Facilitating Enterprise Information System Engineering through a UML 2.0 Profile: A Case Study

M. Nikolaidou[1], N. Alexopoulou[12], A. Tsadimas[1], A. Dais[1], D. Anagnostopoulos[1]
{mara@di.uoa.gr, nancy@hua.gr, tsadimas@hua.gr, adais@hua.gr, dimosthe@hua.gr}

[1] *Harokopio University of Athens,*
El. Venizelou Str, 17671Athens, Greece

[2] *Department of Informatics and Telecommunications,*
University of Athens, Panepistimiopolis, 15771, Athens, Greece

Abstract

Modern enterprise information systems are distributed systems usually built on multi-tiered client server architectures and can be defined using well-established frameworks such as the Zachman framework or the Open Distributed Processing Reference Model (RM-ODP). Both frameworks identify views regarding the system designer's viewpoint, but they do not suggest a methodology for view creation. A consistent framework for enterprise information system engineering, compatible with both the Zachman framework and RM-ODP is proposed by the authors. It consists of a metamodel describing alternative system views, a corresponding methodology comprising discrete stages performed either by the system designer or software tools and a UML 2.0 profile for view representation. In this paper, a case study where the proposed framework was applied is discussed, focusing on the features provided to the system designer using the UML 2.0 profile. The profile is implemented by extending the Rational Software Modeler functionality.

1. Introduction

When building an enterprise information system (EIS), the desired properties of the system should be defined, such as its structure and behavior, while the role of the system in the environment should also be considered. Many different stakeholders may be involved in this process, as defined in the Zachman framework (Zachman, 1999). Each of these stakeholders focuses on certain concerns and considers these concerns at a certain level of detail. A *viewpoint* defines the way the system is conceived by a stakeholder according to his concerns (Boer, 2004). The conception of the system according to a certain viewpoint is described as a system *view*, thus one or more views correspond to a certain viewpoint. Each view may be formally defined by a *model*, while it is 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). A consistent representation of the systems implies that each view is not examined in isolation but interrelations between views are formally defined. We argue that the way system views are related must be fully and typically defined in the corresponding models. 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 (e.g. UML), in a common manner, facilitating the transformation between representation modeling languages.

Having adopted this viewpoint-oriented description of information systems, we defined for the system enginnering viewpoint three complementary views, namely *Functional, Logical* and *Physical.* Some of them may be further decomposed into subviews emphasizing specific entities into a greater level of detail. These views are part of a framework introduced in (Nikolaidou et al., 2006) which offers a consistent framework for information system engineering. More specifically our framework comprises:

- A metamodel describing different views and the relations between them (EIS metamodel). These relations are strictly defined using constraints.

- A methodology for EIS engineering based on the proposed views. The methodology consists of discrete stages performed by 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 automatically, as a result of the metamodel constraint validation.
- A UML representation for all defined views. A UML 2.0 profile is defined for this purpose (EIS engineering profile).

The overall framework is briefly presented in section 2, emphasizing the supported views and the corresponding UML 2.0 profile. In section 3, a case study where the proposed framework was applied is discussed, focusing on the features provided to the system designer using the UML 2.0 profile. A Rational Software Modeler plug-in has been implemented to support the additional functionality of the profile.

2. EIS Engineering Framework

The framework is based on three complementary views:

Functional View is used to describe functional specifications such as system architecture, user behavior and application requirements. System architecture refers to the architectural model adopted. In case of our framework, multi-tiered client-server models are described. Services provided by each application tier (called module) are also defined. User behavior is modeled through user profiles describing 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* subview.

Topology View facilitates the definition of system access points and the resource allocation and replication. To characterize any location (i.e. a building, an office, etc.), the term *site* is used. 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.

2.1 EIS Engineering Methodology and Metamodel

The proposed methodology includes the following discrete stages of the system engineering process:

1. System requirements 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. 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) and (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 et al., 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.

The proposed methodology stages along with the EIS model consisting of the predefined views are presented in figure 1. 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. Requirements 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, if any, the existing network architecture (Physical View). Each view is represented by one or more UML diagrams properly extended. All the required extensions are grouped into a UML 2.0 profile which also describes the relations between views.

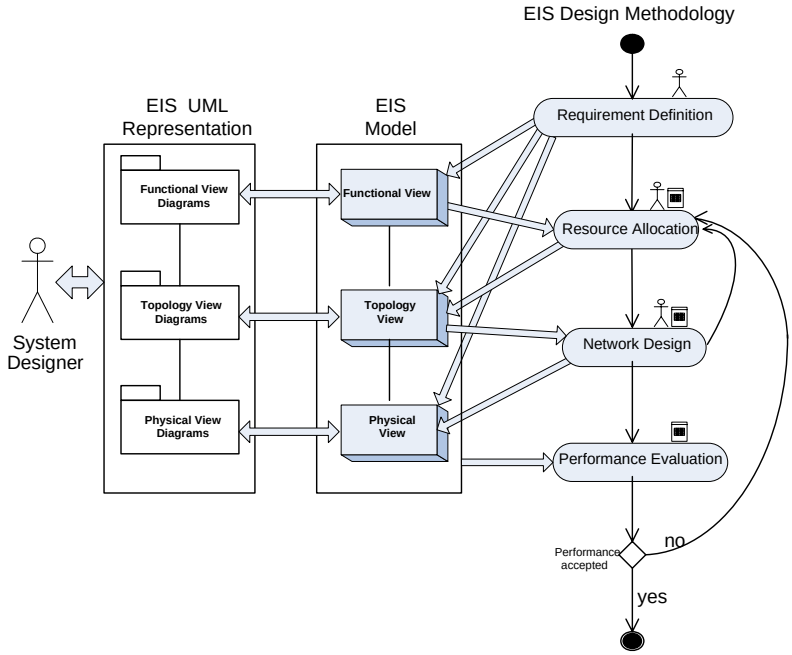

Figure 1: EIS Engineering Framework

As already mentioned the models created follow a formal metamodel which 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 Physical View is modified, this modification must be depicted in Topology View as well). Embedding restrictions within the metamodel facilitates the management of the EIS engineering, as the overall system model is taken into account 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.

All the entities of the metamodel along with their interdependencies are presented in figure 2. As shown in figure 2, despite the fact that views concentrate on different aspects and thus include different model elements, there are however correspondences between them indicated in the diagram by the lines that cross view boundaries.

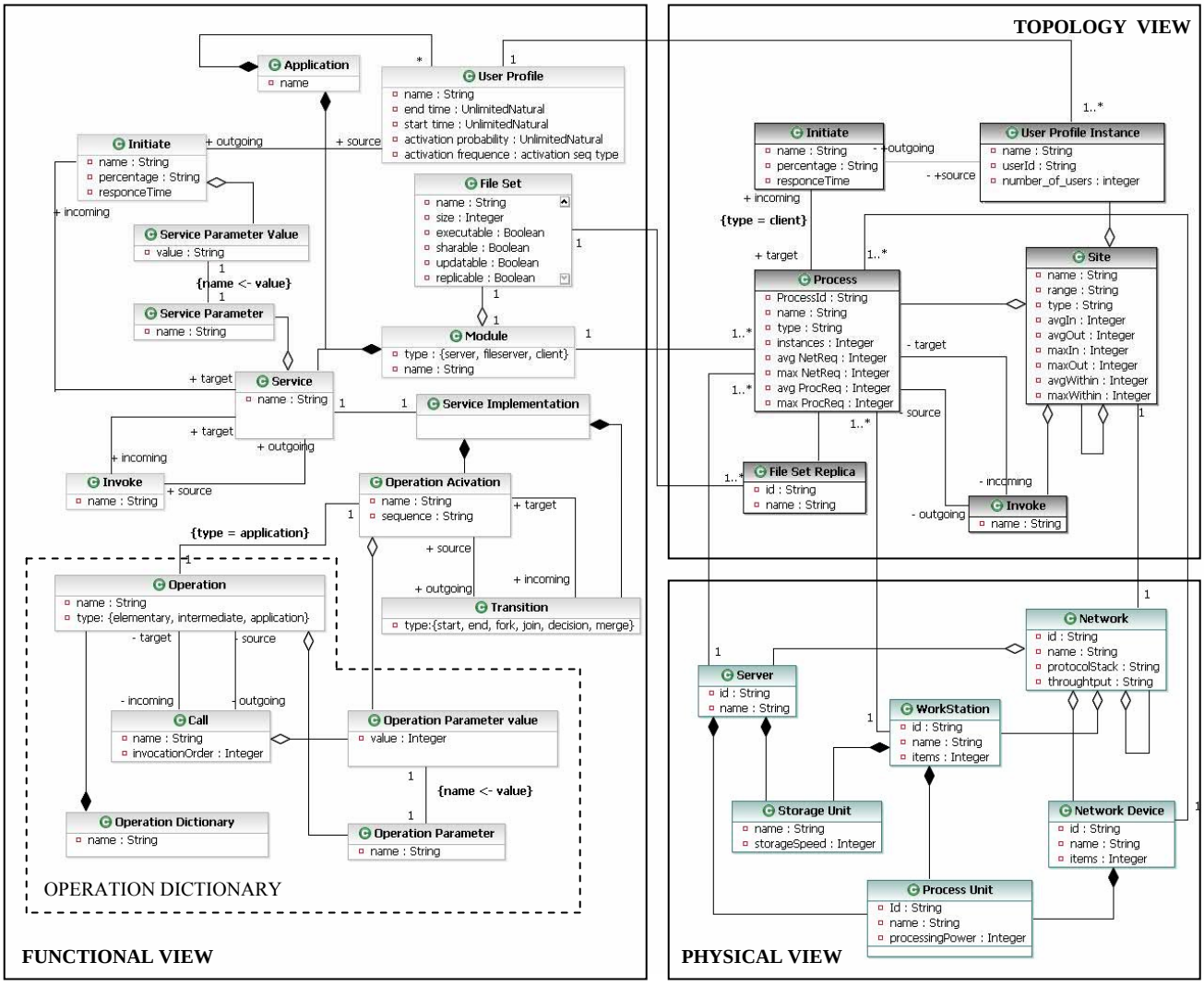

Figure 2: EIS Engineering Metamodel

2.2 EIS Engineering UML 2.0 Profile

The defined UML 2.0 (OMG 2004a; OMG 2004b) profile comprises a number of stereotypes. Essentially, the concepts of the metamodel are reflected onto the stereotype attributes and constraints. Attributes convey the information required to describe the 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:

1) automatic computation of specific attribute values.
2) limiting attribute value range.
3) 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).
4) model validation in view and overall model level.

Attributes and constraints for each stereotype are analytically introduced in (Alexopoulou et al, 2006). Following, the UML diagrams selected for each view are briefly presented. Stereotypes are listed in Figure 2 along with the EIS metamodel entity they correspond to. The relative icons are also included, so that the

reader can understand the figures presented in the case study of section 4. Functional view is represented through UML component diagram, since component diagrams are eligible for depicting system functionality at a logical level. Concerning service description subview, it is represented through activity diagram, as it involves flow of operations. UML communication diagrams, which depict interaction between entities, are suitable for the representation of Operation Dictionary, since the latter involves interactions between operations showing in particular invocation order and parameter passing between them. Physical View, which comprises the network infrastructure, is illustrated through UML deployment diagrams, which are commonly used to represent network architectures (Kaehkipuro, 2001). Lastly, the representation of Topology View is based on UML component diagrams.

FUNCTIONAL VIEW		
Stereotype	**EIS Metamodel Entity**	**Notation**
ServerModuleComponent	Server Module	
FileServerModuleComponent	Server Module	
ClientModuleComponent	Client Module	
ServiceComponent	Service	
Invoke	Invoke relationship between Services	
UserProfileComponent	User Profile	
Initiate	Initiate relationship between User Profiles and Services of Client Modules	
Operation Dictionary		
OperationAction	Service operation	
ElementaryOperationLifeline	Elementary Operation	
IntermediateOperationLifeline	Intermediate Operation	
ApplicationOperationLifeline	Application Operation	
ArgumentsMessage	Message sent between operations conveying parameter values	

PHYSICAL VIEW		
Stereotype	**EIS Metamodel Entity**	**Notation**
NetworkPackage	Network	
ServerDevice	Server	
WorkstationDevice	Workstation	
ProcessUnitDevice	Process Unit	
StorageUnitDevice	Storage Unit	

TOPOLOGY VIEW		
Stereotype	**EIS Metamodel Entity**	**Notation**
SitePackage	Site	
ServerProcessComponent	Server Process Instance	
ClientProcessComponent	Client Process Instance	
UserProfileComponent	User Profile Instance	
DataComponent	Data Entity	
Initiate	Initiate relationship between User Profile Instances and Client Process Instances.	
Invoke	Invoke relationship between Server Process Instances	

Figure 3. Stereotypes of the EIS Engineering Profile

3. Case Study

The proposed framework has been applied for the engineering of a typical banking system. In this case, resource allocation and network design stages were performed by IDIS software tool (Nikolaidou, 1999), that supports the representation and exploration of resource allocation and network topology design through algorithms combining mathematics and rules of thumb. To evaluate distributed system performance, the discrete event simulation tool described in (Nikolaidou, 2003) was used. Requirements definition was performed by the system designer using the EIS engineering UML 2.0 profile, implemented in Rational Modeler (IBM Co, 2005).

An appropriate UML modeling tool for EIS engineering UML 2.0 profile implementation must fulfill the following requirements: a) it must be UML 2.0 compatible, b) it must facilitate mechanisms to extend the provided functionality (e.g. by importing profiles) and c) it must export models in XML based on existing UML classes and profile-specific stereotypes. After serious considerations regarding various UML 2.0 tools,

we decided to implement the profile in the Rational Software Modeler environment (IBM Co, 2005). The extensibility features of the Rational Software Modeler are based on the open-source Eclipse platform. Eclipse provides useful APIs, frameworks (e.g. Workbench, Workspace, Help, etc.) and plug-ins that facilitate the development of new tools. EIS models are stored in an XML format in accordance with XMI (OMG 2005) to ensure interoperability. Before using a specific tool, the partial transformation of EIS model into the tool-specific internal model is realized. Using this transformation, the invocation and initialization of any tool can be automatically performed. The case study focuses on requirements definition and aims at demonstrating the use and implementation of EIS engineering profile through Rational Modeler. However, hits on the overall framework functionality are provided.

The Bank supports 38 discrete teller transactions. The amount of transactions/day varies according to branch size, while the average amount of teller transactions in large branches is over 10.000 per day. The required response time is 15-18 sec for most 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. Transactions are composed by 24 discrete atomic transactions initiated by TMS. Each transaction consists of 3 to 7 atomic ones. All atomic transactions are implemented by stored procedures running in the central database. 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.

Functional View

Functional view facilitates the system designer to a) define the EIS architecture (client and server modules) and b) define the functionality provided by its modules and the requirements imposed by them and the interaction between them to the network infrastructure.

EIS modules identified were the following: *File Server*, *CentralDB, LocalDB*, *TMS* and *Citrix*. Since *LocalDB* represents logging, only a simple *insert* service was implemented for recording the log. *CentralDB* supports 33 stored procedures, represented as a different service. *TMS* Module includes 24 services corresponding to discrete atomic transactions. *Citrix* Module includes 38 services corresponding to discrete teller transactions. They involve the invocation and processing of forms, the activation of atomic transactions through *TMS* and log recording. Tellers are modeled as User Profiles initiating *CITRIX Client* modules corresponding to each teller transaction. In the following, we focus on teller transaction to demonstrate real-world system representation capabilities of the proposed framework.

Figure 4 represents a fraction of Functional view, implemented as a Component diagram in Rational Modeler, emphasizing services needed for the representation of transactions trx31600 (i.e. cash deposit) and trx2000 (i.e. request business loan). As depicted in the figure, services are represented as component stereotypes and modules as package stereotypes. The *trx31600* service of the *Citrix* Service Module is selected in the figure. Additional stereotype attributes are stored in the corresponding fields supported within Rational Modeler platform (bottom right part of figure 4). Input parameters of each service are added by system designer through a custom menu created using Rational Modeler Eclipse API. In this case (*trx31600*), only the *module* attribute is filled, since the service has no input parameters (*inputParameterList* attribute is empty). On the left part of Rational Modeler's screen in figure 4, it is shown that *trx31600* service component is further decomposed into other entities.

Trx31600 service is described by the corresponding activity diagram, implemented as a subdiagram of the Functional view component diagram. It is represented in figure 5. As shown in the figure, trx31600 includes the activation of the appropriate forms (operation action 1), the activation of the central database through the TMS (operation action 2 and 3) and local database update (operation action 4). Each discrete step is represented by an action instantiating a predefined operation included in the Operation Dictionary. Operations represent requirements imposed to system resources (network, processing nodes, etc). When defining an action, all input parameter values of the corresponding operation must be filled. They must be either constant

or already defined as trx31600 service input parameters. As shown in figure 5, all operation input parameters must be constant, since trx31600 service has no i*nputParameterList*. The corresponding validation constraint is implemented as a custom script initiated by Rational Modeler's *Run Validate* default menu appearing when right-clicking on any UML diagram entity. Some of the actions, as *request* (selected in figure 5), result in the invocation of other services.

A constraint automatically adds the corresponding invoke entity between the relative service components of Functional View (figure 4). The invoke entity has the same name as the action.

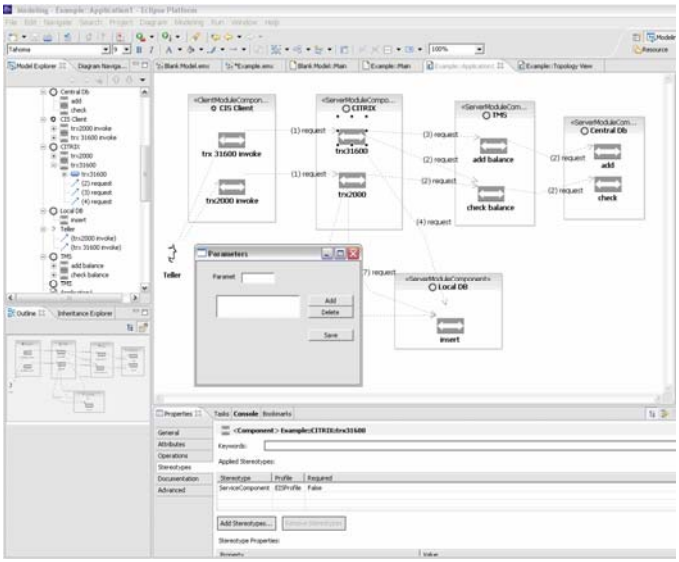

Figure 4: Fraction of Functional View – Transactions trx31600 and trx2000

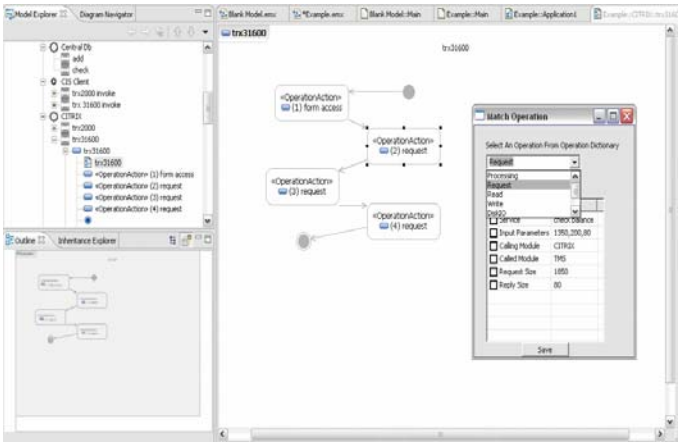

Figure 5: Trx31600 activity diagram

Operation Dictionary

Figure 6 represents a fragment of the operation dictionary. All operations are decomposed into elementary ones (processing, storing, transferring), representing processing, storing and network requirements. The system designer may add new operations in the dictionary, to enhance operation expression.

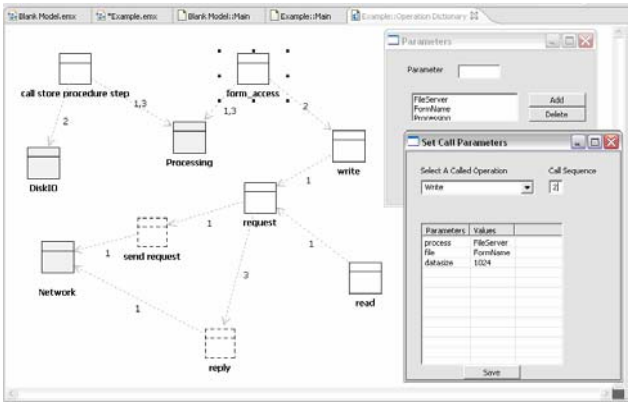

Figure 6: Operation Dictionary Fragment

In figure 6, the addition of *form_access* operation is presented. Three steps should be accomplished: parameter definition, definition of dependencies to existing operations and validation performance. A related constraint checks if all the parameters defined for an operation are passed as values to called operations used for its execution. Parameter and dependency definition is performed through pop-up forms. *Form_access*

operation parameters are *FileServer, form_name* and *processing. Form_access* operation "uses" two other operations in order to be executed: *processing* and *write*. First, calls *processing* (which is elementary operation) and then *write* and then again *processing*. Parameter values of the called operation must be defined. The pop-up window entitled *Set Call Parameters* depicts *write* operation parameter definition.

Topology and Physical Views

The Topology View facilitates process and user profile allocation to sites. Allocation is performed by the designer through Rational Modeler interface. Alternative, the designer may invoke *IDIS* to perform the allocation of processes or data. Three different types of branches are supported: large, medium and small. Large branches have more than 30 tellers stationed at two different floors. The upper floor is dedicated to business transactions (10 tellers), while all others are served in the main hall. The corresponding fraction of Topology View is depicted in figure 7. Each hall is represented as a subsite of a branch site (both represented as *Site Packages*). Headquarters is also presented as a site.

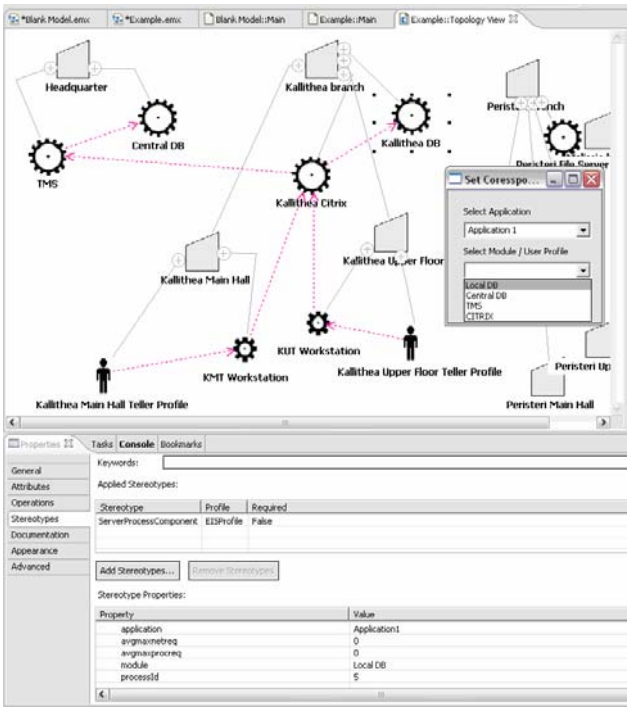

Figure 7: Fraction of Topology View

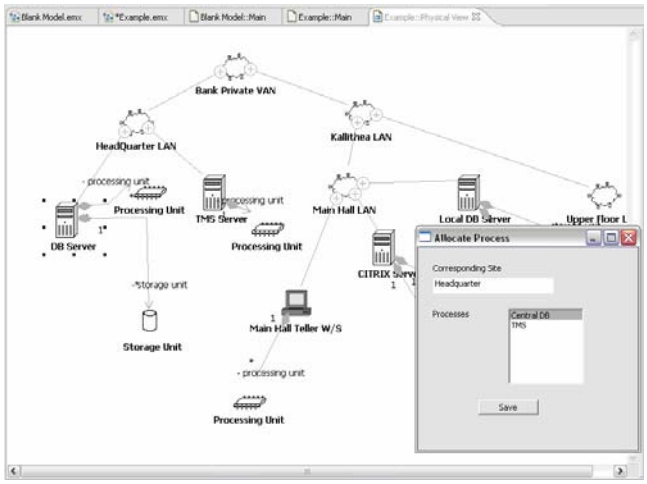

Figure 8: Fraction of Physical View

Tellers, modeled as *users*, are placed in *Main Hall* and *Upper Floor* sites, along with corresponding *Citrix client* processes. Since the system relies on server-based computing, most server processes are placed only in headquarters, while no replication is employed simplifying the overall architecture. Furthermore, since there was a request to maintain log data in local branch databases, a local database server replica is placed in each branch. The only issue to be explored was the placement of *CITRIX Server*. Although the system designer placed a *CITRIX Server* process in each branch, the logical configuration tool removed the processes from medium and small branches and placed one in Headquarters to minimize communication cost. This is codified in the EIS model stored in XML. When this model is loaded again in the UML tool, Topology View appears automatically updated.

Processes and users appearing in Topology View must correspond to EIS modules and user profiles represented in Functional View. As shown in figure 7, when defining process replicas, a shortcut menu containing two drop-down lists appears. The first one corresponds to the application (described by a discrete Functional View) and the other one to the module (defined within the Functional View). Furthermore, the corresponding relationships between processes and modules must be defined in both diagrams. The properties

of server process stereotype are shown at the bottom part of Rational Modeler Screen. In figure 7, the property values of *Kallithea DB* server component are shown. A related constraint is activated by the *Run Validation* menu option.

Physical View is rather trivial. It facilitates network design and is performed by the designer through Rational Modeler interface. Alternative, the designer may invoke *IDIS* to perform this task. A fraction of it is presented in figure 8. The overall network is TCP/IP based. Branches are connected to headquarters using leased lines, forming a private WAN. The connection speed is indicated as the name of membership relation between node devices and site packages. As indicated in the figure, branches are internally supported by switched 100BaseT Ethernet. The structure of Physical View in the banking system (network architecture) was predefined. As shown in the figure, the system designer may define the processes running on a node through a pop-up window. The candidate processes for a server node must belong to the corresponding site and be server processes. Network hierarchy must correspond to site hierarchy and vice versa. Thus, when validating the model presenting in figure 8, an additional site (corresponding to the Bank Private WAN) should be automatically added in Topology View of figure 7.

4. Conclusions

A consistent framework for EIS engineering was introduced. It consists of a metamodel describing alternative system views and the relations between them, a corresponding methodology comprising discrete stages performed by the system designer or software tools and a UML 2.0 profile for view representation. The main advantage of the proposed framework is the formal definition of views and their consistent UML 2.0 representation. This is accomplished using constraints in both the metamodel and the UML profile. As proven by the presented case study, constraints play an important role in the consistent representation of the system under study, since they impose the necessary restrictions and relationships between entities participating in different views. The proposed framework is currently tested in terms of completeness and expressiveness, using large-scale EIS architectures as test cases.

5. Acknowledgement

This research was supported in part by Pythagoras program (MIS 89198) co-funded by the Greek Government (25%) and the European Union (75%).

6. References

Alexopoulou N., Nikolaidou M, et al, 2006. "Introducing a UML Profile for Distributed System configuration", in proceedings of *ICEIS 2006*.

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

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

IBM Co, 2005. Introducing Rational Software Modeler, http://www-128.ibm.com/devworks/rational//05/329_kunal/

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

Nikolaidou M., Lelis D., Mouzakis D., Georgiadis P., 1999. "A Discipline Approach towards the Design of Distributed Systems", *Distributed System Engineering Journal*, Vol. 2, No 2, IOP.

Nikolaidou M. Anagnostopoulos D., 2003. "A Distributed System Simulation Modeling Approach", *Simulation Practice and Theory Journal*, Vol. 11, No 4, Elsevier Press.

Nikolaidou M., N. Alexopoulou, A. Tsadimas, A. Dais, D. Anagnostopoulos, 2006. "A Consistent Framework for Enterprise Information System Engineering", in proceedings of *IEEE EDOC 2006*.

OMG Inc, 2004a. UML Superstructure Specification, Version 2.0, 8/10/2004.

OMG Inc, 2004b. UML 2.0 Infrastructure Specification, 30/4/2004.

OMG Inc, 2005. MOF 2.0/XMI Mapping Specification, v2.1

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