

# Frameworks for Model-based Analysis and Design of Enterprise Information Systems

Mara Nikolaidou<sup>1</sup>, Nancy Alexopoulou<sup>12</sup>  
{[mara@di.uoa.gr](mailto:mara@di.uoa.gr), [nancy@hua.gr](mailto:nancy@hua.gr)}

<sup>1</sup> Harokopio University of Athens,  
El. Venizelou Str, 17671 Athens, Greece

<sup>2</sup> Department of Informatics and Telecommunications,  
University of Athens, Panepistimiopolis, 15771, Athens, Greece

## **1. Introduction**

When building an enterprise information system, the desired properties of the system should be defined, such as its structure and behavior, while the role of the system in its environment should also be considered. Many different stakeholders may be involved in this process, each of which focusing on certain concerns and considering these concerns at a certain level of detail. Therefore, various methodologies and frameworks have been developed aiming at a consistent development and configuration of enterprise information systems. Most of them have adopted the notion of *separating concerns* by establishing different viewpoints, each depicting the concerns of a specific stakeholder (e.g. user, designer, implementer, etc.). Following, we focus on the system designer viewpoint, exploring issues related to the analysis and design of Enterprise Information Systems (EIS).

*System engineering* is the process of analyzing system requirements, designing the desired architecture of a system and exploring performance requirements, ensuring, thus, that all system components are identified and properly allocated and that system resources can provide the desired performance. It corresponds to the system designer viewpoint. Although, vendors (as IBM or Oracle) actively promote information system development based on multi-tiered architectures, the proposed solutions, although expensive, often fail to provide the desired performance. This is due to the fact that system designers often neglect system engineering issues contributing to the overall application performance. In spite of design issues being interrelated, they are often modeled and studied in isolation, resulting in poor system performance. In practice, discrete issues, as network architecture description or resource allocation are supported by autonomous automated or semi-automated tools, each of which adopts its own metamodel for system representation. Thus, no interaction between them is supported. To effectively explore EIS engineering, heterogeneous tools and system models should be integrated. This integration could be accomplished by adopting *model-based system engineering* (Oliver et al., 1997). Model-based system engineering (MBSE) provides a central system model (tool-independent) that captures system requirements and design decisions that fulfill them at different levels of abstraction. It enables integration of system models supported by autonomous design tools and interoperability between them without interfering with their internal implementation. Model-based system engineering has already demonstrated a positive impact on

large-scale systems. Thus, we argue that it is best suited for enterprise information system analysis and design. When applying model-based system engineering, a multi-level, technology-neutral model for EIS representation should be defined, taking into account different aspects of the system, such as network architecture, resource allocation, application execution requirements, etc, involved in system design.

Existing well-known frameworks may be used for system modeling. The Open Distributed Processing Reference Model (RM-ODP) is such a framework, dealing with aspects related to the distribution, interoperation and portability of distributed information systems. Another widely referenced framework is the *enterprise architecture framework* defined by Zachman, which specifies the development process of enterprise information systems, starting from the identification of the enterprise's business objectives and resulting in a detailed system implementation. Independently of the framework used, the different system views defined from each viewpoint can be effectively depicted through models. Models may be expressed using various modeling languages. However, the most popular and widely adopted modeling language for the representation of models is the Unified Modeling Language (UML). Numerous designers use the extension mechanisms provided by UML to create profiles (i.e. specializations of UML diagrams) to better serve their modeling purposes.

The main focus of this chapter is the exploration of model-based analysis and design requirements for enterprise information systems. Three alternative approaches for model-based EIS engineering are discussed, based on the above requirements. All of them adopt UML as the modeling language for EIS representation. These are: a) the RUP system engineering approach, b) the UML4ODP proposed standard with emphasis on Engineering Viewpoint and c) the EIS Engineering Framework proposed by the authors.

The rest of the chapter is organized as follows: In section 2, background information regarding model-based system engineering and system viewpoints is provided. Also, Zachman's framework and RM-ODP are briefly discussed and their relevance to the proposed approach is presented. In section 3, the requirements for successful model-based engineering for enterprise information systems are presented. In sections 4-6, the three alternative approaches are analysed and discussed based on these requirements. Future trends and conclusions reside in the last two sections.

## **2. Background**

This section describes the main principles of model-based system engineering (MBSE) and its advantages for enterprise information system analysis and design. System modeling is a critical issue in MBSE. The IEEE Std 1471, which provides guidelines for the description of systems from different perspectives (viewpoints), is considered as the basis for the definition of a central system model for MBSE. An overview of existing frameworks for EIS modeling can be found in (Goethals et al., 2006) and (Leist, 2006). We discuss two of them, namely the Zachman framework and RM-ODP, which support the designer perspective, are well-known and come from different origins. It should be noted, that although defined prior to IEEE 1471 standard, they both adopt the concepts of views and viewpoints.

### **2.1 Model-Based System Engineering**

System modeling constitutes an important part of system engineering, since it may facilitate the complete description of all aspects involved and contribute to the effectiveness of the whole process. How many different system models should be supported? Should all of them provide the same level of detail? How can the correspondences between different models be identified and ensured? Since discrete design issues are usually resolved by different methodologies and autonomous software tools, the support of different system models cannot be avoided. In many cases, these models are not compatible, thus, design issues, although interrelated, are often solved in isolation. Even if a certain design problem, for example network architecture, is optimized, there is no guarantee that the overall EIS architecture will be optimized as well. To resolve such a situation, a central, tool-independent model should be adopted.

Model-driven technologies for application development, such as Model Driven Architecture (MDA) (Brown, 2004), proposed by OMG, enable the definition of *platform-independent models* (PIMs) for the specification of system functionalities and *platform-specific models* (PSMs) for the specification of the implementation of these functionalities on a particular technological platform and the definition of couplings between PIMs and PSMs. Modeling languages, methods and tools have been established to support model-driven software development. In a similar fashion, model-based system engineering (MBSE) provides a central system model (corresponding to a PIM) that captures, at different levels of abstraction, system requirements and design decisions that fulfill them. In addition, tool-specific models could be defined (corresponding to PSMs), while MBSE also provides for model transformation (couplings between PIM and PSMs). Thus, the interoperability between models and methods corresponding to discrete design issues is achieved, without interfering with their internal implementation in the respective software tools. The central system model serves all engineering activities, for example it could be executed by a simulator to validate design decisions.

The Unified Modeling Language (UML) is a modeling language attempting to standardize graphical language elements for modeling software systems. It is a well-known software engineering standard, since most software developers are familiar with it, while there is a lot of activity in advancing both the UML supported functionality and the UML tools. Numerous designers use the extension mechanisms provided by UML to create profiles (i.e. specializations of UML diagrams) to better serve their modeling purposes. UML 2.0 (OMG, 2007) consists of thirteen diagram types used for structural, behavioral and interaction modeling. Many diagram types, such as use-case, state, activity, can be used for general functional requirement analysis. Evidently, UML is adopted in MBSE as well, serving as a common enterprise notation language, while UML extensions have been proposed for system engineering (Murray, 2003a) (Nikolaidou et al., 2006).

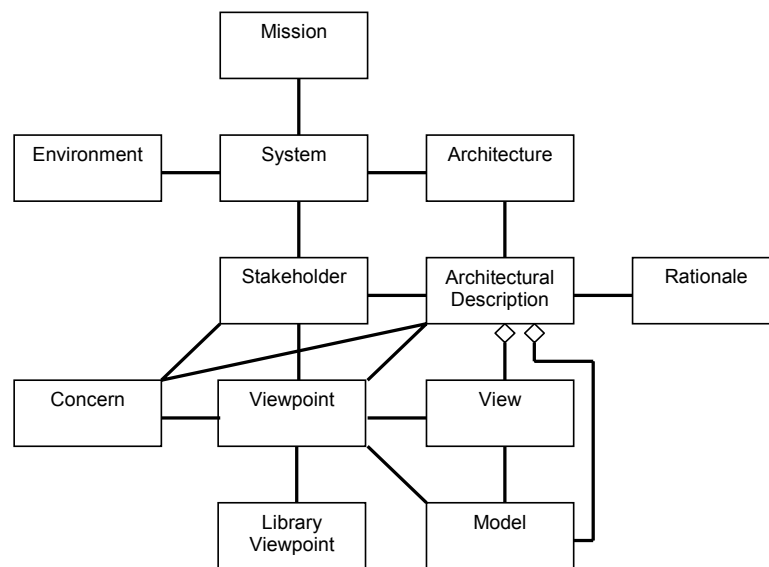
As it will be elucidated in this chapter, MBSE is appropriate for enterprise information system analysis and design. When applying MBSE, a multi-level, technology-neutral model for EIS representation should be defined, taking into account different aspects of the system, such as network architecture, resource allocation, application execution requirements, etc, involved in system design. Existing modeling frameworks are explored for this purpose in the following paragraphs. Independently of the framework used, we suggest the UML should be adopted for model representation.

## **2.2. Defining Views and Viewpoints for Enterprise Information System Architecture**

An important milestone in the field of enterprise system architecture descriptions is ANSI/IEEE Std

1471 - Recommended Practice for Architectural Description of Software-Intensive Systems (IEEE1471). It defines enterprise system concepts and their relationships that are relevant for architectural description, thus provides a standard way of defining EIS architecture models. It also provides guidance on the structure of architectural descriptions.

The main concepts standardised are *architecture*, *architectural description*, *concern*, *stakeholder*, *viewpoint* and *view*. *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”. *Architecture Description* is “a collection of artifacts documenting the architecture”. *Stakeholders* are “people with key roles or concerns about the system”, while *concerns* are “the key interests crucially important to the stakeholders and determine the acceptability of the system from stakeholder specific perspective”. *Views* are “representations of the whole system from the perspective of a related set of concerns”, while *viewpoints* define “the perspective from which a view is taken”. The main concepts of IEEE 1471 standard and their interrelations are depicted in figure 1.



**Figure 1: IEEE/ANSI Std 1471 conceptual model**

A viewpoint defines: a) how to construct and use a view, b) the information that should appear in the view, c) the modelling techniques for expressing and analyzing the information and d) a rationale for these choices (by describing the purpose and intended audience of the view). Different stakeholders with different roles in the system have different concerns, which are expressed through different viewpoints. Each view is a capture of the representation of the system architecture design, typically comprising of one or more architecture *models*. In simple words, a view is what you see, while a viewpoint is where you are looking from – the vantage point or perspective which determines what you see. Viewpoints are generic, while a view is always specific to the architecture for which it is created. To successfully define an architecture description, specific characteristics should be obtained (Hilliard, 2001):

- Views should be modular. A view may consist of one or more architectural models.

- Views should be well-formed. Each view has an underlying viewpoint specifying view definition using a formal method, as languages, notations, models and analytical techniques.
- View consistency should be ensured. Viewpoints may also include any consistency or completeness checks associated with the underlying method to be applied to models within the view; any evaluation or analysis techniques to be applied to models within the view; and any heuristics, patterns, or other guidelines which aid in the synthesis of an associated view or its models.

Although not defined in IEEE 1471, additional issues should be addressed, such as (Hilliard, 2001):

- View integration and inter-view consistency. It has been long recognized that introducing multiple views into architectural descriptions leads to an integration problem. How does one keep views consistent and non-overlapping? The introduction of viewpoint declarations, while not solving the problem, gives us a tool for detecting overlaps and inconsistencies, and potentially a substrate for solving the integration problem.
- Formalization. The conceptual framework of IEEE 1471 is an informal, qualitative model. If it is useful, which appears to be the case, it may be insightful to attempt to formalize the concepts therein. Such a formalization could have benefits in several topics, as view checking, view integration, and inter-view analysis.

Since its publication in 2000, IEEE 1471 has received much appraisal. The concepts of stakeholders, concerns and views are accepted as essential. The terminology proposed by IEEE 1471 is now being used by many architects. The focus on concerns of stakeholders is a good stimulus for otherwise possibly too technically oriented IT architects. After all, it is the interests of the stakeholders that need to be served.

IEEE 1471 proposes a formal method to define system architectures, but it does not propose nor prescribe any specific viewpoint for system architects and stakeholders (Greefhorst et al., 2006). However, it can be used as a guide to define viewpoints and views for EIS model-based analysis and design, as discussed in the following sections. Inter-view consistency and formal description is the focus of our concern. As already mentioned, each view may be formally defined by a *model*, while it should also be communicated to the stakeholder by a *representation model*, which is a concrete representation of the system view on some medium (e.g. paper or computer program) (Boer et al., 2004). The aforementioned definitions are adopted throughout the rest of the chapter.







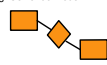
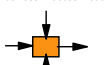
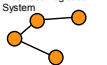
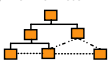

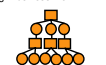
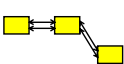
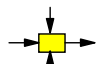
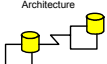
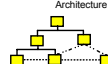

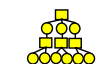












The attainment of a consistent representation of the systems entails that view interrelations must be typically defined. In order to formally define a viewpoint, one should define a metamodel describing the supported views independently of the modeling language used for system representation and then define the representation model. In this way, a view may be represented using different languages, such as UML or ISDL, in a common manner, facilitating thus the transformation between representation modeling languages. As indicated in (Dijkman et al., 2003), two basic relations are identified between views: *refinement* (the internal view refines the external view on a different level of detail) and *complement* (two views may complement each other by considering complementary concerns).

### 2.3. Zachman Framework

Enterprise information systems can be described based on the Zachman framework. The widely

referenced Enterprise Architecture framework of Zachman (Zachman, 1999), simply referred as the “Zachman Framework” is a logical structure for organizing and classifying the artifacts created during the development of enterprise information systems. The purpose of the framework is to ensure the establishment of enterprise information systems starting from the identification of the enterprise’s business objectives, as a typical problem of modern enterprises is the time-consuming and costly implementation of information systems that often fail to meet business objectives.

## ENTERPRISE ARCHITECTURE - A FRAMEWORK <sup>TM</sup>

	DATA <i>What</i>	FUNCTION <i>How</i>	NETWORK <i>Where</i>	PEOPLE <i>Who</i>	TIME <i>When</i>	MOTIVATION <i>Why</i>	
SCOPE (CONTEXTUAL)  <i>Planner</i>	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 Significant to the Business 	List of Business Goals/Strat Critical Success Factor 	SCOPE (CONTEXTUAL)  <i>Planner</i>
ENTERPRISE MODEL (CONCEPTUAL)  <i>Owner</i>	e.g. Semantic Model  Ent = Business Entity Rein = Business Relationship	e.g. Business Process Model  Proc = Business Process I/O = Business Resources	e.g. Business Logistics System  Node = Business Location Link = Business Linkage	e.g. Work Flow Model  People = Organization Unit Work = Work Product	e.g. Master Schedule  Time = Business Event Cycle = Business Cycle	e.g. Business Plan  End = Business Objective Means = Business Strategy	ENTERPRISE MODEL (CONCEPTUAL)  <i>Owner</i>
SYSTEM MODEL (LOGICAL)  <i>Designer</i>	e.g. Logical Data Model  Ent = Data Entity Rein = Data Relationship	e.g. Application Architecture  Proc = Application Function I/O = User Views	e.g. Distributed System Architecture  Node = I/S Function (Processor, Storage, etc.) Link = Line Characteristics	e.g. Human Interface Architecture  People = Role Work = Deliverable	e.g. Processing Structure  Time = System Event Cycle = Processing Cycle	e.g. Business Rule Model  End = Structural Assertion Means = Action Assertion	SYSTEM MODEL (LOGICAL)  <i>Designer</i>
TECHNOLOGY MODEL (PHYSICAL)  <i>Builder</i>	e.g. Physical Data Model  Ent = Segment/Table/etc. Rein = Pointer/Key/etc.	e.g. System Design  Proc = Computer Function I/O = Data Elements/Sets	e.g. Technology Architecture  Node = Hardware/System Software Link = Line Specifications	e.g. Presentation Architecture  People = User Work = Screen Format	e.g. Control Structure  Time = Execute Cycle = Component Cycle	e.g. Rule Design  End = Condition Means = Action	TECHNOLOGY MODEL (PHYSICAL)  <i>Builder</i>
DETAILED REPRESENTATIONS (OUT-OF-CONTEXT)  <i>Sub-Contractor</i>	e.g. Data Definition  Ent = Field Rein = Address	e.g. Program  Proc = Language Stmt I/O = Control Block	e.g. Network Architecture  Node = Addresses Link = Protocols	e.g. Security Architecture  People = Identity Work = Job	e.g. Timing Definition  Time = Interrupt Cycle = Machine Cycle	e.g. Rule Specification  End = Sub-condition Means = Step	DETAILED REPRESENTATIONS (OUT-OF-CONTEXT)  <i>Sub-Contractor</i>
FUNCTIONING ENTERPRISE	e.g. DATA	e.g. FUNCTION	e.g. NETWORK	e.g. ORGANIZATION	e.g. SCHEDULE	e.g. STRATEGY	FUNCTIONING ENTERPRISE

**Figure 2: Overview of the Zachman Framework**

The Zachman’s framework is deployed in two dimensions. The first dimension addresses the different perspectives of the stakeholders participating in information system development. These perspectives derive from the parallelism of information system development with the construction of a building. As such, Zachman defines the *Owner’s*, *Designer’s* and *Builder’s* viewpoints. The first viewpoint, defined by the *business model*, is a description of the enterprise within which the information system will function. The second delineates how the system will satisfy the requirements ensuing from the business objectives, yielding the *system model*. The third viewpoint represents how the system will be implemented, providing the *builder model*. To produce a comprehensive framework for enterprise information system development, Zachman has added three more viewpoints, namely *Scope* which denotes the business purpose and strategy defining the context for the other viewpoints, the *Out-of-Context* which includes implementation-specific details, and the *Operational*, which is the functioning system. The second dimension distinguishes different focal points of the system. The *Data* aspect describes what entities are involved, while the *Function* aspect shows how the entities are processed. The *Network* perspective indicates where the entities are located. Apart from the *what*, *how* and *where*, the framework addresses also three other questions, specifically *who*, *when* and *why*. As such, it defines the *People* who work with the system, when events occur (*Time* aspect) and why these activities take place (*Motivation* aspect). The

combination of the two dimensions in a matrix, with the focal points indicated by the columns and the different perspectives by the rows, yields the Zachman's framework as presented in figure 2.

Each cell constitutes a separate view. As such, an organization should create a wide range of diagrams and documents representing the different views defined within the Zachman framework. As shown in figure 2, the Zachman framework contains suggested specification models for each view (e.g. using ER technique for modeling the data description in the owner's viewpoint or using functional flow diagrams for modeling the process description in the owner's viewpoint). However, the Zachman framework doesn't suggest a specific methodology or technique for the description of view models. Moreover it does not typically define a metamodel to integrate the information of all cells nor does it describe a way to trace information between cells (Frankel et al., 2003). Its objective is to provide some basic principles that should guide the implementation of enterprise information systems. As such, it says nothing about the development of conformant views or the order that should be developed. The strength of the framework is that it provides an organized way of thinking about an enterprise, in respect to information systems, so that it can be described and analyzed. It enables the individuals involved in producing enterprise information systems to focus on selected aspects of the system without losing sight of the overall enterprise context. Moreover, it facilitates them to find out possible gaps and inconsistencies between view representations and thus modify the models appropriately to eliminate all inconsistencies.

EIS engineering issues are obviously addressed in the *System Model* row of the Zachman's matrix. The system designer may actually work concurrently with the system developer (the builder of the model), although system design is usually performed prior to its implementation. As already stated, the Zachman framework does not specify whether these two stages must be performed sequentially or in parallel. In many cases, during system design, although system architecture is defined and the services provided by the distributed applications are identified, detail software design and implementation is considered in the builder model. In practice, system engineering issues can be dealt with independently of the status of software development process. Thus, following we will focus on the System Model row of the Zachman's matrix.

Lastly, it should be noted that while a plethora of methodologies and formalisms exist, each applicable to some subset of cells, Zachman however encourages a single common language to describe the subject of all the cells as well as their interrelationships, rather than using a specialized notation for each view separately (Sowa & Zachman, 1992).

## 2.4. RM ODP

As enterprise information systems are distributed, they can alternatively be described by the Reference Model of Open Distributed Processing (RM-ODP). The Open Distributed Processing Reference Model (RM-ODP) is a conceptual framework established by ISO (ISO/IEC, 1998) for the specification of large-scale distributed systems. RM-ODP integrates aspects related to the distribution, interoperation and portability of distributed systems in such a way that network/hardware infrastructure is transparent to the user. RM-ODP manages system internal complexity through the *separation of concerns*, addressing specific problems dealt with during system development from different viewpoints (ISO/IEC, 1998). It provides an object-oriented representation of the system, while it is highly technical, relatively complex and focuses on distributed application development. RM-ODP manages system internal complexity through the identification of five generic and complementary viewpoints which are as follows:

- *Enterprise* viewpoint, which concentrates on the business activities of the specified system.

- *Information* viewpoint, which focuses on the information that needs to be stored and processed in the system.
- *Computational* viewpoint, which describes system functionality through functional decomposition of the system into components that interact via interfaces.
- *Engineering* viewpoint, which examines the mechanisms and functions required to support distributed interactions between components.
- *Technology* viewpoint, which focuses on the choice of technology for system implementation.

For each viewpoint there is an associated viewpoint language which can be used to express a specification of the system from that viewpoint. The object modeling concepts give a common basis for the viewpoint languages and make it possible to identify relationships between the different viewpoint specifications and to assert correspondences between the representations of the system in different viewpoints. Viewpoint languages provide the means for the detailed description of systems according to the viewpoint perspective. System views are formally defined based on the corresponding viewpoint languages.

System engineering issues are addressed in RM-ODP *Engineering Viewpoint*. The engineering language focuses on the way system component interaction is achieved and on the resources needed to do so. In the engineering language, the main concern is the support of interactions between computational objects, defined in the computational view to represent a service or a program operating in the distributed platform. As a consequence, there are very direct links between the viewpoint descriptions; computational objects are visible in the engineering viewpoint as *basic engineering objects*, representing the actual implementation of computational objects. Engineering objects are physically located and associated with processing resources by grouping them into *nodes*, which can be thought of as representing independently managed computing systems. A *cluster* is a grouping of *basic engineering objects*, used for resource allocation purposes (all objects in a cluster are manipulated as a single entity). Clusters form *capsules* (a single entity for the purpose of resource allocation and protection). *Capsules* are associated to *nuclei* which are responsible for making communications and processing facilities available (to capture the notion of a virtual machine). When engineering objects in different clusters interact, mechanisms are needed to cope with it. The set of mechanisms needed to do this constitute a *channel* (represents client-server communication), which is made up of a number of interacting engineering objects: *Stubs* are concerned with the information conveyed in an interaction, *binders* are concerned with maintaining the association between the set of basic engineering objects linked by the channel, and *protocol objects* manage the actual communication. Basic engineering entities and their interrelations are depicted in figure 3.

The concepts defined to describe system architecture (as clusters, capsules or nuclei) are complex ones, while they are not adopted by system designers in their every-day work, thus they cannot be instantly related to them. Network architecture is described in great detail using client-server concepts, while the description of systems entities (for example communication channels) might be too detailed. Alternative architectural approaches should be easily described within Engineering Viewpoint to enhance its acceptance by system designers. Furthermore, the dependencies between viewpoints although identified, are not formally enforced.

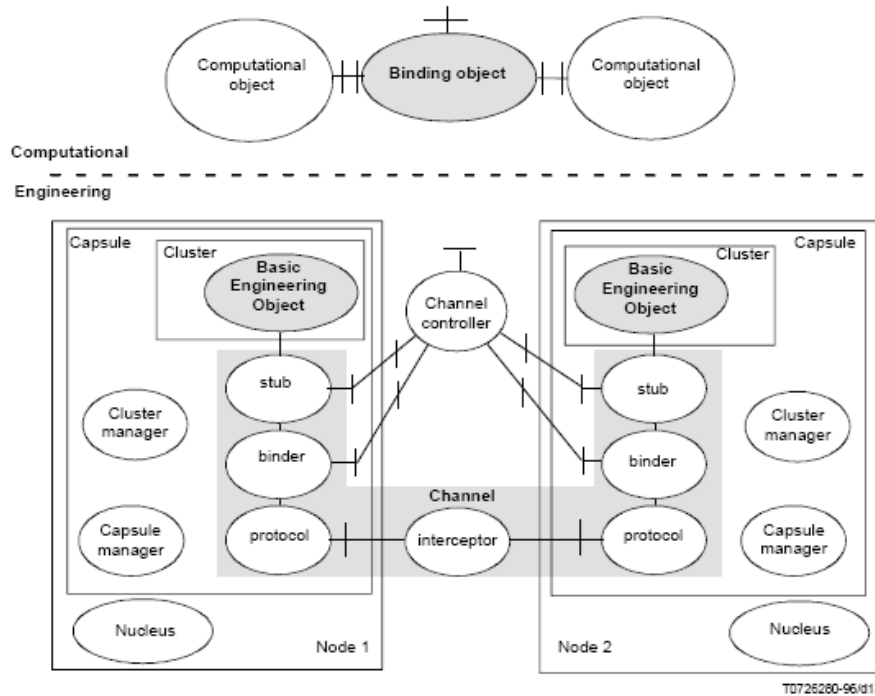
Regarding system engineering, within Engineering Viewpoint the following aspects are clarified:

- A system-oriented view of distributed applications
- System access points



- The distributed platform infrastructure (e.g. network architecture and hardware configuration)
- The association of software components to network nodes (resource allocation), in order to ensure performance requirements.

However, the means of actually performing resource allocation are not provided, since performance requirements are not depicted within Engineering Viewpoint.



**Figure 3: Basic engineering entities and their interrelations (ISO/IEC, 1998)**

Frankel (Frankel, 2003) suggests separating the Engineering Viewpoint into two discrete sub-viewpoints, the *logical* and the *deployment* one. The deployment one focuses on a technology-independent description of the network architecture and hardware configuration. The logical one corresponds to the description of distributed application architecture and the policies adopted for the operation (e.g. replication policy). This separation helps in clarifying the dependencies between application requirements and distributed platform infrastructure.

### **3. Model-based EIS Analysis and Design**

Modern enterprise information systems are based on distributed architectures, consisting of a combination of Intranet and Internet web-based applications. They are built on multi-tiered client-server models (Serain, 1999), as the J2EE architecture. Such platforms distinguish application logic from the user-interface and contribute to system configurability and extensibility. Despite the fact that vendors (such as IBM and Oracle) actively promote information system development using the aforementioned architectures, the proposed solutions, although expensive, often fail to provide the desired performance (Savino-Vázquez et al., 2000). This is due to the fact that design issues,

although interrelated, are solved in isolation, while the internal complexity of applications is neglected when estimating the quality of service (QoS) imposed to the network supporting them.

In practice, discrete issues, as network architecture description or resource allocation are supported by autonomous automated or semi-automated tools (Gomaa et al., 1996, Graupner et al., 2001, Nezelek et al., 1999). Each of these tools supports its own representation metamodel (for example queuing networks, Petri-nets, objects), while different system properties are depicted in them. The existence of a common metamodel describing all EIS properties is of great importance for the efficient requirement analysis and design of such systems, since it would facilitate the communication between autonomous design stages/tools and act as a “reference point”. Thus, a model-based approach for EIS analysis and design is considered most efficient.

In order to provide an integrated framework for model-based enterprise information system engineering the following requirements should be addressed:

- *Definition of a common, multi-layered, platform-independent model of EIS architecture.* EIS architecture description should follow IEEE Std 1471, thus EIS model definition should consist of well defined views and viewpoints. Each view should address a discrete design issue and should be formally defined. Furthermore, view and inter-view consistency should be well-established, since the main reason for adopting model-based design is to ensure integration of discrete design issues/tools. Lastly, compatibility of the proposed model with Zachman System Model or RM-ODP engineering viewpoint should also be supported.
- *Covering basic EIS architecture design issues* (as defined in both Zachman’s System Model and RM-ODP Engineering viewpoint). These are: a) definition of EIS architecture (e.g. a system-oriented view of distributed applications), indicating system performance requirements, b) definition of system access points, c) description of platform-independent distributed infrastructure (e.g. network architecture and hardware configuration) and d) mapping of software components to network nodes (resource allocation), in order to ensure performance requirements.
- *Description of a methodology for EIS architecture design.* This could be part of the viewpoints defined or independent of them. Thus, it could be applied at different levels of detail, facilitating the progressive definition of system architecture.
- *Definition of a UML representation model for EIS architecture.* It should provide for an integrated, easy-to-use interface for system designer.
- *Tool integration - Model exchangeability.* Since discrete design issues may be resolved using autonomous tools, heterogeneous tool integration should be supported. Most of them employ their own internal model for EIS representation. Thus, tool coordination and internal metamodel transformation should also be supported. According to model-based design principles, consistency is ensured, since the common metamodel acts as a “reference point”. Prior to using an existing tool, the partial transformation of the common metamodel (platform-independent) into the tool’s internal metamodel (platform-dependent) must be facilitated. Using this transformation, the invocation and initialization of any tool can be automatically performed. Input/output parameters must be represented in the common metamodel. Their values could be either entered by the system designer or automatically computed by the tool.

Following, we discuss three alternative approaches for model-based EIS engineering with regard to the above requirements. All of them adopt UML as the modeling language for EIS representation. These are: a) the RUP system engineering approach (Murray, 2003a) (Murray 2003b), b) the UML4ODP proposed standard with emphasis on Engineering Viewpoint (ISO/IEC, 2006) and c) EIS Engineering Framework proposed by the authors.

#### **4. RUP System Engineering**

Rational Unified Process for Systems Engineering (RUP SE) is a framework developed by Rational (Murray, 2003a) to address system engineering issues in conjunction with RUP methodology for software engineering. RUP SE adopts all the modeling concepts and perspectives of RUP and is fully compatible with it. The purpose of RUP SE is to support teams of system engineers as they determine the black box view of the system (e.g. the system as a whole, that is the services it provides and the requirements it meets) and specify an optimal white box system design (e.g. elements or parts that make up the system) that meets all stakeholder needs. In particular, RUP SE comprises:

1. an architecture framework, which describes the internals of a system (architectural elements) from multiple viewpoints
2. a set of UML-based artifacts for system architecture modeling
3. a methodology, called *use-case flowdown* (Murray, 2003c), for deriving requirements for architectural elements.

##### **4.1. RUP SE System Architecture Modeling Framework**

The RUP SE system architecture framework is deployed in two dimensions (Brown & Densmore, 2005), as shown in Table 1. The first dimension defines a set of viewpoints that represent different areas of concern that must be addressed in the system architecture and design. Analytically, *Worker* viewpoint expresses roles and responsibilities of system workers regarding the delivery of system services. *Logical* viewpoint concerns the logical decomposition of the system into a coherent set of UML subsystems that collaborate to provide the desired behavior. *Physical* viewpoint regards the physical decomposition of the system and specification of physical components. *Information* viewpoint focuses on the information stored and processed by the system. *Process* viewpoint examines the threads of control that carry out the computation elements. Lastly, *Geometric* viewpoint denotes the spatial relationship between physical components.

In addition to viewpoints, building system architecture requires levels of specification as the architecture is being developed. There are four model levels defined in RUP SE, consistent to RUP. As shown in Table 1, these constitute the second dimension of the RUP SE architecture framework. The *Context* level treats the entire system as a single entity: a black box. It does not address the system's internal elements. At the *Analysis* level, the system's internal elements are defined (white box approach), describing domain elements at a relatively high level. These elements vary, depending on the specific viewpoint. For example, in the Logical viewpoint, *subsystems* are defined to represent abstract, high-level elements of functionality. Less abstract elements are represented as *sub-subsystems* or *classes*. In the Physical viewpoint, *localities* are defined to represent the places in which functionality is distributed. The *Design* level is where design decisions that will drive the implementation are captured. The *Implementation* level concerns decisions about technology

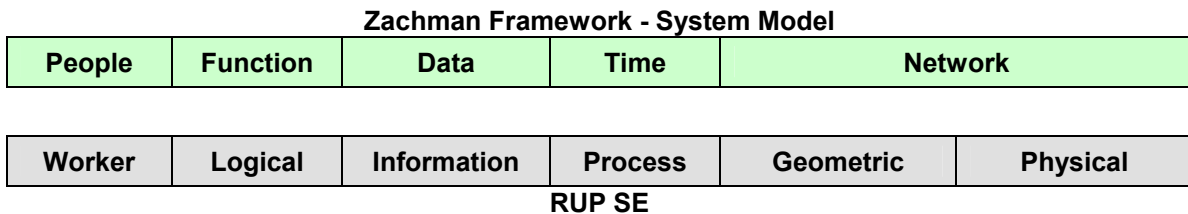
choices for implementation. The intersection of model level rows with the viewpoint columns yields the different views of a system lifecycle. As shown in Table 1, each view comprises different model elements. It should be noted that RUP SE does not dictate that all system development efforts require every viewpoint. The introduced viewpoints are a mechanism to address different stakeholders' concerns but also to maintain an integrated, consistent representation of the overall system design.

	VIEWPOINTS					
MODEL LEVELS	Worker	Logical	Information	Physical	Process	Geometric
Context	UML Organization diagram	UML System Context Diagram UML Use Case Diagram Specification	UML Enterprise Data View Containing Extended Product Data	UML Enterprise Locality View	UML Business Processes diagram	Domain-dependent Views
Analysis	UML Partitioning of System into Human Machine	UML System Logical Decomposition Diagram	Product Data Conceptual Schema	UML System Locality View	UML Process View	Parameterized Geometric Model Layouts
Design	UML System Worker View	UML Software Component Design	Product Data Schema	UML Descriptor Node View	UML Detailed Process View and Timing diagrams	MCAD Design
Implementation	Hardware and Software Configuration					

**Table 1: The RUP SE Architecture Framework (Murray, 2003b, Brown & Densmore, 2005)**

One could identify a correspondence between RUP SE and Zachman's viewpoints, while we consider that Context, Analysis and Design model level could be incorporated within the System Model row of Zachman's matrix. The Context model level, in particular, may constitute the bridge to the upper Zachman row (Business Model) and the Design Model to the lower (Technology Model).

As presented in figure 4, all system model aspects of Zachman framework, except for the motivation which is not examined within RUP SE, are covered by the corresponding RUP SE viewpoints. RUP SE defines 18 different views corresponding to the System Model row of Zachman framework, which could be a bit confusing for the system designer. Furthermore, there is no formal definition of the models corresponding to each view, although the purpose and functionality of each of them is clearly defined, as stated in IEEE Std 1471.



**Figure 4: Mapping RUP SE to Zachman Framework**

## 4.2. UML Representation Model

RUP SE employs UML 1.4 to create system artifacts for each view specified in the architecture framework. Each viewpoint is described using specific collaborating entities through context, analysis and design levels. The use of UML for both object and relational database modeling is a well-developed practice that RUP SE makes use of in the information viewpoint, thus no stereotypes were defined. The process viewpoint is represented as collaborating processes, using standard UML semantics (for example activity diagrams). The same is applied to geometric viewpoint as well, described as collaborating components (standard component diagrams).

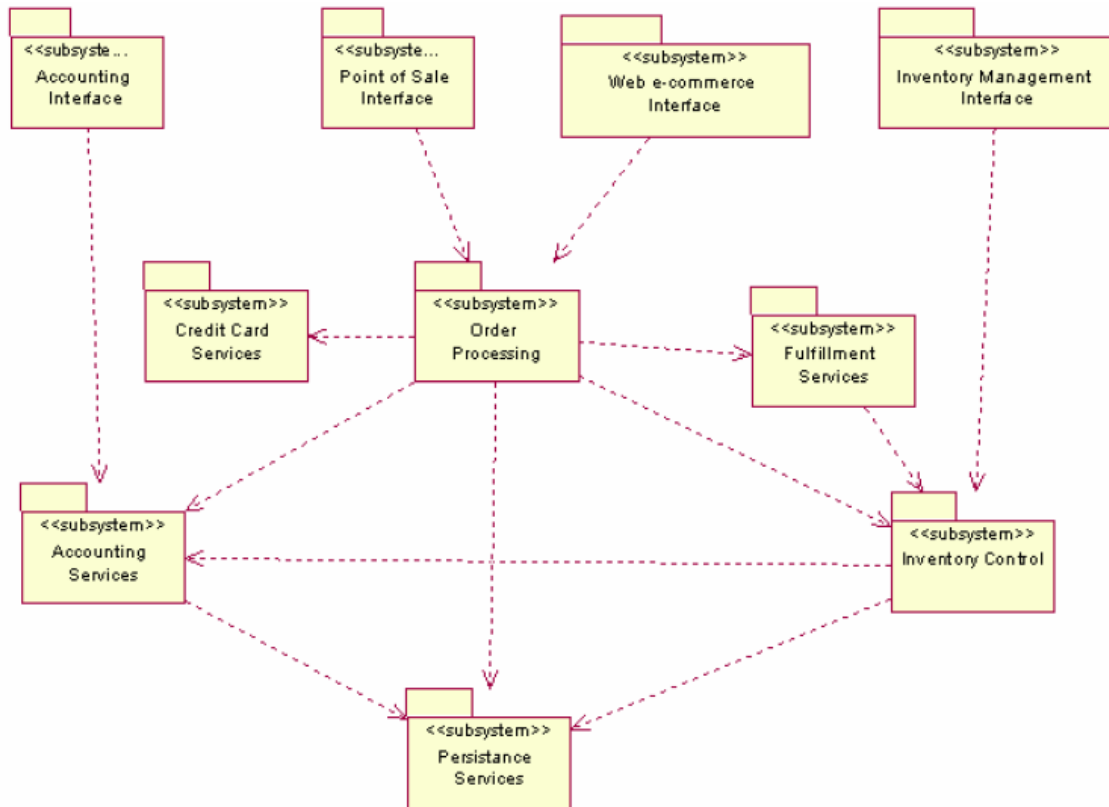
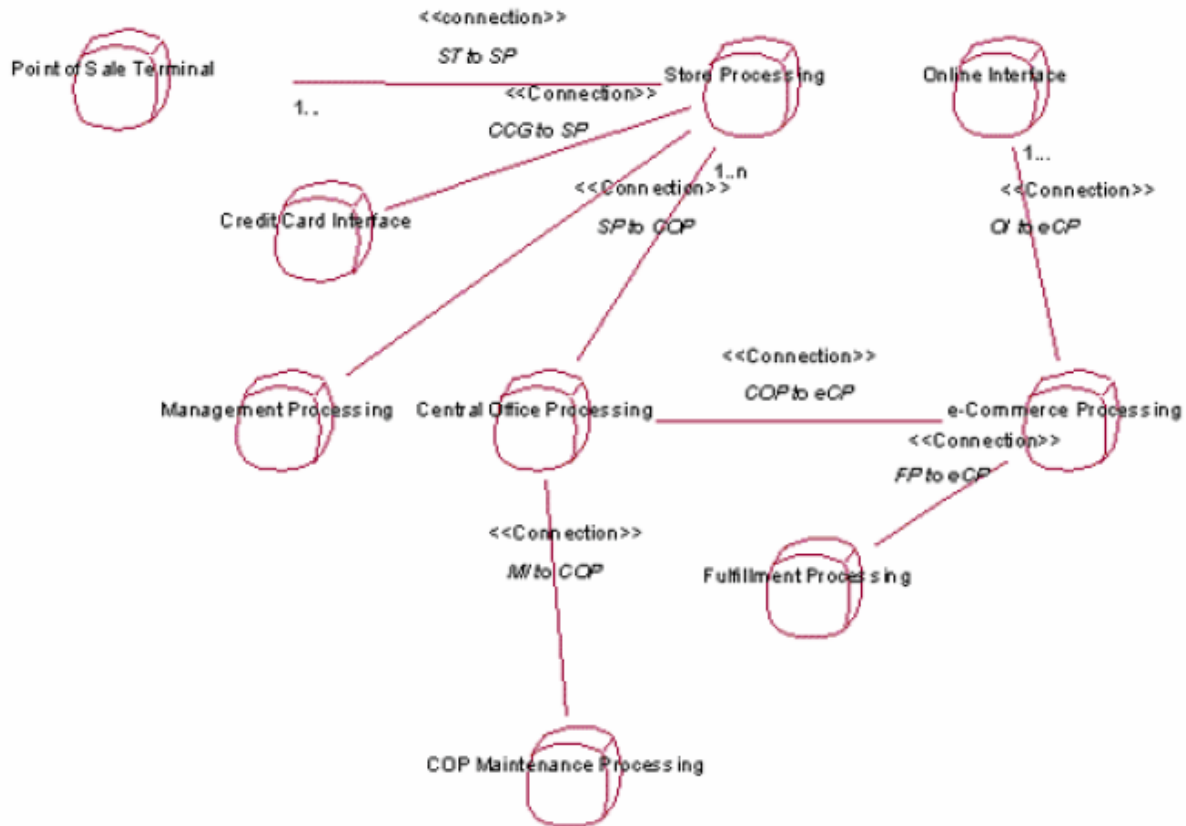


Figure 5. A system context diagram for a retail system (Murray, 2003b)

Worker viewpoint mainly consists of *worker diagrams*, deriving from class diagrams containing *worker* and *machine* stereotypes. Logical viewpoint consists of *context diagrams*, used to depict logical decomposition of the system as a coherent set of UML subsystems that collaborate to provide the desired functionality. In UML 1.4 systems and subsystems inherit from classifiers and packages; there is no UML syntax that captures both the classifier and package aspects of a subsystem. In RUP and RUP SE, proxy classes are used to represent the classifier semantics. In RUP SE, systems/subsystems are stereotypes of proxy and package entities, while their distinct semantics are appropriately defined. *System context diagram* captures a black box description of the system (*Context level*) and is further decomposed to its components in *System Logical Decomposition Diagram* (*Analysis level*). Figure 5 presents as an example a RUP SE *system context diagram* for a retail system.



**Figure 6. A locality diagram for a retail system (Murray, 2003b)**

In the Physical viewpoint, the system is decomposed into elements that host the logical subsystem services. *Locality diagrams* are the most abstract expression of this decomposition. They express where processing occurs without tying the processing locality to a specific geographic location, or even the realization of the processing capability to specific hardware. Locality refers to proximity of resources, not necessarily location, which is captured in the design model. The locality diagrams show the initial partitioning, how the system's physical elements are distributed, and how they are connected. The term locality is used because locality of processing is often an issue when considering primarily nonfunctional requirements. Locality is defined as stereotype of UML *Node* element. Figure 6 presents a *locality diagram* that documents an engineering approach to a click-and-mortar enterprise that has a number of retail stores, central warehouse and a web presence. The rounded cube icon is used for the representation of the locality.

To support RUP SE, a RUP plug-in is provided for IBM Rational tools. The currently available plug-in was released in June 2003 and can be used together with RUP v. 2003. It is based on UML 1.4, while in future versions RUP SE will move on to UML 2.0 semantics.

### 4.3. Use-case flowdown methodology for EIS architecture design

Moving down model levels adds specificity to the models. As you move down the levels, each view is a more specific decision, resulting in configuration items at the implementation level. It is important to note that each model level *realizes* requirements discovered at a higher level. For example,

Physical viewpoint at the design level contains a descriptor node diagram, which shows a physical design that realizes each locality contained in the system locality diagram.

1	Model an enterprise whitebox as a set of collaborating systems.
2	Model how systems collaborate to realize enterprise services, mission, and so forth.
3	Create a context diagram for the system.
4	Determine actors (i.e., entities that collaborate with the system).
5	Identify I/O entities.
6	Aggregate similar collaborations between the system and its actors into use cases.
7	Add use-case detail: performance, pre- and post-conditions, and so forth.
8	Identify system service and aggregate similar whitebox steps.
9	Add system attributes from your analysis of enterprise needs.

**Table 2. Simple flowdown example – System Context Diagram (Murray, 2003c)**

The Context level treats the entire system as a single entity, thus the transition from Context to Analysis level is the process of adding detail in the system model (black box to white box representation). In going from Analysis to Design, subsystems/classes and localities are transformed into hardware, software, and worker designs. This is not a direct mapping; since design decisions have to be made about how the functionality represented in the subsystems and classes will be allocated. Factored into these design decisions are considerations for supplementary requirements and distribution represented by the localities. The resulting design must realize all of the specifications from the Analysis level. In other words, designing the system at the Analysis level, creates requirements that the Design level must satisfy. Again, going from the Design level to the Implementation level is a transformation, but this time the mapping is more direct.

Use-case flowdown (Murray, 2003c) is the methodology used for the transition between model levels. Flowdown can be applied to add detail within a model level or to specify elements at a lower model level. For example, it can be used to determine system services at the Context level, but similarly, it can be used at the Analysis level to identify subsystem services and to break subsystems into further subsystems. Through use-case flowdown requirements may propagate from context to analysis and to design model levels. Use-case flowdown is applied recursively. Table 2 includes the steps of a simple flowdown for constructing system context diagram, as the one described in figure 5, and identifying system services.

Flowdown steps may be applied as a *joint realization* analyzing the way the elements of multiple viewpoints collaborate in carrying out a service. The generic procedure of joint realization flowdown for context model level is presented in Table 3.

1	Choose the participating viewpoints. The logical viewpoint is mandatory.
2	<p>For each white box step in realizing a black box service, you must:</p> <ul style="list-style-type: none"> <li>- Specify the logical element that executes it.</li> <li>- Model how the additional viewpoints participate. For example, you might include: <ul style="list-style-type: none"> <li>- <i>Physical viewpoint</i> -- Specify hosting locality; if there are two localities, then decompose into two steps.</li> <li>- <i>Process viewpoint</i> -- Specify executing process; if there are two processes, then decompose into two steps.</li> <li>- <i>Information viewpoint</i> -- Specify which data schema element supports handling of any information that is used.</li> </ul> </li> </ul> <p>Throughout this process, apply the following <i>joint realization rule</i>: If a given logical element white box step requires more than one element of the other viewpoints, divide that step into further steps so that each step requires exactly one.</p>
3	<p>Create interaction diagrams for each viewpoint:</p> <ul style="list-style-type: none"> <li>- Architecture interaction diagram</li> <li>- Locality interaction diagrams</li> <li>- Process interaction diagrams</li> </ul>

**Table 3. Joint realization procedure (Murray, 2003c)**

#### 4.4. Discussion

The plethora of views all referring to the system model, although providing the capability of detail system description, they are complex to manage. The most important issue is that they should be kept aligned and consistent with respect to each other. The design activity must ensure that these views can be related to each other, either directly or indirectly, and to the information system as well. Thus, in order to ensure consistency and avoid the loss of critical information during system design, various types of relations between different views (and corresponding models) should be enforced (e.g. equivalence or refinement relations). To this end, the formal definition of a metamodel describing views lacking in RUP SE is very important.

RUP SE addresses all issues related to EIS design, utilizing the 6 viewpoints defined. Furthermore, different levels of detail are supported through model levels. Although the use-case flowdown methodology is concrete, it is a complex process, which can not be easily automated. The integration of specific tools for system design is also not mentioned.

UML 1.4 diagrams are employed for the illustration of proposed views. RUP SE framework defines appropriate stereotypes for the views and a plug-in is provided for UML 1.4. UML 2.0 will be supported in a later version.

RUP SE is best suited for EIS that are large enough to obtain internal complexity, have concurrent hardware and software development, obtain architecturally significant deployment issues and include a redesign of the underlying information technology infrastructure to support evolving business processes. Usually it is applied in conjunction to RUP.

### **5. UML4ODP**

UML4ODP is a standard developed by ISO (ISO/IEC, 2006), which further refines the ODP systems by using UML for the expression of ODP system specification in terms of RM-ODP viewpoint



specifications. Using UML concepts, as well as the lightweight extension mechanism supported by UML, it provides:

- a set of UML 2.0 profiles (one for each RM-ODP viewpoint) and the a way to use these profiles
- a profile for correspondences between viewpoints
- a profile for conformance of implementations to specifications.

UML4ODP is also concerned about the relationships between RM-ODP viewpoint specifications and model driven architectures such as MDA. UML notation contributes to RM-ODP's acceptance and promotes its usage by system designers. The Engineering Profile of UML4ODP expresses the concepts specified in the RM-ODP engineering viewpoint and conforms to engineering language.

### 5.1. Engineering Viewpoint metamodel

The basic entities of engineering viewpoint metamodel and their interrelations as defined in UML4ODP standard are illustrated in figure 7. Most of the entities presented in the figure have been briefly introduced in section 2.4. The metamodel is defined using standard UML notation.

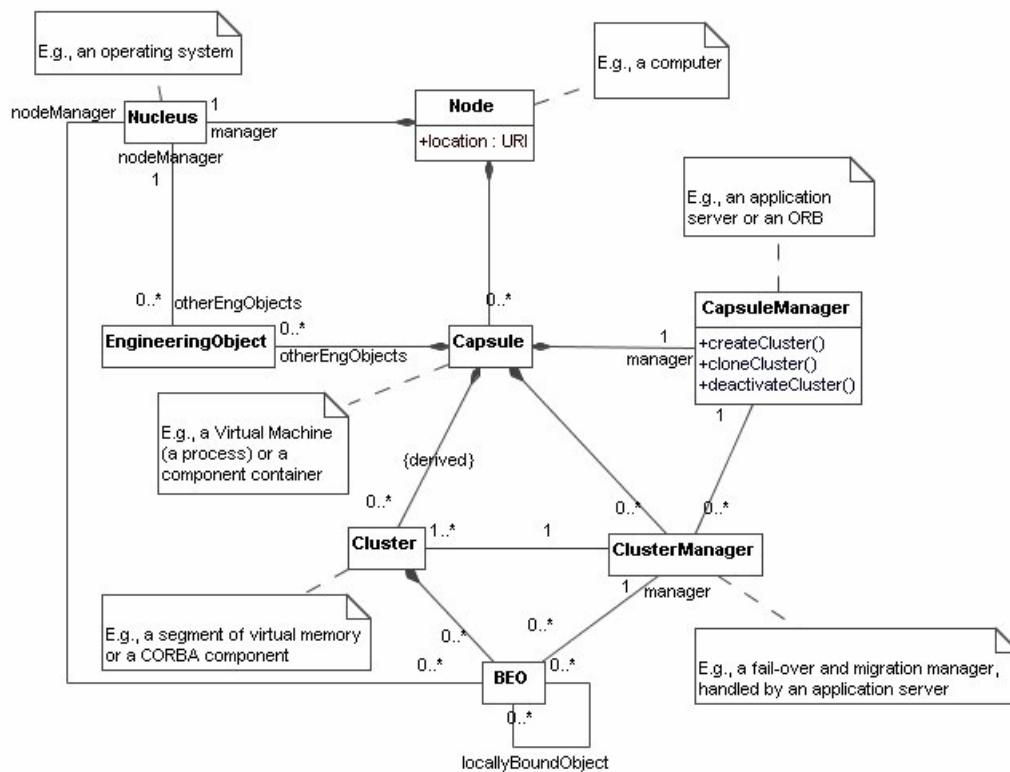


Figure 7: Part of the RM-ODP Engineering Viewpoint metamodel (ISO/IEC, 2006)

### 5.2. Engineering Viewpoint UML profile

In the UML4ODP engineering profile, an *engineering object* is expressed by a UML InstanceSpecification of component (e.g. an instance of *component* UML classifier), stereotyped as *NV\_Object*. Basic engineering objects are particular kinds of engineering objects. Therefore, stereotype *NV\_BEO* that identifies such objects, inherits from *NV\_Object*. A cluster is expressed by a UML InstanceSpecification of component, stereotyped as *NV\_Cluster*. This includes a configuration of basic engineering objects and has bindings to required channels for communication. Likewise, cluster manager, capsule manager, nucleus, and node are expressed by a UML InstanceSpecification of component, stereotyped as *NV\_ClusterManager*, *NV\_CapsuleManager*, *NV\_Nucleus* and *NV\_Node* respectively. A *channel* is expressed by a UML package, stereotyped as *NV\_Channel*. It consists of stubs, binders, protocol objects, and possibly interceptors. It is also expressed by a tag definition of Channel ID for a set of engineering objects (stub, binder, protocol object and interceptor) comprising a channel. Also, a *binder* is expressed by a UML InstanceSpecification of component, stereotyped as *NV\_Binder*. A diagrammatic representation of part of this UML profile is presented in figure 8.

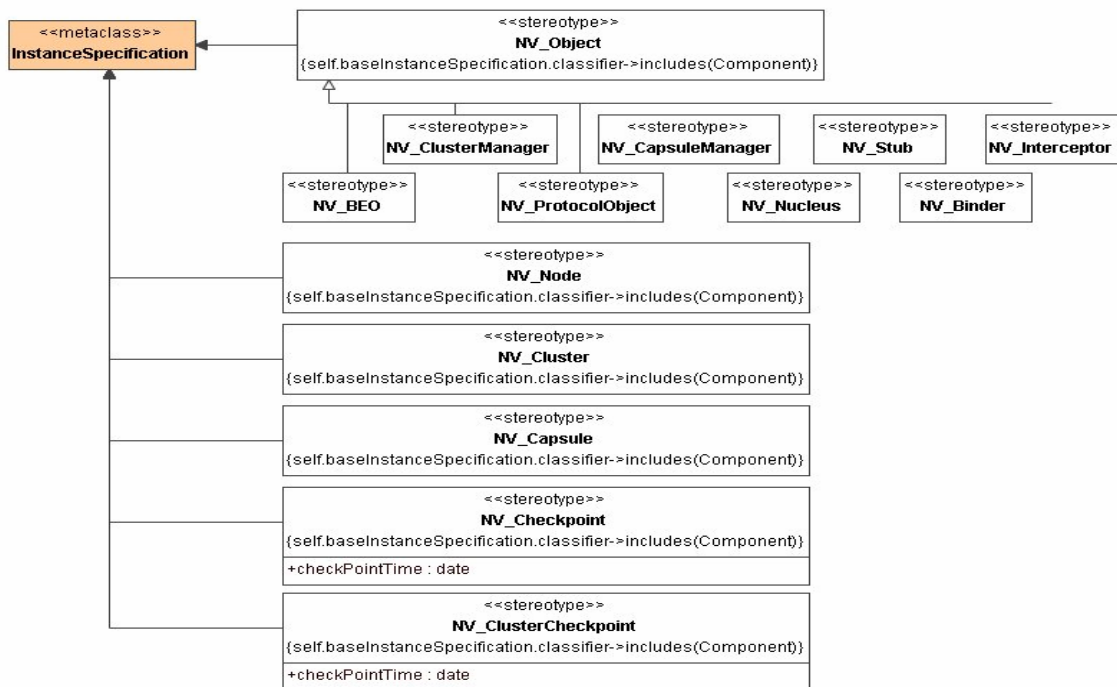


Figure 8: Part of the Engineering profile of UML4ODP (ISO/IEC, 2006)

All the UML elements corresponding to the engineering language are defined within a UML model, stereotyped as *Engineering\_Spec*. Such a model contains UML packages that express:

- structure of a node, including nucleus, capsules, capsule managers, clusters, cluster managers, stubs, binders, protocol objects, interceptors, and basic engineering objects, with UML component diagram,
- channels, with UML component diagram and packages,
- domains, with UML packages
- interactions among those engineering objects, with UML activity diagrams, state charts and interaction diagrams.



The UML profiles of the five ODP viewpoints and corresponding metamodels have been defined using MagicDraw 10.0 and are available from [www.rm-odp.net](http://www.rm-odp.net).

### **5.3. Discussion**

It is self-evident that UML4ODP engineering specification fully corresponds to RM-ODP engineering language. It proposes a well defined metamodel including all entities described in the RM-ODP engineering viewpoint. Based on this metamodel, a complete UML 2.0 profile is defined comprising a set of stereotypes, as well as a number of relative constraints written in OCL. The profile contributes to the wider acceptance and usage of RM-ODP, although it inherits all RM-ODP expression difficulties since it adopts the same terminology.

UML4ODP, as RM-ODP itself is mainly focused on distributed application implementation based on the business requirements the applications should fulfill. However, performance requirements, for example, expected response time for a certain application are not considered. In particular, in the engineering viewpoint, allocation and replication decisions are not effectively explored, since the designer may relate engineering objects to nodes but still has no means to explore the performance of alternative design decisions.

Also, it is not within the scope of UML4ODP to define a formal methodology for EIS architecture design even though EIS architectures may be represented within engineering viewpoint specifications. Lastly, the integration of specific design tools (for example for resource allocation or performance evaluation) is also not mentioned.

## **6. EIS Engineering Framework**

Like RUP SE and UML4ODP Engineering Viewpoint, EIS framework aims at augmenting the system analysis and design through model development. In particular, the framework provides:

- A metamodel describing different views and the relations between them (EIS metamodel). These relations are strictly defined using constraints. The defined viewpoints provide the means to a) describe the network architecture, b) describe application logic in terms of the service requirements imposed to the network infrastructure and c) perform resource allocation.
- A methodology for EIS engineering based on the proposed views. The methodology consists of discrete stages performed by the system designer, software tools or a combination of both. Taking advantage of the formal definition of relations identified between views, system engineering stages may be invoked as a result of metamodel constraint validation, ensuring that each stage can be independently performed.
- A UML representation for all defined views. A UML 2.0 profile is defined for this purpose (EIS engineering profile).

The framework is based on three complementary viewpoints:

*Functional Viewpoint* is used to describe functional specifications (e.g. system architecture, user behavior and application requirements). System architecture refers to the architectural model adopted. In the case of EIS, multi-tiered client-server models are described. Services provided by each application tier (called modules) are also defined. User behavior is modeled through user profiles defining the behavior of different user groups and their performance requirements. Application requirements are described in terms of quality of service (QoS) requirements imposed to

the network infrastructure, e.g. amount of data processed, transferred or stored. Each service is described in a greater level of detail through the *service description* sub-view.

*Topology Viewpoint* facilitates the definition of system access points and the resource allocation and replication. The term *site* is used to characterize any location (i.e. a building, an office, etc.). As such, a site is a composite entity which can be further analyzed into subsites, forming thus a hierarchical structure. Functional and Topology views are interrelated. Resources (e.g. processes and files) correspond to services and data described through Functional view and are located into sites.

*Physical Viewpoint* refers to the aggregate network. Network nodes are either workstations allocated to users or server stations running server processes. Topology and Physical views are interrelated. Both are decomposed to the same hierarchical levels of detail. At the lowest level, network nodes are related to processes/data replicas.

Figure 10 suggests a mapping of the proposed views to System Model row of Zachman’s matrix and RM-ODP Engineering viewpoint (note that Frankel suggestion is adopted).

<b>Zachman Framework - System Model</b>	People	Function	Data	Process	Network	
<b>EIS Engineering Framework</b>	Functional			<i>not addressed yet</i>	Topology	Physical
<b>RM-ODP Engineering Viewpoint</b>	Logical Sub-Viewpoint					Deployment Sub-Viewpoint

**Figure 10: Mapping EIS Engineering Viewpoints to Zachman Framework and RM ODP**

### 6.1. EIS Metamodel

Following, the metamodel will be analytically described in respect to each viewpoint.

#### *Functional Viewpoint*

For each distributed application operating in the EIS, a discrete *Functional View* is defined. Applications are conceived as sets of interacting *modules* (either server or client), such as Application Servers, Database Servers, etc. Modules represent a coherent unit of functionality provided by a system. Each module offers specific *services*, representing the specific set of tasks executed when a module is activated in a certain way. *Data entities* are defined to indicate portions of data used by application modules. A File Server module is used in each application for managing data entities. For each data entity, the name, size and specific characteristics (whether it is executable or data, shareable, updatable and replicable) must be defined.

User behavior is also described in Functional View, through *user profiles* activating client modules. Each profile includes *user requests*, which invoke specific client services. Each *user request* acquires a *percentage* attribute, indicating how often the user activates the specific application module and a *response time* attribute indicating the time within which the request must be served.

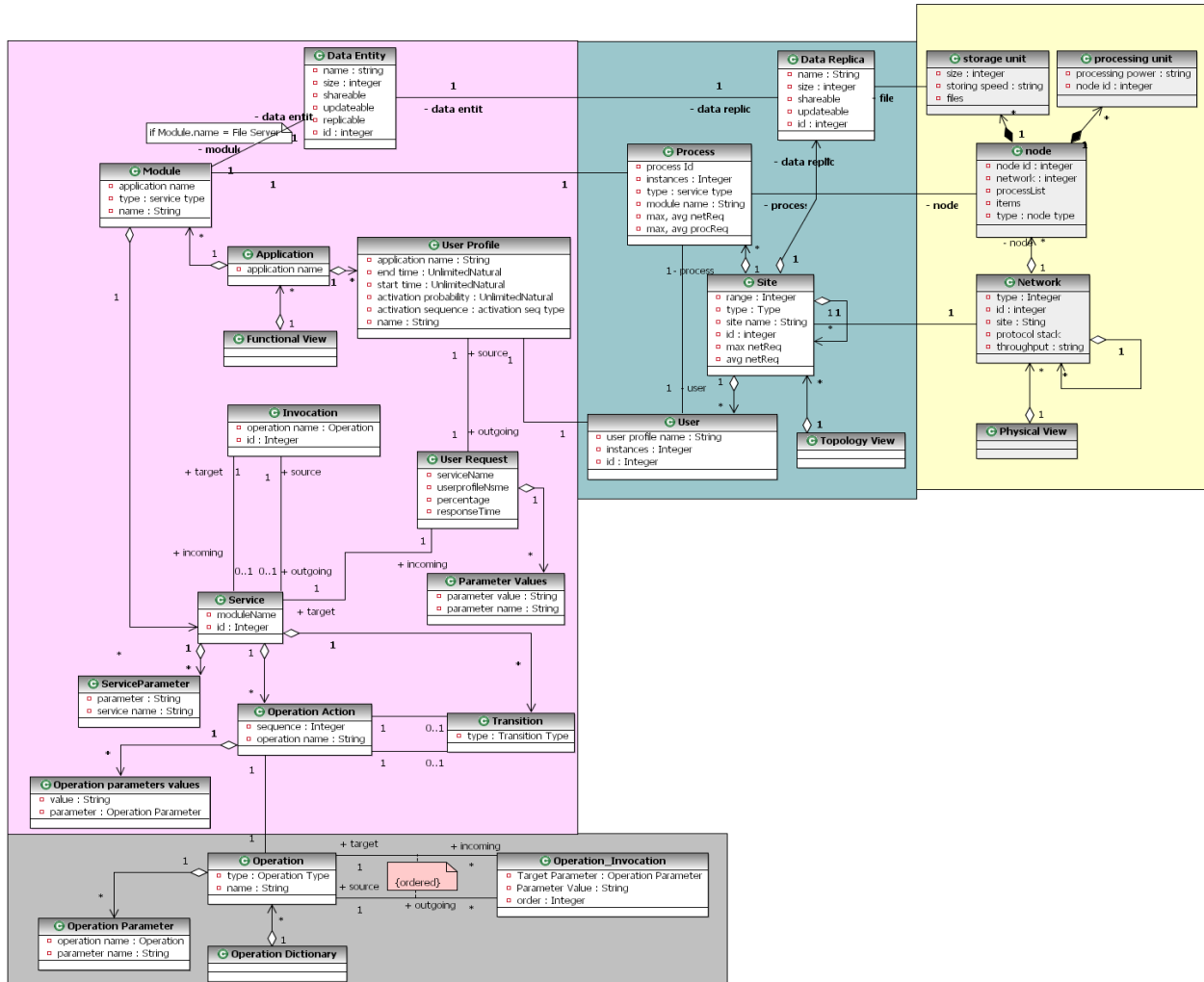


Figure 11: Proposed EIS Metamodel

For each module service, the requirements imposed to the distributed platform infrastructure must be defined. Thus, the portion of data processed, stored or transferred must be estimated. Also other services participating in its implementation must be identified. This is performed using a set of predefined *operations*, sketching service functionality and describing its needs for *processing*, *storing* and *transferring* (called *elementary operations*) (Nikolaidou & Anagnostopoulos, 2005). Since it is difficult for the system designer to estimate the elementary operations describing service requirements, an operation library, named *Operation Dictionary* is provided. *Complex operations* are added in the dictionary, as *request* responsible for other service activation, or *write/read* for data entity management. Complex operations represent the requirements of composite functionality. All complex operations are further decomposed into others, elementary or not. The system designer may add custom complex operations in the Dictionary, to ease the description of a specific application. Thus, a Service Description sub-view is defined for every service appearing in the Functional view (see figure 11).

### Physical Viewpoint

Physical view comprises the network infrastructure. The overall *network* is decomposed to subnetworks producing thus a hierarchical structure. LANs typically form the lowest level of the

decomposition. Nodes, such as servers and workstations are associated with LANs of the lowest level. Nodes may include a *processing unit* and a *storage unit*.

### *Topology Viewpoint*

Topology view comprises sites, processes (defined as instances of application modules), data entity replicas (stored in the corresponding File Server processes) and users (defined as instances of user profiles) (see figure 11). Two types of sites are supported: composite, composed by others, and atomic, not further decomposed, constituting therefore the lowest level of site hierarchy. Users, processes and data replicas are associated with atomic sites. In essence, the hierarchy indicates where (in which location) each process runs and each user is placed. The site hierarchy should correspond to the network hierarchy depicted in the Physical view, while processes, files and users are related to nodes included in Physical view. Each site is characterized of Quality of Service (QoS) requirements as *average and maximum network rate* regarding process communication a) within site limits (*avgWithin* and *maxWithin* attributes of the Site entity), b) exiting the site (*avgOut* and *maxOut*) and c) entering the site (*avgIn* and *maxIn*). These requirements must be satisfied by *throughput* attribute of the corresponding network (see attributes of *Network* entity in figure 11). Thus, Topology and Physical views are interrelated. Both views can be either defined by the system designer or automatically composed by logical and physical configuration tools. The introduction of progressive site refinement, as well as the mapping of site range onto network range, enables the identification of dependencies between application configuration and network topology (Nikolaidou & Anagnostopoulos, 2005).

Consistency between these two views is accomplished using constraints embedded in the metamodel. Some of the constraints implementing the restrictions imposed between Topology and Physical views include:

- Network and site hierarchy must be identical, thus corresponding network and site entities must have corresponding parents. This constraint is used to initiate the respective logical or physical configuration tool, whenever the site or network hierarchy is changed.
- Topology View may only contain components (e.g. processes) related to entities (e.g. modules) belonging to existing Functional Views.
- Constraints are used to relate Topology view entities (e.g. a server process) to the respective Physical view entities (e.g. server node).

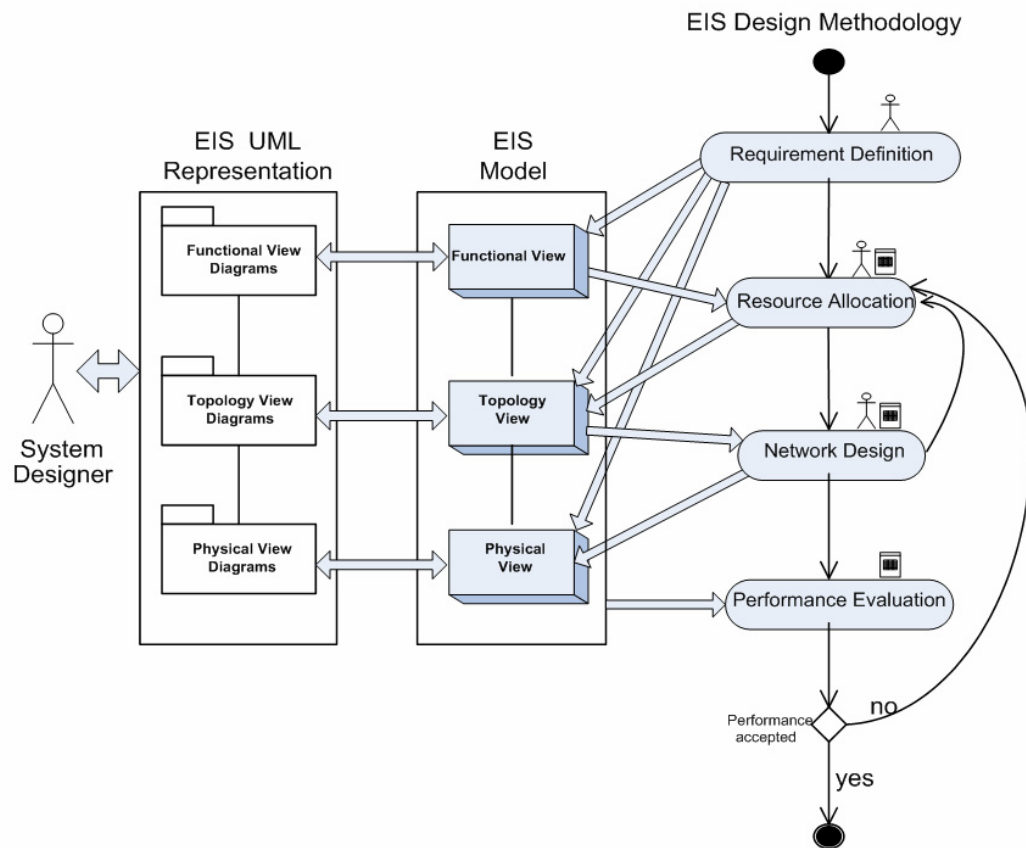
It is obvious, that definition of constraints is a powerful mechanism to represent the dependencies between Topology and Physical view in a similar fashion for both the user (system designer) and configuration software tools.

## **6.2. EIS engineering Methodology**

EIS engineering framework facilitates the following discrete stages of System Engineering process:

1. System requirement definition
2. Resource (process/data) allocation and replication policy definition
3. Network architecture design
4. Performance evaluation of the proposed solution (prior to implementation); although it is not a necessity, it is certainly useful.

As resource allocation and network design problems cannot be independently solved, stages (2) and (3) are repeatedly invoked for different abstraction levels until an acceptable solution is reached (Nikolaidou & Anagnostopoulos, 2005). Both resource allocation and network architecture problems are usually supported by automated or semi-automated tools using mathematics, heuristics or a combination of both. These tools may be repeatedly invoked for different abstraction levels (Graupner et al., 2001, Nezlek et al., 1999). The system designer may perform or partially perform these tasks on his own, thus both options must be supported. To evaluate system performance, a simulation tool as the one described in (Nikolaidou & Anagnostopoulos, 2003) can be used. The simulator uses as input the overall system model and produces performance results. Since each of these tools supports its own representation metamodel (for example queuing networks, Petri-nets, objects), there is a need to properly create and instantiate the “internal” system model prior to invoking the tool. In order to facilitate model exchangeability, the common metamodel is realized in XML, which is a standard exchangeable format. The partial transformation of the common metamodel into tool-specific metamodel must be facilitated before using an existing tool for a specific configuration stage.



**Figure 12: EIS Engineering Framework**

The proposed methodology stages along with EIS model consisting of the predefined views are presented in figure 12. Discrete stages receive/modify information from/to specific system views, as depicted by the arrows between them. The relation between views and between stages is also depicted in the figure. Requirement definition is the initial stage and corresponds to the definition of system architecture and application requirements (Functional view), the system access points



(Topology view) and existing network architecture – if any- (Physical view). A metamodel is provided for the formal definition of views and the relations between them. Each view is represented by one or more UML diagrams properly extended, thus a corresponding UML 2.0 profile is defined. Relations between views must also be described in the UML profile. Specific tool invocation and co-ordination must also be facilitated either by the profile or the metamodel itself or by both.

The metamodel itself contains relationships and restrictions inflicted between system entities belonging to the same or different views, which may lead to a specific stage invocation (e.g. if the network hierarchy in the Physical view is modified, this modification must be depicted in the Topology view as well). Embedding restrictions within the metamodel facilitates EIS engineering process management taking into account the overall system model and not a specific system view corresponding to a discrete stage. Thus, the overall process becomes more effective, since discrete stage (and corresponding tool) dependencies are depicted within the model as view dependencies and consequently they are easily identified. Furthermore, it becomes more efficient to integrate autonomous software tools at different levels of detail, as each of them is independently invoked without knowing the existence of others.

### 6.3. EIS Engineering UML 2.0 Profile

In order to provide a standard method to represent system views and help the designer to efficiently interact with them, a UML 2.0 profile was defined facilitating the following:

1. Representation of EIS metamodel different views. More than one UML 2.0 diagrams may be used for each view. Thus a specific system entity may participate in more than one diagram represented through a different UML entity.
2. Linkage between different model views, as represented in the metamodel.
3. Representation of all relationships and restrictions included in the metamodel. This must be applied between entities participating in the same or different UML diagrams to ensure model consistency.
4. Definition of system entities, attributes and relationships.

UML 2.0 diagrams are used for the representation of different EIS views. The relative EIS entities are depicted as UML elements, properly extended to include additional properties and constraints. This means that appropriate UML 2.0 stereotypes have been defined for each view. Essentially, the concepts of the metamodel are reflected onto the stereotype attributes and constraints. Attributes convey the information required to describe EIS metamodel entities (e.g. *throughput*, *activationFrequency*, *processingPower* etc.). Constraints, which are extensively used within the profile, represent relationships and restrictions between metamodel entities maintaining model consistency. Constraints mainly facilitate:

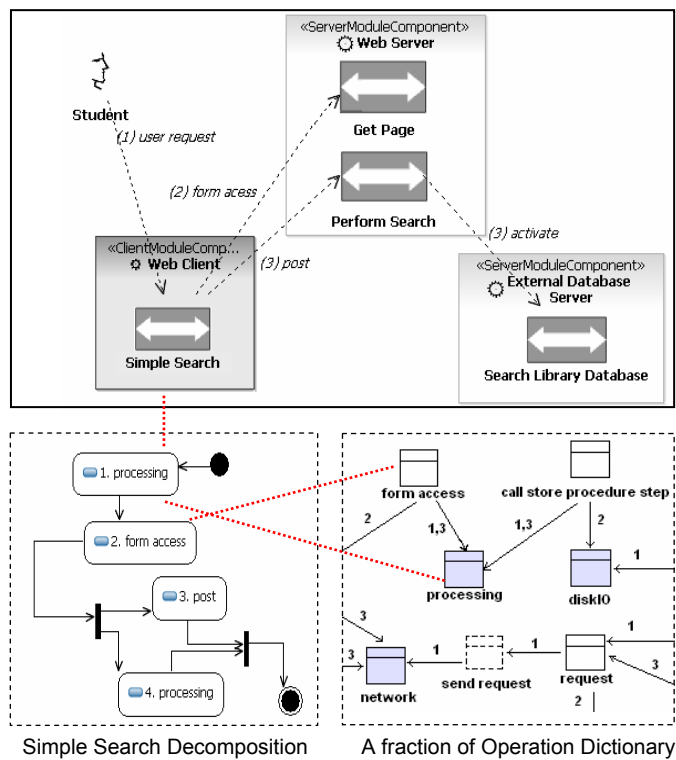
- automatic computation of specific attribute values
- limiting attribute value range
- relating attribute values of specific elements to attribute values of other entities belonging to the same or other UML diagrams (implementing thus the linkage between different models) and
- model validation in view and overall model level.

Attributes and constraints for each stereotype are analytically introduced in (Nikolaidou et al., 2006). Following, the UML diagrams employed for each view are briefly presented. Each stereotype has

been named so that the first part of the name indicates the corresponding EIS metamodel entity, while the second part denotes the UML class it derives from.

### Functional View

Functional Views are represented as UML Component diagrams, since the latter are eligible for representing system functionality at a logical level. As such, modules are defined as stereotypes of the UML *Component* element (*ServerModuleComponent* and *ClientModuleComponent*). Module services are also defined as stereotypes of *Component* (*ServiceComponent* stereotype) because in UML, a component has recursive properties, meaning that it may include other components. For the interactions among services, the *InvokeDependency* stereotype has been defined. *Dependency* is the relationship defined in UML between components. *FileServerModuleComponent* is the stereotype defined for the representation of a File Server, which is associated with *DataEntityComponent* stereotypes used to depict data.



**Figure 13. Functional View example**

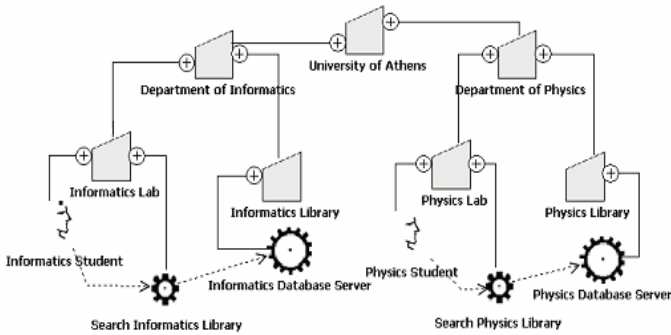
The *UserProfileComponent* stereotype has been defined for the representation of user profiles. Each profile may initiate client module services. Therefore, we have defined the *InitiateDependency* stereotype as a specialization again of UML *Dependency*. As mentioned earlier, the interaction between user profiles and services plays a determinative role in system engineering, since user profiles include performance requirements imposed by users. This is indicated by attributes of the *UserProfileComponent* and *InitiateDependency* stereotypes. *ActivationProbability* attribute, for example, denotes how often a service is initiated while the user profile is active. *Percentage* attribute of the *InitiateDependency* stereotype indicates how often a specific service is activated by the user profile, while *responseTime* denotes the time constraints imposed to the execution of the service in respect to the user profile.

Concerning service implementation, it is represented through an activity diagram (*ServiceImplementationActivity* stereotype), as it involves flow of operations. Consequently, each *ServiceImplementationActivity* maps to a *ServiceComponent*. Thus, these two stereotypes have the same attributes. As already mentioned, service implementation consists of a sequence of operation activations executed upon module activation. Operations are represented through the *OperationAction* stereotype. The Operation Dictionary that includes the operations is represented through communication diagrams as the latter are used to show interactions among elements.

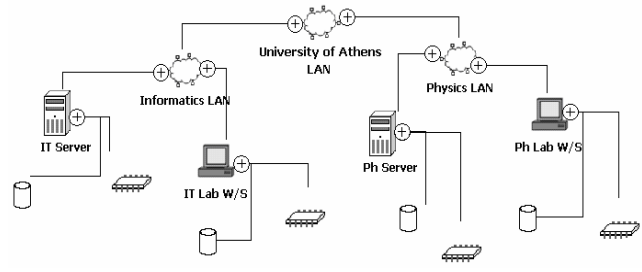
Figure 13 presents a simple application as an example. A user (student) initiates a simple search in a library OPAC, thus performs a database search through the appropriate CGI in the Web server. The example involves three modules: *Web Client*, *Web Server* and *External Database Server*, consisting of services. Web Server module, for example, includes two services, *Get Page* and *Perform Search*. Figure 13 illustrates also the implementation of the *Simple Search* service as well as a fraction of the Operation Dictionary. The dotted lines indicate the correspondences among the external part of Functional View, the implementation of Simple Search and the Operation Dictionary fraction.

*Physical View*

UML deployment diagrams are typically used to represent network architectures (Kaehkipuro, 2001). As such, the elements that denote devices are represented through stereotypes of *Device* (*ServerDevice*, *WorkstationDevice*, *ProcessUnitDevice*, *StorageUnitDevice* stereotypes), which is a specialization of the UML *Node* element, commonly used in deployment diagrams.












**Figure 14a: Topology View Example**



**Figure 14b: Physical View Example**

As each network comprises sub-networks, the most suitable UML element for its representation is the *Package* element, which is used for grouping purposes. Thus, we have created the *NetworkPackage* stereotype from the UML *Package* element. These stereotypes may be connected to each other through the membership relation introduced in UML 2.0. Its notation is presented in the example of figure 14b. This example illustrates part of the University of Athens Library network.

	Stereotype	Notation	Constraints
Physical View	Network Package		<ol style="list-style-type: none"> <li>1. The value of attribute <i>type</i> is either "atomic" or "composite".</li> <li>2. Composite <i>NetworkPackages</i> contain only other <i>NetworkPackages</i> while simple <i>NetworkPackages</i> correspond to simple LANs and contain only <i>ServerDevices</i> or <i>WorkstationDevices</i>.</li> <li>3. Each <i>NetworkPackage</i> corresponds to a single <i>SitePackage</i> in the Topology View.</li> <li>4. Corresponding <i>Network</i> and <i>Site Packages</i> are of the same type.</li> <li>5. Corresponding <i>Network</i> and <i>SitePackages</i> have corresponding parents.</li> </ol>
	Server Device		<ol style="list-style-type: none"> <li>6. Each <i>ServerDevice</i> relates to a set of <i>ServerProcessComponents</i> defined in the Topology View.</li> </ol>
	Workstation Device		<ol style="list-style-type: none"> <li>7. Each <i>WorkstationDevice</i> relates to one <i>userProfileComponent</i> defined in the Topology View.</li> <li>8. Each <i>WorkstationDevice</i> relates to all <i>ClientProcessComponents</i> defined in the Topology View that are invoked by the <i>userProfileComponent</i> assigned to it.</li> <li>9. The <i>items</i> value is the same as the <i>instances</i> value of the corresponding <i>userProfileComponent</i> in the Topology View.</li> </ol>
	ProcessUnit Device		<ol style="list-style-type: none"> <li>10. Each <i>ProcessUnitDevice</i> relates to an existing <i>ServerDevice</i> or <i>WorkstationDevice</i>.</li> </ol>
	StorageUnit Device		<ol style="list-style-type: none"> <li>11. Each <i>StorageUnitDevice</i> relates to an existing <i>ServerDevice</i> or <i>WorkstationDevice</i>.</li> <li>12. Each <i>StorageUnitDevice</i> hosts <i>data replicas</i> defined in the Topology View.</li> </ol>
Topology View	SitePackage		<ol style="list-style-type: none"> <li>1. The value of attribute <i>type</i> must be either "atomic" or "composite".</li> <li>2. Composite <i>SitePackages</i> contain only other <i>SitePackages</i> while simple <i>SitePackages</i> contain only <i>ServerProcessComponents</i>, <i>ClientProcessComponents</i>, and <i>UserProfileComponents</i>.</li> <li>3. Each <i>SitePackage</i> corresponds to a single <i>NetworkPackage</i> in the Physical View.</li> <li>4. Corresponding <i>Network</i> and <i>Site Packages</i> have corresponding parents.</li> <li>5. <i>max</i> and <i>avg</i> attributes are automatically computed based on traffic flow within, in and out of the site.</li> </ol>
	Server Process Component		<ol style="list-style-type: none"> <li>6. <i>application</i> corresponds to one Functional View.</li> <li>7. The <i>module</i> attribute indicates an existing <i>ServerModulePackage</i> in the selected Functional View.</li> <li>8. The value of the <i>name</i> attribute is produced as a concatenation of <i>processId</i> and <i>module</i> attributes.</li> <li>9. Each <i>ServerProcessComponent</i> relates to a <i>ServerDevice</i> in the Physical View.</li> <li>10. <i>NetReq</i> attributes are automatically computed based on traffic flow to the <i>ServerProcessComponent</i>.</li> <li>11. <i>ProcReq</i> attributes are automatically computed based on the processing requirements of the process.</li> </ol>
	DataReplica Component		<ol style="list-style-type: none"> <li>12. The names and other attribute values are extrapolated by corresponding <i>DataEntityComponent</i> attributes of relative Functional View.</li> <li>13. <i>DataReplicaComponent</i> is related to an existing <i>StorageUnitDevice</i> of Physical View.</li> </ol>
	Invoke Dependency		<ol style="list-style-type: none"> <li>14. <i>Invoke</i> connects only <i>ClientProcessComponents</i> or <i>ServerProcessComponents</i> to <i>ServerProcessComponents</i>.</li> <li>15. Every <i>Invoke</i> relationship is included in the corresponding Functional View.</li> </ol>

**Table 4. Stereotypes and Constraints for Physical and Topology Viewpoints**

## Topology View

Topology view is based on UML Component diagrams. All entities included in Topology view are represented through the corresponding stereotypes of Component (*ServerProcessComponent*, *ClientProcessComponent* and *UserProfileComponent* stereotype). Data replicas are also represented through a stereotype of Component (*DataReplicaComponent* stereotype). Since each site comprises sub-sites, the most suitable UML element for its representation is the *Package* element (as with network in Physical view) Therefore, we have defined the stereotype *SitePackage* as a specialization of *Package*. Interaction among process instances, as well as between user profile and client process instances, are represented through the *InvokeDependency* and *InitiateDependency* stereotypes respectively, with different constraints though, defined within the context of Topology view. Sites relate to each other through the membership relationship. The Topology view corresponding to the Physical view of figure 14a is presented in the figure 14b.

As already stated, in order to represent relationships and restrictions between Physical and Topology views of the EIS metamodel and relate the corresponding stereotypes constraints are defined. An excerpt of the constraints defined in both views to ensure the consistency between them is included in Table 4.

Constraints 3-5 of the network package and 3-4 of site package ensure that site hierarchy of the Topology view should correspond to the network hierarchy depicted in the Physical view. Composite sites correspond to composite networks, while atomic sites correspond to atomic networks representing simple LANS. *Max* and *avg netReq* attributes of Site Package are automatically computed based on traffic flow within, in and out of the site (constraint 5 in Topology view). Instances of processes/user profiles (constraints 6-11 in Physical view and constraint 9 in Topology view) and data replicas (constraint 12 in Physical views and 13 in Topology view) located in atomic sites are allocated to nodes (servers or workstations) and storage devices included in the corresponding LAN of Physical view. Note that constraints, as for example those relating data replicas to storage devices, are applied in both views to avoid inconsistencies. Constraints are checked every time the system designer makes a change in either Topology or Physical view or a design tool is invoked. Changes may be either prohibited or propagated.

The proposed UML 2.0 profile has been implemented in Rational Software Modeler in the form of a plug-in (EIS plug-in). EIS plug-in, apart from the definition of the stereotypes and constraints, it also provides additional functionality that first, augments usability for the system designer and second, performs validation of the constraints defined within as well as between viewpoints. Through this plug-in, external tools can be invoked either by the system designer or automatically to enforce consistency between views.

## 6.4. Discussion

In EIS engineering framework, a small number of viewpoints is proposed. The viewpoints as well as their interrelationships are formally defined through a metamodel. Based on this metamodel, consistency between views is ensured through the definition of constraints relating model entities, belonging to the same (view consistency) or different (inter-view consistency) views. The small number of views enables the easy manipulation of them.

EIS framework strictly focuses on system design issues, as resource allocation and architecture specification. The Function column of the Zachman's matrix is regarded as the initial view the

system designer should consider, corresponding to system architecture and functionality (Functional viewpoint). Data and people columns do not fall into the scope of EIS engineering framework as the latter deals only with data allocation and replication policies rather than data specification, while user profiles are used mainly to indicate user behavior and performance requirements affecting system modules. All views are supported by UML 2.0 stereotypes, typically defined in a UML profile. Constraints are used extensively to maintain consistency and depict all the relations defined between system entities included in the metamodel.

The methodology proposed addresses all issues related to EIS design, utilizing the viewpoints defined, through four discrete stages (Requirement Definition, Resource Allocation, Network Design and Performance Evaluation). The last three stages may be performed by the system designer or specialized tools. Moreover, EIS framework supports model exchangeability through the transformation of the common metamodel to internal tool-specific metamodels.

## **7. Future Trends**

Model-based software engineering promotes the development of applications based on consistent models representing both application requirements and the respective implementation. Model-based system engineering, defined in a similar fashion, is based on the assumption that a central system model can be defined covering all aspects related to system analysis and design. In both cases, UML is the dominating choice for representation purposes. An important issue though is the provision of an integrated model for both software and system engineering and the relative methodologies. There are already endeavors initiated, such as RUP framework, working towards this direction. It is important that not only common principles are introduced, but also a formal identification and definition of dependencies and transformations is established.

Agile systems, as described by Dove (2005), are an imperative for modern enterprises operating in highly turbulent environments, since such systems are able to adjust to both expected and unpredicted changes in a timely and cost-effective manner. Model driven architectures, as MDA, are working towards obtaining system agility, which can be strengthened by a common model-based approach to study all aspects regarding enterprise information system design and development. Existing enterprise architecture frameworks, as those presented in this chapter, should be enhanced to serve agility.

## **8. Conclusions**

This chapter discussed model-driven engineering of enterprise information systems and the principle requirements for applying it to EIS analysis and design. Three different frameworks, namely RUP SE, UML4ODP and EIS engineering framework, were studied in respect to these requirements. The frameworks have different origin, and they target at different system engineering aspects. However, they all adopt UML language for model representation and contribute to system analysis and design process. The characteristics of each framework in respect to the system engineering requirements they address are summarized in Table 5.

	RUP-SE	UML4ODP	EIS
Central system model	●	●	●
System Engineering Design Issues	●	●	●
Methodology	●	○	●
UML representation	●	●	●
Model exchangeability – Tool Integration	○	○	●

Legend:  
○ not supported  
● supported  
● fully supported

**Table 5. Characteristics of RUP-SE, UML4ODP and EIS Frameworks in respect to system engineering requirements**

## **References**

Boer F.S., Bonsangue m.M., Jacob J., Stam A., Torre L. 2004. A Logical Viewpoint in Architectures. Proceedings of IEEE EDOC 2004.

Brown Dave & Densmore Jim 2005. The new, improved RUP SE architecture framework. IBM Rational Edge.

Brown Allan 2004. Model driven architecture: Principles and practice. Software System Modeling, 3, 314–327.

Dijkman R.M., Quartel D.A.C., Pires L.F., Sinderen M.J. 2003. An Approach to Relate Viewpoints and Modeling Languages. Proceedings of IEEE EDOC 2003.

Dove Rick 2005. Fundamental Principles for Agile Systems Engineering. Conference on Systems Engineering Research (CSER), Stevens Institute of Technology, Hoboken, NJ, March 2005.

Frankel D. 2003. Applying EDOC and MDA to the RM-ODP Engineering and Technology Viewpoints: An Architectural Perspective. INTAP - David Frankel Consulting.

Frankel, D., Harmon P., & Mukerji J. 2003. The Zachman Framework and the OMG's Model Driven Architecture. Business Porcess Trends.

Goethals F, Lemahieu W, Snoeck M, Vandenbulcke J. 2006. An overview of enterprise architecture framework deliverables. In Banda RKJ (ed) Enterprise Architecture-An Introduction, ICFAI University Press.

Gomaa H., Menasce D., Kerschberg L. 1996. A Software Architectural Design Method for Large-scale Distributed Information Systems. Distributed System Engineering Journal, 3(3), IOP.

Graupner S., Kotov V., Trinks H. 2001. A Framework for Analyzing and Organizing Complex Systems, Proceedings of the 7<sup>th</sup> International Conference on Engineering Complex Computer Systems, IEEE Computer Press.

Greefhorst D., Koning H. & Hans van Vliet 2006. The Many Faces of Architectural Descriptions. Information Systems Frontiers, 8(2), 103-113.

Hilliard Rich 2001. IEEE Std 1471 and Beyond. Position Paper for SEI's First Architecture Representation Workshop, January 2001.

IEEE Std 1471. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. 2000.

ISO/IEC, 1998. Information Technology – Open Distributed Processing – Part 1 – Overview – ISO/IEC 10746-1 | ITU-T Recommendation X.901.

ISO/IEC, 2006. Information Technology – Open Distributed Processing – Use of UML for ODP system specifications. ITU-T Recommendation X.906.

Kaehkipuro P. 2001. UML-Based Performance Modeling Framework for Component-Based Distributed Systems. Lecture Notes in Computer Science 2047, Performance Engineering, Springer-Verlag.

Leist S. & Zellner G. (2006). Evaluation of Current Architecture Frameworks. Proceedings of SAC'06, April, 23-27, 2006, Dijon, France.

Murray Cantor 2003a. Rational Unified Process for Systems Engineering - Part 1: Introducing RUP SE Version 2.0, Rational Edge.

Murray Cantor 2003b. Rational Unified Process for Systems Engineering – Part II: System Architecture. Rational Edge.

Murray Cantor 2003c. Rational Unified Process for Systems Engineering – Part III: Requirements Analysis and Design. Rational Edge.

Nezlek G.S., Hemant K.J., Nazareth D.L. 1999. An Integrated Approach to Enterprise Computing Architectures. Communications of the ACM, 42(11), ACM Press.

Nikolaidou M. & Anagnostopoulos D. 2003. A Distributed System Simulation Modeling Approach. Simulation Practice and Theory Journal, 11(4), Elsevier Press.

Nikolaidou M., Anagnostopoulos D. 2005. A Systematic Approach for Configuring Web-Based Information Systems. Distributed and Parallel Database Journal, 17, 267-290, Springer Science.

Nikolaidou M., Alexopoulou N., Tsadimas A., Dais A., Anagnostopoulos D. (2006). Extending UML 2.0 to Augment Control over Enterprise Information System Engineering Process. International Conference on Software Engineering Advances (ICSEA 2006), Tahiti, French Polynesia, October 29 - November 1, 2006.

Oliver W. David, Kelliher P. Timothy & Keegan G. James 1997. Engineering Complex Systems with Models and Objects. INCOSE.

OMG Inc, 2006. Object Constraint Language. Version 2.0, 6/5/2001.

OMG Inc, 2007. Unified Modeling Language: Superstructure. Version 2.1.1, 3/2/2007.



Savino-Vázquez N.N. et al., 2000. Predicting the Behavior of three-tiered applications: dealing with distributed-object technology and databases. Performance Evaluation, 39(1-4), Elsevier Press.

Serain Daniel (1999). Middleware. Practitioner Series Springer-Verlag.

Sowa F. J. & Zachman A. J. 1992. Extending and formalizing the Framework for Information Systems Architecture. IBM Systems Journal, 38(2&3), 590 – 616.

Zachman A. J. 1999. A Framework for Information Systems Architecture. IBM Systems Journal, 31(3), 445 –470.

## **Terms and Definitions**

**System Engineering:** The process of analyzing system requirements, designing the desired architecture of a system and exploring performance requirements, ensuring, thus, that all system components are identified and properly allocated and that system resources can provide the desired performance.

**Model-based System Engineering:** A system engineering method providing a central system model (tool-independent) that captures system requirements and design decisions that fulfill them at different levels of abstraction. It enables integration of system models supported by autonomous design tools and interoperability between them without interfering with their internal implementation.

**View:** A representations of the whole system from the perspective of a related set of concerns.

**Viewpoint:** The perspective from which a view is taken. It serves a specific category of system stakeholders.

**System Model:** A set of entities and their relationships describing a system.

**System Model Representation:** A graphical notation for the illustration of a system model.

**Central System Model:** A technology-neutral multi-level model used as a basis for model-based system-engineering.