on. This view shows organizational structures, types of workers, etcetera. Any view may show components of the other views, but only as they relate to the view that is being described (i.e., the main interest area at that moment). A functional view model may for example include organizational and information components, but only as they relate to functions; and an information view model may show all functions, organizations, and infrastructures that use a particular piece of information. It is explicitly stated that '*many work products integrate information from other views (and sometimes other perspectives) than the view associated with the primary TEAF Matrix cell for the work product*'.

Please note that the TEAF allows bureaus to define own perspectives that can be mapped to the TEAF perspectives. Above this, any bureau may define additional views and perspectives that focus uniquely on areas important to stakeholders within that bureau or for external stakeholders.

As stated, the TEAF also defines a number of work products (the contents of the cells in Figure 7). The TEAF distinguishes 'essential work products' from 'supporting work products'. The meaning of the terms 'essential' and 'supporting' is slightly different from those in the C4ISR framework: essential products '*are required to be produced for an enterprise. These generally present the broadest perspective of the enterprise*'. As such, the products in the top two rows of the framework are essential. Supporting products '*generally provide more depth or more specialized perspectives of the enterprise*'. These models can be found in the designer and the builder rows of the matrix [14]. They are considered to be important to some organizations and not to others (this is in contrast to essential products, which are important to any enterprise architecture).

The TEAF also defines an enterprise architecture repository. This repository is meant to store, access, and manage all enterprise architecture information. Each bureau should establish a repository appropriate for its needs. The repository could be used to combine the products of diverse contributors, possibly using diverse tools, into a single, consistent model. A metamodel is used as the integrating structure of the repository. This metamodel defines the enterprise architecture element types and their interrelationships. In general, the architecture is built in accordance with a set of work product guidelines and templates, each of which describes a portion of the metamodel. In July 2000, no common department-wide format or mechanism for interchange of architecture information had been specified yet.

4 Our observations

The Zachman framework presents a number of perspectives related to the different participants in the system development process. It is clear there is a relationship between the 'architecture process' and the 'system development process'. Though a clear definition of this relationship is of fundamental importance, it is not very well documented in literature. One of the premises in our research is that it is very hard to manage something that is unclear. In our opinion, this is also true for the relationship between the system development and the architecture process. It is one of our goals to clearly specify the place of the architecture process in the plethora of processes companies are confronted with. The architecture process can only pay off if it is properly integrated with other processes, as *Enterprise Engineering and Program Management* and *Capital Planning and Investment Control (CPIC)*.

In our opinion, the Zachman framework is the most comprehensible and comprehensive framework of those presented. Most frameworks only present a small number of viewpoints and some do not mention whether they are dealing with the columns or with the rows of the Zachman framework. Above this, most papers do not get to the nitty-gritty of the frameworks, resulting in a less good definition of the diverse viewpoints. It is for example sometimes unclear whether the information-view should only show a model of the data (all the classes for example), or should also tell for each peace of data in which function it is used when and where. Also, different perspectives reveal different constraints, but it is not clear if constraints should for example only be propagated top-down in the Zachman framework, or also bottom-up, i.e., whether business models should be adapted if they are not realisable under the current technology. Besides this, most frameworks do not argue why the chosen views have been selected, and as such the soundness of the foundations of these frameworks is unclear. The only thing Kruchten [7] says about this is the following: '*Other sets of views*

[then mine] *have been proposed and discussed at our company and elsewhere, but we have found that proposed views can usually be folded into one of the four existing views. A cost and schedule view, for example, folds into the development view, and an execution view into a combination of the process and physical view.*' However, the power of architecture descriptions lays exactly in making the right abstractions, without folding views into other views.
The motivation and the ability to work correctly with a framework depend on the understanding of the framework and of the importance of all parts of the framework. It is thus very important to get a clear definition (at meta-level) of the contents of every cell. In our opinion, one should clearly define whether the focus is on the columns or on the rows of the Zachman-framework, or rather on a combination of both. If a combination of the columns and the rows is contemplated, it should be explicitly stated which rows and columns are being combined. Above this, the underpinnings of the framework should be made understandable.

One of the strengths of the Zachman framework is that it can serve as a classification scheme for information entities. We do, however, believe that this classification scheme is not adapted to the recent developments in the economy, i.e., that it does not provide a basis for classifying B2B integration initiatives. Where would public processes as identified by RosettaNet for example fit in? They would not, or at least not in a sensible way! Nevertheless, it is very important to know which B2B initiatives do exist, what is their role in the overall process, and how everything is related to one another (for example, how *private* processes relate to *public* processes) not only functionally, but also in terms of service levels. It seems to us that the Zachman framework is rather focussing on functional requirements than on non-functional ones (response times and the like). Service Level Agreements are, however, getting more important by the day, especially in a B2B context. We conclude that the Zachman framework is currently not ready for the new concepts produced by IT and economic evolutions.

In the C4ISR framework, much attention is paid to the set of products that constitute an architecture description. One of the key ideas in this framework is that the key to integratable architectures is not just in the viewpoint selection, but also in the creation of a common set of products. It is clear that, to be able to integrate ADs, it should at least be clear and unambiguously defined what is depicted in the models. It is currently unclear to us whether it is really necessary to demand the creation of a number of specific products: maybe it is sufficient if the descriptions follow some 'descriptive model'. Zachman for example identifies a descriptive model for every column in his framework. Archicture Markup Languages (AMLs) then could allow the transformation of models with a specific format (used in one company) into models with another format (used in another organization). Please note that the C4ISR essential products rather *seem* 'essential' to us because they narrow the view to specific (essential) matters, rather than because they are modelled in a specific way. Therefore, we may actually say that they implicitly define viewpoints. In our opinion, this phenomenon is the consequence of the fact that the C4ISR framework does not clearly mention which rows and columns of the Zachman framework are envisioned. The TEAF [14] mentions the following: '*The views and perspectives are convenient ways of organizing the various types of work products, but the most valuable comparison between any two architecture frameworks is in their respective types of work products and the data captured in them. Thus, two architecture descriptions built with different view and*

perspective structures (e.g., the FEAF views and perspectives vs. the TEAF views and perspectives) will still be mutually understandable and comparable if they consist of the same work products'[6]. In our opinion, it is evident that for two things to be comparable, they need to be looked at from the same point of view, and they need to be described in the same way. *The way* things are described, i.e., which format is used, is however not as fundamental as *what* is being described: it may become possible to automatically transform a model in one format into another model with another format, describing the same system from the same point of view (not from another point of view, as other information is needed for this).

Many of the classic enterprise architecture frameworks presented focus on the *software* architecture, rather than on the total enterprise architecture. We believe that the top two rows of the Zachman framework are neglected too often. Zachman [17] states that the sources of legacy frustration arise from fundamental architectural [description] deficiencies: rows 1, 2 and 3 (and even 4) models were seldom built, and the only application models that are left in many cases are those that are machine readable (often denoted as row 6). We should learn lessons from the past, and should consequently immediately document B2B efforts from different perspectives, before we hit management troubles again! Among these perspectives we should definitely find business perspectives. From the TEAF it is clear that the work products situated in the top two rows of the Zachman framework are essential when integration is the objective.

Acknowledgments

This paper has been written as part of the 'SAP-leerstoel'-project on 'Extended Enterprise Infrastructures' at the K.U.Leuven, sponsored by SAP Belgium.

References

[1] TOGAF (December 2002), The Open Group Architecture Framework (TOGAF), Version 8, Enterprise Edition, pp. 303.
[2] van de Heuvel W., Proper E (November, 2002), De pragmatiek van architectuur, *Informatie*, pp. 12-16.
[3] Maier M., The IEEE 1471-2000 Standard - Architecture Views and Viewpoints, www.opengroup.org/architecture/togaf/agenda/ 0107aust/presents/maier_1471.pdf
[4] Hilliard R., Impact assessment of IEEE 1471 on The Open Group Architecture Framework, www.opengroup.org/architecture/togaf7/ procs/p1471-togaf-impact.pdf (visited on 29/1/2003).
[5] Zachman J. (1987), A framework for information systems architecture, *IBM Systems Journal*, Vol. 26, No.3, pp. 276-292.
[6] Sowa J., Zachman J. (1992), Extending and formalizing the framework for information systems architecture, *IBM Systems Journal*, Vol. 31, No. 3, pp. 590-616.
[7] Kruchten P. (November 1995), The 4+1 View Model of Architecture, *IEEE Software*, pp. 42-50.

[6] Please note that many of the TEAF work products are based on those of the C4ISR framework.

[8] Soni, D., R.L. Nord & C. Hofmeister, 'Software architecture in industrial applications', in: R. Jeffrey, D. Notkin (eds.), *Proceedings of the 17th International Conference on Software Engineering*, ACM Press, 1995, pp. 196-207.
[9] Tapscott D., Caston A. (1993), The New Promise of Information Technology, McGraw-Hill, pp. 313.
[10] OMG Architecture Board ORMSC (July 2001), Model Driven Architecture (MDA), ormsc/2001-07-01 pp. 31.
[11] The Open Group, Architecture Description Markup Language (ADML), http://www.opengroup.org/architecture/adml/adml_home.htm
[12] The Chief Information Officers Council (September 1999), Federal Enterprise Architecture Framework Version 1.1, pp. 41.
[13] Department of Defense - C4ISR Architectures Working Group, (December 1997), C4ISR Architecture Framework Version 2.0, pp. 239. http://www.c3i.osd.mil/org/cio/i3/AWG_Digital_Library/
[14] Department of the Treasury, Chief Information Officer Council, Treasury Enterprise Architecture Framework, Version 1, pp. 164. http://ustreasury.mondosearch.com/cgi-bin/MsmGo.exe?grab_id=49270800&EXTRA_ARG=IMAGE%2EX%3D42%00%26IMAGE%2EY%3D13&host_id=1&page_id=4177&query=teaf
[15] Frankel, D., Parodi, J.: Using Model-Driven Architecture to Develop Web Services, IONA Technologies white paper (2002)
[16] Olanders K., Henson J., Carrico T., LMI Enterprise Architecture Practice, An Analytical Framework, http://www.lmi.org/Services/FMS/FMSHOME/LEAP.htm
[17] Zachman J. Enterprise Architecture and Legacy Systems, Getting Beyond the "Legacy", http://members.ozemail.com.au/~visible/papers/zachman1.htm
[18] Zachman J., Information Systems Architecture, http://aiken.isy.vcu.edu/projects/dbna/kw/contents/framework/explainei/isaintro.html
[19] Ambler S. (2002), Architecture and Architecture Modeling Techniques, http://www.agiledata.org/essays/enterpriseArchitectureTechniques.html (visited on 29/1/2003).
[20] Lassing N., Rijsenbrij D., van Vliet H. (September 2001), Zicht op aanpasbaarheid, *Informatie*, pp. 30-36.
[21] Clements P., Kazman R., Klein M. (2002), *Evaluating Software Architectures – Methods and Case Studies*, Addison-Wesley, pp. 322.
[22] Perry D., Wolf A. (October 1992), Foundations for the Study of Software Architecture, *ACM Software Eng. Notes*, pp. 40-52.
[23] Farooqui K., Logrippo L., de Meer J. (February 1996), The ISO Reference Model for Open Distributed Processing – An Introduction, http://lotos.site.uottawa.ca/ftp/pub/Lotos/TechRep/CNIS-94.pdf