Model-based Enterprise Information System Architecture Design using SysML

Mara Nikolaidou, Anargyros Tsadimas, Dimosthenis Anagnostopoulos
Department of Informatics & Telematics, Harokopio University of Athens
70 El. Venizelou Str, 176 71 Athens, Greece
{mara, tsadimas, dimosthe}@hua.gr

***Abstract*—A model-based approach for the design of enterprise information system architecture is proposed in this paper. It facilitates the progressive refinement of system architecture based on a well-defined model and the systematic manipulation of information exchange with other methodologies addressing related engineering activities (for example software design). The Systems Modeling Language (SysML) is used for the representation of the proposed system model. The SysML profile, constructed for that purpose, is presented. The experience obtained when applying the proposed SysML profile in the renovation of a large-scale enterprise information system is also briefly discussed.**

I. Introduction

Enterprise Information Systems (EIS) are large-scale, composite systems, consisting of heterogeneous components often built in an autonomous fashion. Enterprise information system engineering relates to the efficient construction of the EIS and is characterized by extensive complexity, since the integration of different components and also strategies, methods and tools used to construct them, is necessary to define EIS structure and behavior. Many different stakeholders may be involved in this process, each of which focuses on certain concerns and considers them at a multiple level of detail [11]. Enterprise Architecture (EA) frameworks ([13],[16]) are considered as an attempt to integrate strategies, processes, methods, models and tools toward EIS engineering [1]. 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.). However, in practice, methods and tools applied by a specific stakeholder are supported by different system models, which in many cases are not compatible or even not known to others used by stakeholders with a different viewpoint.

Model-based EIS engineering can be defined as the process of specifying, designing, integrating, validating and operating an Enterprise Information System based on the development of a central model, extended in different levels of increasing detail [8]. The central EIS model can be defined as a collection of views and corresponding viewpoints serving specific stakeholders. In [14], the concept of using Zachman framework [17] as the basis for establishing a central EIS model for model-based EIS engineering was introduced. Zachman framework provides a holistic model of enterprise information infrastructure, focusing on 6 different perspectives serving discrete primary engineering activities according to Zachman matrix row rationale and 6 different aspects according to Zachman matrix column rationale. A plethora of methodologies and formalisms exist ([5],[9],[7]), each applicable to some subset of Zachman matrix cells, while respective system models are defined. Thus, it considered as the most suitable as the basis to construct the central EIS model. Rules governing the Zachman framework, as defined in [17], are applied during model-based EIS engineering as well. Each row may serve a model-based implementation of a discrete primary engineering activity, as defined in [12], addressing the needs of corresponding stakeholders [14].

According to the combination of Zachman's perspectives and aspects, model-based EIS engineering central model consists of 36 EIS sub-models. For example, RUP methodology [7] could be employed for application design within Function Design model corresponding to System Function cell. Since each model is treated autonomously, it should contain all necessary information to perform the respective tasks. Therefore, both internal entities, focusing on the specific model and external entities facilitating the integration with others, corresponding to other cells, construct each EIS sub-model. Each model gathers information from all the cells of the same row (participating in the same engineering activity) and the cells of the same column above and beneath it, while it also may pass information to them. Upper and lower cells participate in the progressive refinement of enterprise requirements for the specific aspect. In [14] we have proposed basic guidelines on a) how to identify the Zachman cell corresponding to a specific EIS engineering activity or specific methodology and b) to construct the corresponding EIS sub-model.

In this paper, we discuss a model-based approach for the design of EIS architecture based on the proposed guidelines. EIS architecture 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. It is usually performed by system architects. Determining system architecture is a complex process [3], which in essence focuses on the integration of EIS components (for example EIS software and data, EIS access points) already defined by other stakeholders than system architects. Since Zachman's system model row deals with design engineering activities, EIS architecture design should be handled within this row. Furthermore, architecture design is related to the network aspect of the system, as defined within Zachman framework. Thus, EIS Architecture Design View is handled within Zachman System Network cell. The proposed EIS sub-model should provide for the progressive refinement of EIS architecture based on a well-defined model constituting *EIS Architecture Design Model* that serves all architecture design tasks and information exchange with other methodologies addressing related EIS design issues (for example software design) corresponding to other Zachman cells. This is considered as a crucial issue, since in most cases the lack of proper information exchange between information system architecture design and software design methodologies results in poor system performance [10].

Systems Modeling Language (SysML) [4] may be used for the model-based design of EIS Architecture, since it supports the concepts of requirements and resource allocation, which are vital to depict EIS Architecture Design tasks, and is supported by most popular modeling tools. The rest of the paper is organized as follows: Section II explains the main concepts of model-based EIS architecture design identifying basic tasks and corresponding views. In section III, the corresponding SysML profile is analytically presented. In section IV, the way SysML profile is implemented and a small example are briefly discussed. The experience obtained when applying the proposed SysML profile in the renovation of a large-scale enterprise information system, indicating its potential, is commented in section V. Conclusions and future work reside in section VI.

II. EIS Architecture Design Model

The basic tasks identified during EIS architecture design are [14]:

1) Functionality Definition, consisting software architecture description. In practice, Functionality Definition consists of the description of functional requirements (e.g. application and data architecture, user behavior and application requirements).
2) Topology Definition, consisting of the description of system access points. It facilitates user, application and data allocation.
3) Network Infrastructure Definition, consisting of the description of platform-independent distributed infrastructure (e.g. network architecture and hardware configuration) and the association of software components to network nodes (resource allocation).
4) Non Functional Requirement (NFR) Definition, consisting of the description of non-functional requirements, focusing on system performance and availability requirements essential for EIS architecture design.
5) EIS architecture evaluation.

It is evident that all aforementioned tasks are interrelated, since non of them can be completed independently, while in most cases tasks are performed in parallel, and often repeatedly by the system architect in order to reach an EIS architecture satisfying both functional (identified during Functionality Definition and partly during Topology and Network Infrastructure Definition) and non-functional requirements (identified during NFR Definition). NFR Definition is performed in parallel with Functionality, Topology and Network Infrastructure Definition. Developing requirements and architectural artifacts in parallel has already been addressed in the literature [15]. After an EIS architecture has been defined, it should be evaluated, most commonly using simulation. Solution evaluation will determine whether a) the proposed solution is satisfying all functional and non-functional requirements, or b) the system designer should improve the proposed architecture or readjust requirements by repeating definition tasks. Thus, EIS architecture evaluation is treated as a discrete independent step, while the corresponding model should provide system designer with any information regarding system redesign or requirement readjustment decisions.

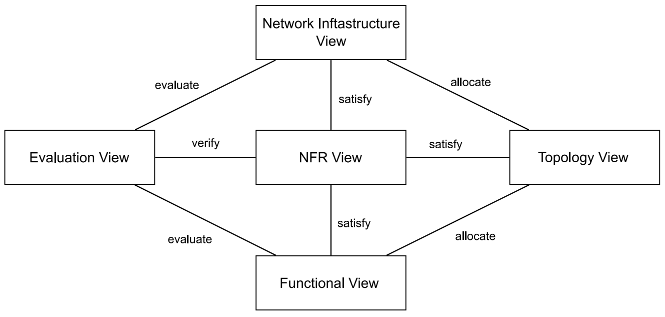

Fig. 1. EIS Architecture Design Model

The *EIS Architecture Design model* consists of discrete views, each of which is used to perform a specific design task, including solution evaluation. It facilitates concurrent execution of the five aforementioned tasks, while it promotes interoperability between them by strictly defining relations between corresponding views. Basic SysML concepts have been taken into consideration when defining the model.

Functional requirements and corresponding design decisions are described using complementary EIS Architecture views focusing on different aspects of system design, namely, *Functional View*, *Topology View* and *Network Infrastructure View*. In practice, each of these views serves the corresponding task identified above.

Using *NFR view*, the system designer is enabled to explore non-functional requirements relationships, while, using other views, the relationship between non-functional requirements and design decisions is explored [18]. Such an approach allows for the progressive and independent execution of EIS architecture composition tasks in parallel, while the impact of design decisions adopted in each of them to the other ones is expressed in terms of non-functional requirements grouped in NFR view. In this manner, the system architect is enabled to realize the affect of specific design decisions (for example the allocation of software to hardware resource) to non-functional requirements imposed to them (for example performance) and vise-versa.

The integration of the *Evaluation View* within *EIS Architecture Design model* facilitates the definition of solution evaluation scenarios and the integration of evaluation results into the common model in order to help the designer to make redesign or requirement readjustment decision if needed. It is comprised of evaluation scenarios. Each scenario represents an EIS Architecture configuration evaluated by the system architect and is based on the combination of entities belonging in Network Infrastructure View and Functional View. Scenarios are usually evaluated using Simulation. Evaluation results are also included in evaluation scenarios to facilitate the system architect to verify related non-functional requirements defined in NFR view. When conflicts are discovered, changes are made to the system configuration by the system architect (e.g. Functional, Topology, Network Infrastructure or even NFR view) and a new evaluation scenario is initiate by system architect until a satisfiable solution is reached.

Each view is treated as a discrete system component, while the relations between them are defined, using basic SysML relation types. All views comprising EIS Architecture Design model and the relations between them are depicted in the figure 1. In all aforementioned views, EIS corresponding aspects are described as a hierarchy of system components. In each view, both internal and external entities are defined. External entities depict information exchange with other stakeholders, for example the software architect. Corresponding external entity properties of Functional, Topology and Network Infrastructure Views depict the functional requirements used by the system architect to design EIS architecture. External entities of NFR view depict the NFRs derived from upper level requirements. External entities also indicate the requirements or constraints imposed by EIS architecture design decisions to other engineering activities. Only internal entities participate in the Evaluation View.

External entities indicate the information imported or exported from/to models serving other engineering activities, than EIS architecture design. Applying the methodology proposed in [14], each EIS engineering activity may be mapped into a specific Zachman cell, while the Zachman matrix itself provides basic rules for information exchange. The EIS model corresponding to a specific cell gathers information from all the cells of the same row (participating in the same engineering activity) and the cells of the same column above and beneath it, while it also may pass information to them. Upper and lower cells participate in the progressive refinement of enterprise requirements for the specific aspect. EIS view integration and inter-view consistency is accomplished by creating mappings between external entities of respective models. EIS Architecture Design model should provide for the exchange of information with model-based methodologies targeting design issues (Zachman system row), for example, data design, software design or design requirements, while it also contributes to the refinement of EIS structure (Zachman network column) by enabling the transformation of the Enterprise access points (Zachman business row) to a complete network architecture (Zachman builder row).

TABLE I
FUNCTIONAL VIEW

EIS Profile Entity	**Type**	**SysML Entity**
Role	external	Block
Initiate	internal	Dependency
Invoke	internal	Dependency
Data Entity	external	Block
Module (Client & Server)	external	Block
Service	external	Block

III. SysML Profile description

In the proposed SysML profile each view is depicted using a discrete diagram. The stereotype mechanism provided by UML and SysML is used to customize SysML functionality to depict EIS Architecture views. Functional, Topology and Network Infrastructure views are described using hierarchical block-definition diagrams. SysML blocks can be used throughout all phases of system specification and design, and can be applied to many different kinds of systems. These include modeling either the logical or physical decomposition of a system, and the specification of software, hardware, or human elements.

Functional view depicts functional requirements related to software components and related data, as well as EIS users. It also includes design decisions related to software architecture. *Roles* are used to depict the behavior of different user groups while *modules* (client & server) are application tiers that comprise of *services*. Each role *initiates* services that belong to client modules, and each service may *invoke* other services that belong to other modules, depending on the functionality of the application. *Data Entities* are used to represent portions of stored data. Main entities of Functional view, along with their type (external or internal) and the SysML base class they extend, are presented in table I. Entities participating in Functional view are related to entities participating in all other diagrams to implement the relations depicted in figure 1. These relations are discussed in the following paragraphs as the rest of the views are briefly presented.

Topology view facilitates the description of system access points in terms of hierarchically related locations, called *sites*. Sites may be *atomic* or *composite* (meaning that are composed of other sites). *Site* entity is an extension of SysML Block entity. Topology View is a Block Definition Diagram that comprises the aforementioned entities. Topology View entities are presented in table II. Entities defined in other views (e.g. Functional and NFR view) may also participate in the diagram, in order to describe Topology view interrelation with other diagrams, as define in figure 1. *Software Allocation* is used to describe the allocation of software modules (client or server) to *atomic sites*, while *Usage Allocation* refers to roles that are allocated to atomic sites. Sites satisfy *traffic requirements*, indicating the amount of information exchange between the modules allocated to them. A traffic requirement is described in terms of traffic coming in, going out and exchanged within each site. Traffic requirements are entities defined in NFR

TABLE II
TOPOLOGY VIEW

EIS Profile entity	Type	SysML Entity
Site (Atomic & Composite)	external	Block
Software Allocation	internal	Allocation
Usage Allocation	internal	Allocation

TABLE III
NETWORK INFRASTRUCTURE VIEW

EIS Profile entity	Type	SysML Entity
Network (Atomic & Composite)	external	Block
Atomic-Network Diagram	external	Block Definition Diagram
Structural-Allocation	external	Allocation
Server	external	System
Workstation	external	System
Network-Device	external	System
Connection	external	Association
PTP-Connection	external	Association
Software-Allocation	internal	Allocation
Usage-Allocation	internal	Allocation

TABLE IV
REQUIREMENTS VIEW

EIS Profile entity	Associated Elements
Constraint-Req	Network-Device, Connections, Server, Workstation
Load-Req (derived)	Network, Server, Workstation
Availability-Req	Server, Workstation
Traffic-Req (derived)	Site
Utilization-Req	Network, Server, Workstation
Service-QoS-Req	Service
Module-QoS-Req (derived)	Module
Response-Time-Req	Service
Behaviour-Req	Role

view. No SysML extension were needed to represent requirement satisfaction relationship. Constraints were used a) to constraint SysML functionality, for example *only modules may be functionally allocated sites* or *atomic sites are allocated only to atomic networks and composite sites or networks have to own at least one atomic element* and b) to compute values of derived entity attributes, for example, *traffic requirements attributes of a specific site are automatically computed from Module-Qos requirements attributes of modules allocated to this site*.

Network Infrastructure view refers to the aggregate network, described through a hierarchical structure comprising of simple and composite *networks*. It is represented using a hierarchy of block diagrams. Hardware components and configurations are also defined using this view (*servers, workstations and network devices*). Networks are inter-connected with *PTP-connections*. Sites are allocated to networks using *Structural Allocations*. Each atomic network owns a specific diagram (Block Definition Diagram), called *atomic network diagram*, which encompasses all hardware elements that belong to that network. Main Network Infrastructure View entities are presented in table III. Most of them are characterized as external entities, since the should be further refined during network implementation by the system constructor. Existing Network infrastructure is depicted using constraint requirements defined in NFR view and associated to appropriate network components in Network Infrastructure view. Elements of Functional, Topology and NFR views may also participate in Network Infrastructure view to represent inter-view relations.

As seen in figure 1, NFR view comprises requirements that are satisfied by entities of the three aforementioned views and are verified by elements of the Evaluation View. Table IV presents main requirements and the related entities that satisfy them. All requirements are defined as stereotypes of SysML requirement entity, while additional stereotype attributes are defined to accommodate specific requirement properties. Requirements may be derived from other requirements, while all of them are treated as internal entities, since they are defined on the context of EIS architecture design. Non-functional requirements Requirements in SysML are described in an abstract, qualitative manner, since they are defined using a name and a description. In the case of EIS Architecture Design, non-functional requirements should be more accurately describe using quantitative properties. Furthermore, derived requirement properties should be automatically computed by combining specific attributes of requirement and allocation entities. Though, SysML provides for non-functional requirements description, SysML requirement entity was heavily extended to effectively represent the quantitative aspects of requirements and the way they derive from each other. Further analysis on NFR view can be found in [18].

Evaluation view is introduced to aim at a)defining specific EIS Architecture configurations, which should be evaluated, and b)storing the evaluation results of different configuration scenarios. This knowledge should be available to the system architect during EIS architecture redesign. In practice, it is used to determine whether the proposed architecture meets specifications placed by non-functional requirements. Since EIS Architecture design process may require to evaluate and properly adjust the proposed architecture more than once, Evaluation View consists of multiple test cases used to evaluate alternative solutions. Since simulation is used for architecture evaluations, these test cases are called *simulation experiments*. A Simulation Experiment is a set of conditions or variables which will be tested to ensure requirements are met. As indicated in figure 1, a simulation experiment is conducted to evaluate design decisions depicted in Functional and Network Infrastructure View, while its results are used to verify requirements defined in NFR View. When conflicts are discovered, changes are made to the system configuration by the system architect (e.g. Functional, Topology, Network Infrastructure or even NFR view) and a new simulation experiment is initiate by the system architect until a satisfiable solution is reached.

Though SysML provides the *test case* entity and corresponding behavior diagrams for system evaluation, it was decided to represent Evaluation view using a set of block

TABLE V
EVALUATION VIEW

EIS Profile entity	SysML Entity
Sim-Experiment	Block Definition diagram
Sim-Service	Block
Sim-Module	Block
Sim-Role	Block
Sim-Workstation	System
Sim-Server	System
Sim-Node	System
Sim-Network-Device	System
Sim-Connection	System
Sim-PTP-Connection	System
Sim-LAN	Block
Usage-Allocation	Allocation

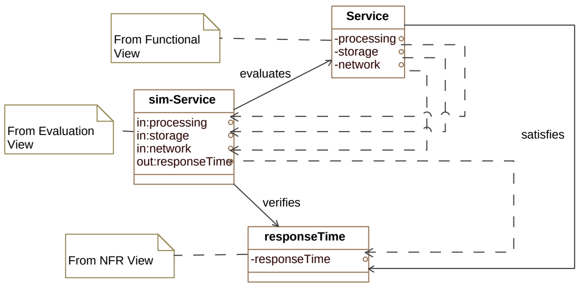

Fig. 2. Sim-Service entity description

definition diagrams, one for each test case. Main entities are described in V. All of them are considered as internal entities. Each simulation experiment entity is created to evaluate a specific EIS Architecture entity and verify corresponding non-functional requirements. For example, a *service* has to satisfy a *responseTime* requirement indicating maximum execution time. This requirement must be verified by *sim-Service* entity, as represented in figure 2. Simulation entities have input and output attributes. Input attributes correspond to initialization conditions passed to the simulation environment and are derived from corresponding attributes of evaluated entities. Output attributes indicate simulation results. To verify a requirement, the system designer should compare output attributes to corresponding requirement attributes, to check if there is a conflict. For example, the *ResponseTime output attribute* of sim-Service is compared to *responseTime requirement* of corresponding *service* entity. If a conflict has been identified, the system designer should alter the system design (e.g. modify the network architecture or the requirement itself) using Functional and NFR views and conduct a new experiment.

IV. SysML Profile Implementation

The profile is implemented as a plugin to MagicDraw modeling tool [2], which provides full SysML support. System designer defines Functional, Topology, Network Infrastructure and NFR views within the design environment. Simulation experiments in the Evaluation view, are automatically created based on the content of Network Infrastructure and Functional views. Simulation experiments are executed using a simulation tool. Entities included in the simulation experiment diagram and their input attributes are used to properly initialize simulation, while simulation results are passed to simulation experiment diagram (each one to the corresponding simulation entity) in order to verify the requirements. The bidirectional information exchange between the profile implemented in MagicDraw tool and the simulation environment are currently under implementation.

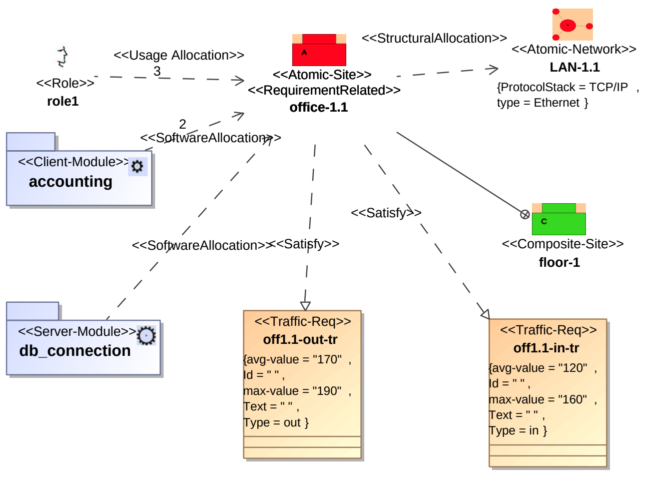

Fig. 3. Topology View example

Figure 3 presents a Topology View example, as defined using MagicDraw. Entity *offfice-1.1* is an atomic site, which belongs to *floor-1* composite site. Role *role1*, client module *accounting* and server module *db_connection*, already defined in Functional view, are allocated to *offfice-1.1* atomic site. The number of users and software module replicas allocated in the specific site are indicate as properties of Usage_Allocation and Software_Allocation relations respectively. Traffic requirements (*off1.1-in-tr* & *off1.1-out-tr*), defined in NFR view, must be satisfied by *offfice-1.1* atomic site. The allocation of *office-1.1* to the atomic network *LAN1* is also depicted in the diagram.

V. Case Study

The SysML profile for EIS Architecture design was applied combined with RUP in a case study where the renovation of the legacy system of a public organization was explored. Alternative software architectures and their implications to hardware/network infrastructure were evaluated. One of the main objectives of legacy system architecture re-design was to enhance application performance without major rewriting the applications themselves. Since performance play a significant role, it was suggested to apply the proposed SysML profile, to explore related design decisions and evaluate them. System architects had to explore two different scenarios: a) to support existing distributed database architecture and try to consolidate hardware and b) to establish a central database architecture resulting in minor applications code modifications. Both scenarios were explored. The organization supports more than

350 interconnected regional offices technologically supported by a central IT Center responsible for IT diffusion and management. Regional offices are divided into three categories according to their size, structure and personnel (large, medium and small). Each category is treaded differently in terms of network infrastructure requirements.

Since application functionality is well-known, the identification of software architecture and performance requirements was perceived as a trivial task. To obtain this information the system designer had to communicate with application maintenance personnel in the corresponding department of the IT Center. RUP methodology was used for software development, thus application description models were developed within Rational Rose platform. Application description (e.g. modules and services) as well as data structures were manually extracted from corresponding Rational Rose [6] files. Though the process was not automated, Functional View external entity properties, enabled the system architect to easily obtain the necessary information. Unfortunately, the identification of service performance requirements was not a straightforward procedure, since software maintenance personnel was not able to accurate provide either response time or service QoS information. Response time requirements were finally defined by system architects, while service QoS information were obtain after monitoring application functionally during working hours by system administration personnel in the current version of the system. Service QoS requirement accurate definition was essential for the effective exploration of application performance based on alternative architecture scenarios.

The network architecture was predefined. The system architect was enabled to explore the two different database architecture scenarios. The system designer modified software allocations in the Topology view and corresponding load requirements satisfied by the network infrastructure were automatically computed using the profile. System performance was evaluated using an existing custom simulator, supporting the same EIS structure as the simulation experiments of the Evaluation view. Simulation results were manually inserted in corresponding Simulation Experiment entities by the system designers. Though, they consisted their incorporation within the profile a positive aspects, since this enabled them to keep track of all redesign decisions and the reasons leading to them. It was estimated that software performance was improved almost by one third by the second scenario, thus, it was decided to apply it although it involved minor application rewriting.

VI. Conclusions

A model-based approach using SysML to design EIS Architecture was explored in the paper. The provision of a well-defined model to perform all design tasks, including system evaluation and adjustment is essential for the system designer, while the characteristics of alternative design decisions should also be integrated in the model. Dealing with non-functional requirements and their verification using Evaluation view enhances the system designer's perception of the way specific design decisions may affect others.

SysML modeling language efficiently supported model-based EIS Architecture design, as it provided the means to accurate depict discrete architecture design views and their relation. EIS description as a system of systems and resource allocation, provided by SysML, enabled the straightforward description of EIS desired functionality. Non-functional requirement description resulted in a heavy extension of SysML requirement entity using both addition attributes and complex constraints, which can not be implemented using OCL. Evaluation View was treated as a hierarchy of block diagrams, since it is used as the means to store evaluation results, and not to describe how this evaluation should be performed as described by SysML test case behavioral diagrams. The proposed profile is currently tested using real-world case studies.

References

[1] "Institute For Enterprise Architecture Developments," http://www.enterprise-architecture.info/.

[2] "MagicDraw UML," http://www.magicdraw.com/.

[3] "INCOSE Handbook SE Process Model," INCOSE, September 2003, http://g2sebok.incose.org/.

[4] "Systems Modeling Language (SYSML) Specification. Version 1.0," O. M. G. Inc, September 2007.

[5] B. Dave and D. Jim, "The new, improved RUP SE Architecture Framework," 2005, iBM Rational Edge.

[6] b. dave, d. jim, and S. J. Vaughan-Nichols, "Web services," pp. 18–21, 2002, ibm rational edge.

[7] D. J. de Villiers, *Using the Zachman Framework to assess RUP*, Rational Edge, 2001.

[8] J. A. Estefan, *Survey of Model-based Systems Engineering (MBSE) Methodologies*, INCOSE MBSE Focus Group, May 2007.

[9] A. Fatolahi and F. Shams, "An investigation into applying UML to the Zachman Framework," *Information Systems Frontiers*, vol. 8, no. 2, pp. 133–143, 2006. [Online]. Available: http://dx.doi.org/10.1007/s10796-006-7977-8

[10] S. Graupner, V. E. Kotov, and H. Trinks, "A framework for analyzing and organizing complex systems," in *ICECCS*. IEEE Computer Society, 2001, p. 155. [Online]. Available: http://csdl.computer.org/comp/proceedings/iceccs/2001/1159/00/11590155abs.htm

[11] IEEE, "IEEE System and Software Engineering - Architectural Description: Std 42010," Tech. Rep., May 2009.

[12] *IEEE Std 15288 -2004, Systems Engineering -System Life Cycle Processes*, Institute for Electrical and Electronic Engineers, June 2005.

[13] S. Leist and G. Zellner, "Evaluation of current architecture frameworks," in *SAC*, H. Haddad, Ed. ACM, 2006, pp. 1546–1553. [Online]. Available: http://doi.acm.org/10.1145/1141277.1141635

[14] M. Nikolaidou, N. Alexopoulou, A. Tsadimas, and D. Anagnostopoulos, "Employing zachman enterprise architecture framework to systematically perform model-based system engineering activities," in *Hawaii International Conference on System Sciences (HICSS-42), Waikoloa, Big Island, Hawaii, January 5-8, 2009*, 2009.

[15] K. Pohl and E. Sikora, "Supporting the Co-Design of Requirements and Architectural Artifacts," in *15th IEEE International Requirements Engineering Conference (RE'07)*, India Habitat Center, New Delhi, 2007, pp. 258–261.

[16] J. Schekkerman, *How to Survive in the Jungle of Enterprise Architecture Frameworks: Creating or Choosing an Enterprise Architecture Framework*. Trafford, 2003.

[17] J. F. Sowa and J. A. Zachman, "Extending and Formalizing the Framework for Information Systems Architecture," *IBM Systems Journal*, vol. 31, no. 3, pp. 590–616, 1992.

[18] A. Tsadimas, M. Nikolaidou, and D. Anagnostopoulos, "Handling non-functional requirements in information system architecture design," in *The Fourth International Conference on Software Engineering Advances ICSEA 2009, September 20-25, 2009 - Porto, Portugal.*