Enterprise Information System Engineering: A Model-based Approach based on the Zachman Framework

Mara Nikolaidou[1], *Nancy Alexopoulou*[1,2]
[1]*Harokopio University of Athens, Athens, Greece*
[2]*Department of Informatics & Telecommunications, University of Athens, Athens, Greece*
mara@hua.gr, nanci@di.uoa.gr

Abstract

Enterprise information system engineering involves heterogeneous methodologies, tools and system models focusing on different system aspects, which should be integrated. Model-based system engineering may contribute towards this direction, by providing a central system model that captures system requirements and decisions that fulfill them at different levels of abstraction. The central system model serves all engineering activities and should be multi-layered, technology-neutral and modular. Zachman's Enterprise Architecture framework may be used as a basis for constructing a central EIS model. Basic guidelines to incorporate model-based EIS engineering methodologies within Zachman framework are discussed in the paper. Furthermore, a model-based EIS design approach based on these guidelines is presented to explore related benefits and drawbacks.

1. Introduction

When building an enterprise information system (EIS), desired properties, such as its structure and behavior, should be defined, 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 focuses on certain concerns and considers these concerns at a certain level of detail [1]. Various methodologies and frameworks have been suggested aiming at EIS engineering. 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.) [2,3]. To effectively explore EIS engineering, heterogeneous methodologies, tools and system models focusing on different aspects should be integrated.

The desired integration of people, methods, models and tools could be accomplished by adopting EIS model-based engineering. *Model-based system engineering* [4] provides a *central system model* that captures all system requirements and decisions that fulfill them at different levels of abstraction. The central system model serves all engineering activities. Thus, it should be multi-layered, modular and composite, facilitating the integration of system sub-models corresponding to different perspectives and their progressive refinement. Relevant methodologies and tools addressing discrete engineering issues may be applied to specific system sub-models. Thus, we argue that it is best suited for EIS engineering. In such a case, a multi-level, composite and technology-neutral central model for EIS should be defined, taking into account different perspectives and aspects of EIS. Existing well-known frameworks may be used for such a purpose. We argue, though, that Zachman's framework [2] could be most suitable for establishing an EIS central model for model-based system engineering. The strength of the framework is that it provides an organized way of thinking about an enterprise, in respect to information systems, so that they can be described in detail, while the requirements imposed to them are analyzed. Different cells identified in Zachman's matrix may serve as modular sub-models incorporated in a central EIS model integrating all system perspectives. Respective viewpoints [1] should be formally defined to establish the definition of sub-models and interrelations between them, thus the integration and interoperability of discrete methodologies for EIS engineering. As such, Zachman matrix will serve as a canvas to integrate different concerns, issues and methodologies towards enterprise information system engineering, while model-based EIS engineering methodologies may use parts of it as a reference point. This can be accomplished, since no specific modeling method is suggested for Zachman matrix cells, thus, it can be viewed as a shell where discrete EIS models may be encapsulated. Basic guidelines to incorporate model-based EIS engineering methodologies within Zachman framework are discussed in the paper. Furthermore, a model-based EIS design

1530-1605/08 $25.00 © 2008 IEEE

approach based on these guidelines is presented to explore related benefits and drawbacks. EIS design is focused on system architecture definition (both hardware and software), thus corresponding Zachman's cells are involved.

The rest of the paper is organized as follows: Section 2 analyses model-based EIS engineering based on Zachman framework. Related issues are discussed and basic guidelines are suggested. Section 3 presents the model-based EIS design approach adopting these guidelines. Conclusions reside in section 4.

2. Model-Based EIS Engineering

System engineering is defined as "*an interdisciplinary approach and means to enable the realization of successful systems"* [4]. It focuses on defining customer needs and required functionality early in the development cycle, documenting requirements, and then proceeding with design synthesis and system validation while considering the overall system life cycle, performance and even maintenance. Discrete engineering issues are usually resolved by different methodologies and autonomous software tools, thus the support of different system models cannot be avoided. How many different system models though should be supported? Should all of them provide the same level of detail? How can the correspondences between different models be identified and ensured, when needed? In many cases, these models are not compatible, thus, system engineering issues, although may be interrelated, are often solved in isolation. To resolve such a situation, a central model, independent of specific methodologies and tools should be adopted.

Model-driven technologies for application development, such as Model Driven Architecture (MDA) [5], proposed by OMG, enable the definition of *platform-independent models* (PIMs) for the specification of system functionalities, *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. 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 from different perspectives and corresponding decisions that fulfill them. The central system model serves all engineering activities and should be multi-layered, modular and composite, facilitating the integration of system sub-models corresponding to different perspectives and their progressive refinement. Corresponding methodologies addressing discrete engineering issues may be independently applied to specific system sub-models.

Identification of correspondences/ dependencies of modular sub-models should also be facilitated. Based on them, the interoperability between corresponding methodologies could be achieved. In addition, for each system sub-model, tool-specific models could be defined (corresponding to PSMs), while MBSE also provides for model transformation (couplings between PIM and PSMs) without interfering with their internal implementation.

2.1. EIS Central Model based on Zachman Framework

When applying MBSE for Enterprise Information System, a multi-level, composite and technology-neutral model for EIS representation should be defined, taking into account different perspectives, corresponding to different stakeholders, for example the designer, the developer or the owner, and different aspects of EIS, for example people, functionality, architecture. An overview of existing frameworks for EIS modeling can be found in [6] and [7]. We argue, though, that Zachman framework [2] could be most suitable for establishing an EIS central model for MBSE.

The Zachman framework, often referred to as the Zachman matrix, is a logical structure for organizing and classifying the artifacts created during the development of enterprise information systems, as depicted in figure 1.

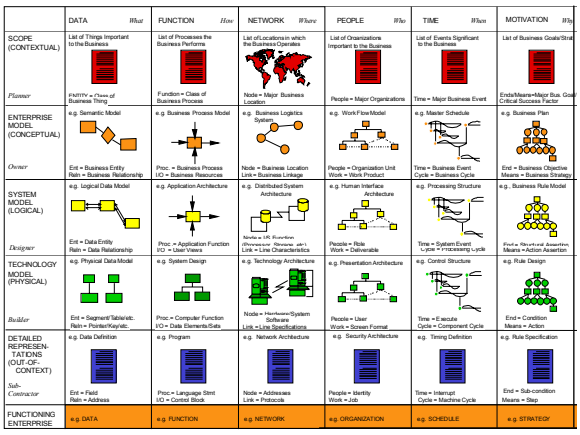

Figure 1. Zachman framework for Enterprise Architecture

The purpose of the framework is to ensure the

establishment of EISs 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. It is deployed in two dimensions. The first dimension addresses the different perspectives of the stakeholders participating in information system development. In particular, Zachman defines 6 perspectives: *Scope* which denotes the business purpose and strategy, defining the context for all others, *Business model*, which is a description of the enterprise within which the information system will function, *System model,* which delineates how the system will satisfy the requirements ensuing from business objectives, *Builder model*, which represents how the system will be implemented, *Out-of-Context* which includes implementation-specific details, and *Operational*, which is the functioning system. The second dimension distinguishes 6 different focal points of the system. *Data* aspect describes what entities are involved, while *Function* aspect shows how the entities are processed. *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 *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 columns and the different perspectives by rows forms the Zachman Framework. For each cell, one could define a wide range of diagrams and documents representing different aspects of the issues studied within its limits.

The strength of the framework is that it provides an organized way of thinking about an enterprise, in respect to information systems, so that EIS and requirements imposed to it can be described and analyzed. It enables the individuals involved in constructing EIS to focus on selected aspects of the system without losing sight of the overall enterprise context. A plethora of methodologies and formalisms exist, each applicable to some subset of cells, while respective system models are defined. Moreover, Zachman matrix may facilitate the identification of inconsistencies between system representations and thus enable experts to modify system models appropriately to eliminate all inconsistencies. Thus, we argue that the 36 different cells identified in Zachman matrix may serve as modular models incorporated in a central EIS model integrating all system perspectives. Sub-models corresponding to cells could be further decomposed into cooperating or refined sub-models, while methodologies addressing specific issues could be associated with specific cells, columns or rows of Zachman matrix.

For effective model-based EIS engineering, models corresponding to Zachman cells and their inter-relations should be formally defined. ANSI/IEEE Std 1471 - Recommended Practice for Architectural Description of Software-Intensive Systems [1] defines enterprise system concepts and their relationships that are relevant for architectural description, thus provides a standard way of defining EIS architecture models. 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". *Stakeholder*s 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". 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. We propose to treat each Zachman cell as a *discrete EIS view*, which may consist of *sub-views*. *EIS views* should be formally defined based on corresponding *EIS viewpoints*

identifying the perspective of a specific stakeholder (as defined in Zachman's matrix rows) for a specific system focal point (as defined in Zachman's matrix columns), for example *network design viewpoint* corresponds to *network* focal point according to the *system designer*.

2.2. Model-based EIS engineering guidelines

To successfully define an architecture description, specific characteristics should be obtained [8]:

- 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 addressed in IEEE 1471, additional issues should also be addressed:

- 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 formalization could have benefits in several topics, as view checking, view integration, and inter-view analysis. Viewpoint formal definition could be the first step towards this direction

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) [9]. The aforementioned definitions are adopted throughout the rest of the paper. The attainment of a consistent system representation entails that view interrelations must be typically defined. In order to formally define a viewpoint, one should define:

a) a name, its purpose and intended audience (stakeholders)

b) a metamodel describing the supported views independently of the modeling language used for system representation, while interrelations between entities participating in the same or different views should be emphasized

c) a representation model for corresponding views

d) methods and techniques employed for analyzing the information appearing in views and

e) its interrelation with other viewpoints

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 [10], two basic relations are identified between views: *refinement* (the internal view refines the external view on a different level of detail) and *complement (t*wo views may complement each other by considering complementary concerns).

Independently of view definition meta-model, we suggest that UML or SysML should be adopted for model representation. 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. UML 2.0 [11] 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 [12, 13]. SysML [14] is a general purpose language for complex system engineering aiming at a

description of a broad spectrum of systems such as mechanical, information systems, etc. SysML also supports the concepts of requirements and resource allocation, while its semantics are more focalized to system engineering though it is fully supported by many UML modeling tools.

Model-based EIS engineering based on the Zachman matrix can be accomplished, by integrating autonomous methodologies addressing specific engineering issues, provided that they fulfill the following guidelines:

1. *Definition of supported viewpoints.*

This is accomplished by determining corresponding Zachman matrix cell and viewpoint purpose that is identifying EIS engineering issues covered.

2. *Definition of a common, multi-layered, model of EIS views for each viewpoint.*

Each view should address a discrete issue and should be formally defined. Furthermore, view and inter-view consistency (belonging in the same or different viewpoint) should be well-established, since the main reason for adopting model-based design is to ensure integration of discrete engineering issues/tools. Since views corresponding to supported viewpoints may also relate to *external* Zachman matrix cells (that is viewpoints not supported by the specific methodology) *external-interrelation* entities are defined in viewpoint meta-model. These entities may serve as linkages between the supported views and specific *external* Zachman matrix cells. It should be noted that no actual interrelation is accomplished, since external viewpoints may be not defined. Though, interrelation rules are partially defined facilitating future mapping between entities and consequentlyinteroperability between discrete methodologies.

3. *Description of a methodology addressing specific engineering issues with respect to supported viewpoints.*

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 solution of related engineering issues.

4. *Definition of a UML/SysML representation model for supported EIS views.*

It should provide for an integrated, easy-to-use interface for corresponding stakeholders.

5. *Tool integration - Model exchangeability.*

Since, even within the purpose of a specific methodology or viewpoint, discrete issues may be resolved using autonomous, heterogeneous tools, tool integration should be supported. Some of them may employ their own internal model for EIS representation. Therefore, tool coordination and internal meta-model transformation should also be supported. According to model-based design principles, consistency is ensured, since the common meta-model acts as a "reference point". Prior to using an existing tool, the partial transformation of the viewpoint meta-model (platform-independent) into the tool's internal meta-model (platform-dependent) must be facilitated. Using this transformation, the invocation and initialization of any tool can be automatically performed.

Following, a model-based approach for *EIS design*, a specific system engineering issue, which is built based on the proposed guidelines, is presented to explore the feasibility of the proposed approach. Once more, it should be noted that, the purpose of the proposed guidelines is to promote the integration of discrete methodologies addressing specific engineering issues, in the case presented system design, provided that they are model-based. Establishing a unique methodology or approach covering all engineering issues based on Zachman matrix, is considered not viable, and thus is beyond the purpose of this paper.

3. EIS Model-Based Design

According to INCOSE [4], s*ystem design* is an important phase of system engineering, determining *system architecture* (i.e. the way system components should be synthesized) to satisfy specific requirements. System design focuses on analyzing performance requirements, system modeling and prototyping, defining and optimizing system architecture and studying system design tradeoffs and risks. *EIS design* is the process of *defining and optimizing the architecture of the information system* (both hardware and software) and exploring performance requirements, ensuring that all software components are identified and properly allocated and that hardware resources can provide the desired performance. Hardware architecture consists of the network architecture and the configuration of processing and storage resources, while software architecture consists of the definition of application architecture (e.g. application tiers and the way they communicate) and the allocation of application tiers and data. In practice, discrete issues, such as network

architecture description or resource allocation are supported by autonomous automated or semi-automated tools [15, 16, 17], each of which adopts its own meta-model (for example Petri-nets, queuing networks or object hierarchies) with different attributes for system representation. Thus, no integration or interaction between them is supported.

A model-based approach for EIS design should be based on a central system model facilitating: 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) association of software components to network nodes (resource allocation), in order to ensure performance requirements.

The first step applying the proposed guidelines of section 2.2 is to identify the corresponding stakeholders and Zachman matrix cells and define the respective viewpoints, thus integrating EIS model used for system design within EIS central model as a sub-model. Since Zachman's *system model* row deals with system design issues, respective cells are emphasized. Supported viewpoints are discussed in section 3.1, while the corresponding meta-model defining them is presented in section 3.2. This meta-model is used for the definition of *EIS Design sub-model* within Zachman's matrix. The relations between system design views are strictly defined using constraints. The model-based EIS design methodology is presented in section 3.3. The methodology takes advantage of the relations identified between views. It comprises discrete stages performed by the system designer, software tools or a combination of both. The UML 2.0 profile (EIS Design UML profile) developed for EIS design sub-model representation is briefly discussed in section 3.4. The integration of software tools performing specific design tasks is also discussed in section 3.3.

3.1. Supported Viewpoints

Based on the scope of the proposed EIS model-based design approach, two cells of Zachman's System Model Row are emphasized: a) *Function*, depicting Application Architecture, which is mainly used to depict application tiers and application performance requirements and b) *Network*, depicting Distributed System Architecture, which should contain the result of system design process. People and Data cells are not utilized, since they do not actively participate in system design, thus corresponding external interrelation entities are defined for them. Two corresponding viewpoints are defined, namely *Function Design Viewpoint* and *Network Design Viewpoint*.

Function Design Viewpoint is used to describe functional specifications (e.g. application architecture, user behavior and application requirements). 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. They act as external interrelation entities for *People* cell (e.g they could be related to user role entities). *Data entities* are defined to indicate portions of data used by applications. They also act as external interrelation entities for *Data* cell. 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. External interrelation entities, ensuring the integration with upper/lower row cells should also be defined. For example, service entity should act as an external interrelation entity to *Function Conceptual* cell, belonging to Enterprise Model row and Function column, which corresponds to the business process sub-model.

Network Design Viewpoint consists of two complementary sub-views: *Topology* and *Physical*. *Topology* view 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* view 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. Frankel [18] also suggests such a separation for RM-ODP

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. Site entity acts as an external interrelation entity to upper network column cells, while network entity acts as such to lower network column cells.

3.2. EIS Design Meta-model

EIS design meta-model is analytically defined in respect to each viewpoint, as depicted in figure 2. As indicated in the figure, for each Zachman's cell a system view and sub-views are defined. Entities belonging in the same view are usually interrelated. The meta-model also contains external interrelation entities (indicated by a cycle around their name) defined for each view. They are used to depict interrelations either between views utilized by EIS design methodology, which are formally defined and, thus, depicted in the figure (for example *module* entity of *function design view* is interrelated with *process* entity of *network design view*), or to other external views corresponding to cells not utilized by the specific methodology. In such case, at least the communication entities, for example *site*, are defined enhancing integration.

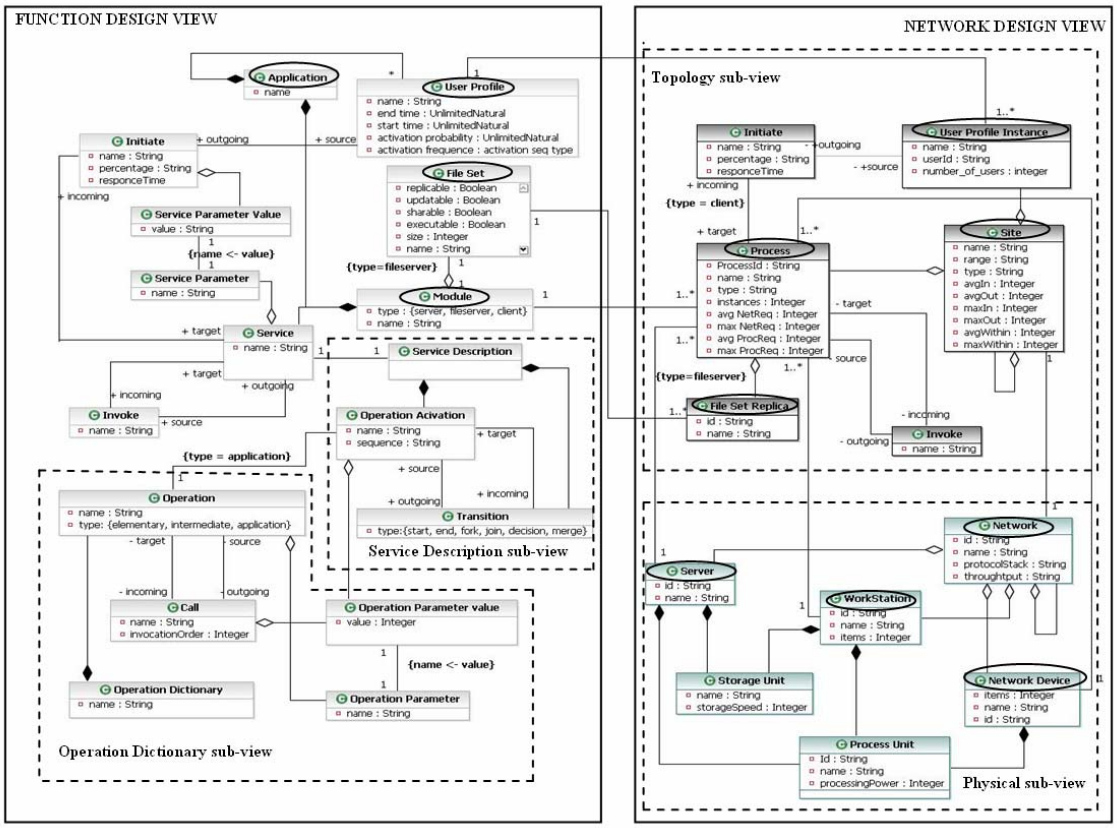

Figure 2. EIS Design Meta-model

Function Design 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. User behavior is also described in Functional View, through *user profiles* (external interrelation entity) activating client modules. Each profile includes *user requests*, which invoke specific client services.

For each service, the requirements imposed to the distributed platform infrastructure must be defined. Thus, a *Service Description sub-view* is defined for every service appearing in Function view. 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*) [19]. Since it is difficult for the system designer to estimate the elementary operations describing service requirements, an operation library, named Operation Dictionary is provided (*Operation Dictionary sub-view*). *Complex operations* are added in the dictionary, representing 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.

Network Design Viewpoint

Physical sub-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 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 2). 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. Thus, Topology and Physical views are interrelated. Both views can be either defined by the system designer or automatically composed by logical and physical design tools [19].

Consistency between the two complementary sub-views is accomplished using constraints embedded in the meta-model. Constraints are also used to ensure function design and network design view interrelations. Examples of the constraints implementing the restrictions imposed between 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 sub-view may only contain components (e.g. processes) related to entities (e.g. modules) belonging to existing Function Views.

3.3. EIS Design Methodology

EIS design process consists of the following stages:

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).

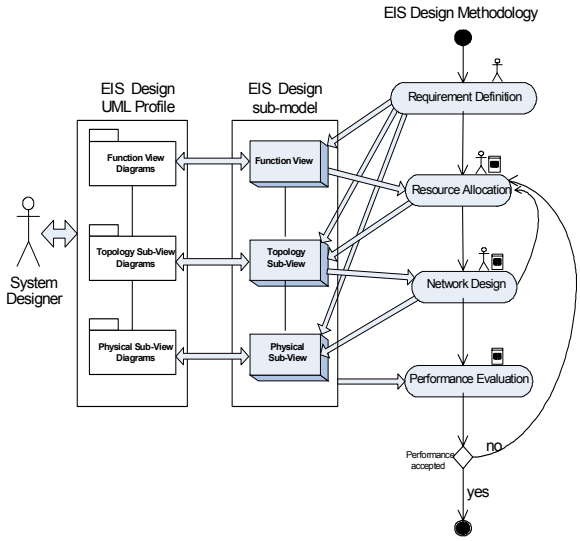

Figure 3. EIS Design Methodology

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 [19]. Both resource allocation and network architecture problems are usually supported by automated or semi-automated tools supporting their own representation meta-model. In order to facilitate model exchangeability, EIS design sub-model is realized in XML, which is a standard exchangeable format. The partial transformation of EIS design sub-model into tool-specific model must be facilitated before using an existing tool for a specific design stage. The proposed methodology stages and the relation between views and design stages are depicted in the figure 3. Requirement definition is the initial stage and corresponds to the definition of system architecture and application requirements (Function view), the system access points (Topology sub-view) and existing network architecture – if any- (Physical sub-view).

3.4. EIS Design UML 2.0 Profile

In order to provide a standard method to represent system design views and help the designer to efficiently interact with them, a UML 2.0 profile was defined. EIS entities are depicted as UML elements, properly extended to include additional properties and constraints. The profile is analytically introduced in [13]. Function View is represented through UML component diagram. Concerning service description sub-view, it is represented through an activity diagram, as it involves flow of operations. UML communication diagrams, which depict interaction between entities, are suitable for the representation of Operation Dictionary sub-view. Physical sub-view comprises the network infrastructure. As such it is depicted through UML deployment diagrams. Lastly, the representation of Topology sub-view is based on UML component diagrams.

EIS design 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, since it augments usability for the system designer and performs constraint validation within as well as between views. Through this plug-in, external design tools can be invoked either by the system designer or automatically to enforce consistency between views.

3.5. Case Study

To evaluate its functionality, the proposed model-based EIS design approach was applied during the re-engineering process of the on-line system of a medium-sized bank, supporting 38 discrete transactions. The system architecture relies on server-based computing. A central database is installed in headquarters, while transaction logs are maintained in local databases of each branch. Transactions are coordinated by a transaction monitoring system – TMS (Tuxido), also installed in headquarters. To enhance security and ensure a single authentication point, all user programs run on a dedicated execution server (CITRIX), while in user terminals only the corresponding client (CITRIX client) is installed. Figure 4 presents a fraction of the Topology and Physical sub-views (snapshots from Rational Modeler environment). In this case, resource allocation and network design stages were performed by IDIS software tool [20].

The formal definition of system design sub-model resulted in the effective integration of autonomous software tools (Rational Modeler, IDIS) with no major coding effort. Furthermore, it enabled us to clearly identify the information necessary to effectively define system architecture, leading to suggestions regarding CITRIX server allocation (not necessary in small branches) and the processing power of Headquarters' servers. During system design, the designer should be able to analyze performance requirements imposed network infrastructure by application services. This was enabled through *Module-Process* interrelation and corresponding computation of *process* entity attributes. Since information system design was only part of bank's on-line system re-engineering, information from other system engineering activities had to be obtained. The definition of "external interrelation" entities, such as *site* entity has proven very useful. Though, sites (Headquarters and Branches) and the way they were interconnected where simply listen on paper (corresponds to first and second row of Network column in Zachman's matrix), we were able to obtain all information needed based site entity attributes. The process of obtaining information regarding *application modules* and *services* from *business process model cell* was more demanding, since a custom tool was used for business process description and information exchange was actually manually performed.

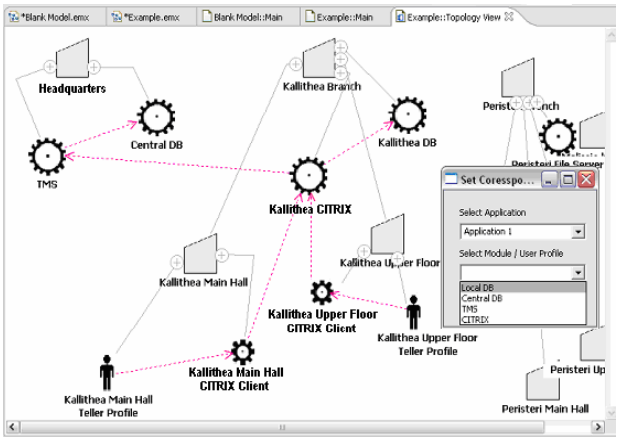

Figure 4a: Topology sub-view fraction

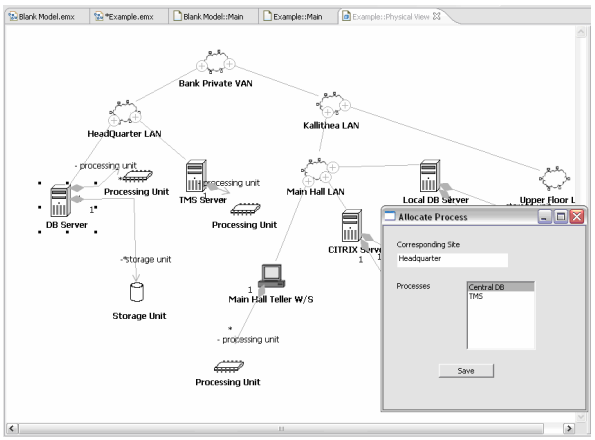

Figure 4b:Physical sub-view fraction

4. Conclusions

Model-based enterprise information system engineering based on the Zachman's framework was explored in the paper. Basic guidelines to incorporate model-based engineering methodologies within Zachman's framework were discussed. Proposed guidelines were easily applied in model-based EIS design, focusing on specific Zachman's matrix cells. They provided the "glue" to fit an autonomous design methodology within Zachman's matrix, using the notion of cell-related viewpoints.

5. References

[1] IEEE. "IEEE Recommended Practice for Architectural Description of Software-Intensive Systems - Std 1471". 2000.

[2] Zachman A. J. "A Framework for Information Systems Architecture". IBM Systems Journal, Vol. 31, No. 3, 1999, pp.445 –470".

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

[4] INCOSE. "INCOSE System Engineering Terms Glossary". October, 1998.

[5] Brown Allan. "Model driven architecture: Principles and practice". Software System Modeling, 3, 2004, pp.314–327.

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

[7] Leist S. & Zellner G. "Evaluation of Current Architecture Frameworks". Proceedings of SAC'06, April, 23-27, 2006, Dijon, France.

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

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

[10] Dijkman M., Quartel C., Pires L.F., Sinderen M.J. "An Approach to Relate Viewpoints and Modeling Languages". Proceedings of IEEE EDOC 2003.

[11] OMG Inc. "Unified Modeling Language: Superstructure". Version 2.1.1, 3/2/2007.

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

[13] Nikolaidou M., et. al. "Extending UML 2.0 to Augment Control over Enterprise Information System Engineering Process". Proceedings of the International Conference on Software Engineering Advances (ICSEA 2006), IEEE Computer Press.

[14] OMG Inc. "Systems Modeling Language (SysML) Specification". Version 1.0,4/6/2006.

[15] Gomaa H., Menasce D., Kerschberg L. "A Software Architectural Design Method for Large-scale Distributed Information Systems". Distributed System Engineering Journal, Vol. 3, No. 3, IOP, 1996.

[16] Graupner S., Kotov V., Trinks H. "A Framework for Analyzing and Organizing Complex Systems". Proceedings of the 7th International Conference on Engineering Complex Computer Systems, IEEE Computer Press, 2001.

[17] Nezlek G.S., Hemant K.J., Nazareth D.L. "An Integrated Approach to Enterprise Computing Architectures". Communications of the ACM, Vol. 42, No. 11, 1999, ACM Press.

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

[19] Nikolaidou M., Anagnostopoulos D. "A Systematic Approach for Configuring Web-Based Information Systems". Distributed & Parallel Database Journal, 17, 2005, pp. 267-290, Springer Science.

[20] Nikolaidou M., Lelis D., Mouzakis D., Georgiadis P. "A Discipline Approach towards the Design of Distributed Systems", Distributed System Engineering Journal, Vol. 2, No 2, 1996, IOP.