Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

MODEL DRIVEN ARCHITECTURE – A TOOL FOR ENTERPRISE ARCHITECTURE: A LOOK AT EMERGENCY RESPONSE SYSTEMS

by
Ritu Lamba

Today's fast changing markets and technology drive software industry to deliver high quality enterprise software solutions. The enterprise system architecture should be designed to keep business functions separate from technological implementation to accommodate the fast changing business environment.

In this thesis we have studied the current best practices in defining enterprise architectures and Zachman's framework in particular. We have further examined the Model Driven Architecture (MDA) approach and its application to enterprise architecture definition. MDA separates the business logic from the underlying platform technology and defines a representation model based on precise semantics. Zachman's framework defines a set of views and category of models to describe complex objects as a combination of simple logical cells independent of each other. We mapped the two approaches into a model driven framework for enterprise architecture definition, which leverages the abstraction levels of MDA and the exhaustive views of Zachman's framework. We also examined the current work on designing Emergency Response Systems and customized our generic method to address their specifics. This thesis describes an ideal emergency response system, which we define as a virtual enterprise system, and articulates an Emergency Response System Design (ERSD) Framework that is a checklist of views for comprehensive system definition

MODEL DRIVEN ARCHITECTURE – A TOOL FOR ENTERPRISE ARCHITECTURE: A LOOK AT EMERGENCY RESPONSE SYSTEMS

by
Ritu Lamba

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Department of Computer Science

January 2006

MODEL DRIVEN ARCHITECTURE – A TOOL FOR ENTERPRISE ARCHITECTURE: A LOOK AT EMERGENCY RESPONSE SYSTEMS

Ritu Lamba

Dr. Vassilka Kirova **Date**

Research Professor, Department of Information Systems, NJIT

Dr. Fadi P. Deek **Date**

Dean, College of Science and Liberal Arts, Professor of Information Systems and Mathematical Sciences, NJIT

Dr. James A. McHugh **Date**

Professor, Department of Computer Science, NJIT

BIOGRAPHICAL SKETCH

Author: Ritu Lamba

Degree: Master of Science

Date: January 2006

Undergraduate and Graduate Education:

- Master of Science in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 2006

- Bachelor of Engineering in Electronics
  Tatyasaheb Kore Institute of Engineering and Technology, India, 2000

Major: Computer of Science

To my beloved family for their support and encouragement

ACKNOWLEDGMENT

I would like to express my deepest appreciation to Dr. Vassilka Kirova for being my thesis advisor. This work would not have been possible without her in dispensable efforts and involvement. I appreciate her insightful guidance and encouragement that inspire me at every step of my study.

I would also like to express my appreciation to Dr. Fadi P. Deek and Dr. James A. McHugh for serving as member of my thesis committee.

Special thanks go to my husband, Roopak Gupta for his understanding and encouragement.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF FIGURES

LIST OF TABLES

Table **Page**

CHAPTER 1

SOFTWARE DEVELOPMENT LANDSCAPE

1.1 Introduction

At the dawn of computing era Mainframe systems were considered the best for high end computing and heavy data handling. This large centralized computer managed data and automated business practices. Corporations benefited while the individual user had to be contented with computers systems that were slow and not much fun to work with.

The next era brought a paradigm shift in computer science and introduced us to the personal computer. The advent of Personal computers generated a new wave of handling the software application in a large enterprise world. The users became more productive and capable. The systems were totally disjointed and the data was shared with help of media drives. Still it was not much of use but a major break though in perseverance of computer science.

Then the personal computer started talking to each other though a network and the whole scenario changed drastically. It bought in the new words like LAN and by magic the new network of personal computers was ready to replicate the massive power of mainframe systems. Programming language such as Visual Basic and Turbo Pascal enabled the easy creation of custom application.

Then came the era of pervasive computing where all the computers of the world can be connected through information super highway known as "Internet". The primary difference in this era to its predecessor is that all the devices are widely linked, perhaps even globally, allowing them to be used in ways that part from their original purpose.

These devices are not only linked, but everywhere, the "system" is now the entire environment of the individual.

This new information medium has shifted business from the traditional brick-and-mortar infrastructures to a virtual world where they can serve customers not just the regular eight hours, but round-the-clock and around the world. The new business requirements have raised a need of software that provide enterprise wide support in a distributed environment with the capability to integrate all different piece of software as a single big enterprise system. Such new systems have to deal with a myriad of the complex interdependent dependability concerns like robustness, security, and reliability, ability, error recovery, service integrity which raised the complexity of these systems to a new high.

This higher complexity of the software systems has made the development of such software's as an error-prone and arduous task. Such software needs higher abstraction for smooth manageability. Software developers are in constant search of good abstractions that can help them to reduce the time and effort required to evolve and create new dependable and robust systems.

1.2 Raised Level of Abstraction in Software Development

The history of Software engineering is marked by ever raising level of abstraction. Journey of software development began with the binary coding, machine centric computing writing sequence of 1's and 0's. This laboriously writing out the bit pattern that corresponded to the native CPU later replaced by assembly language, a set of mnemonics designed for each hardware platform. The next phase of computing came

3GLs like FORTRAN and COBOL and now the effort involves more expressive traditional languages such as Java, C, C++ as well as domain-specific 4GLs.

The level of abstraction increased as we move from one language to another requiring software developer to learn a new higher-level language that may be mapped into lower-levels one from 3GLs to assembly code to machine code to the hardware. This increased level of abstraction improved software industry viability which is determine by the extent we can produce systems whose quality and longevity are in line with their cost of production.

Study on design abstractions has produced structuring techniques that are based on functional abstractions (e.g., Structured Design), data abstractions that encapsulate behavior and state of a conceptual entity (Object-Oriented development), and service-based abstractions in which data and functional elements pertaining to a set of provided services are encapsulated in units called components (Component-Based development). There are numerous quantitative context-specific evidences how effectively these abstractions can be used to manage complexity.

Today, we are on the cusp of a new era in computing, middleware grew out of sockets and other networking utilities thanks to Internet and rapidly changing Information technology and its infrastructure. The development environment, originally command-line tools, integrated compilers and linkers, is no longer just a matter of programming against an operating system, but rather writing against middleware, which is, in effect, a distributed, concurrent and secure operating system that works at a higher level of abstraction.

Application modeling using the Object Management Group's Unified Modeling Language, and application generation via Model Driven Architecture (MDA), are only the latest steps on this ladder of abstraction [1]. MDA models are even more abstract, in that they are farther away from the computer and closer to the business point of view. It still takes a person with technical knowledge to construct them, but a small model can correspond to hundreds or even thousands of lines of 3GL code. The models bear enough resemblance to the business information and processes that they automate to make it possible to train some business analysts to read them.

MDA is the latest in the series of moves to raise the level of abstraction at which we develop software. Unlike previous efforts to raise abstraction level, however, MDA seeks to continually push it higher. It is not satisfied with having narrowed the gap between the business and IT. It seeks opportunities to narrow it further. [3]

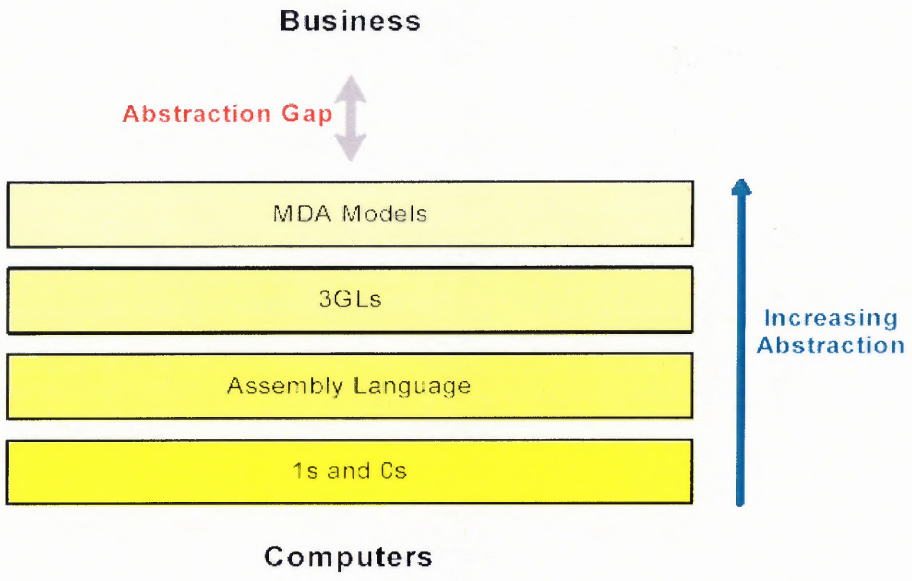


Figure 1.1 Abstraction Levels

1.3 What are Models?

Software modeling means visualizing the design of the system before getting started with the application development just like when building a house an architect produce the detail blue print. Models helps, by letting us work at a higher level of abstraction. Model is an abstraction of the system representing the essential characteristics of the system. A model may do this by hiding or masking details, bringing out the big picture, or by focusing on different aspects of the prototype. Models are more abstract in a sense that they are further away from the computer and closer to business point of view.

Modeling is not a new concept for the software industry. Model exists for a long time but software developers use them mostly as simple sketches of design ideas, often discarding them once they've written the code. There has been a strong tendency in the software industry to view formal design as superfluous to the production process, or as something that would be nice to do but that is unrealistic given the realities of short-term time pressures.

MDA tends to break down this resistance. It promotes the use of models that are not simply design artifacts but are actually production artifacts that drive code generators or are directly executed by virtual machines. Gradually, IT people stop looking upon MDA modeling as competing for time with production activities and start to see it as a productivity booster. [3]

It is useful to characterize the models in different categories depending upon the abstraction criteria that determine what information is included in these models. A model that is based on specific abstraction criteria is often referred to as a model from the viewpoint defines by those criteria or in short the view of the system.

Models have been classified into three different categories

- **Conceptual Models**: These models describe the real world situation such as a business process. These models represent domain modeling. This model can be viewed at a higher level of abstraction.
- **Specification Models:** These models describe what software system must do, what information it holds and what behavior system must exhibit. These models assume ideal computing platform.
- **Implementation Models:** These models describe how system must be implemented considering all the computing environment constraints and limitation. This model can be viewed at a lower level of abstraction.

Model simply describes the system from a different viewpoint that corresponds to different level of abstraction.

1.4 Why Modeling?

In the previous section we established the definition of model. In this section we'll discuss the benefits of developing software by help of models.

Jim Rumbaugh, one of the three leading designers of UML and a Rational Software methodologist, says: "The brave new e-world has turned previous assumptions on their head, and old approaches to business or software will no longer succeed. The e-world is now distributed, concurrent and connected. Concurrent, distributed systems have extremely complex interactions that can be hard to understand, let alone predict. Vague specifications are a major problem. In the past, the specifications for a monolithic system only affected the single system, and if it didn't work exactly as specified, nobody really

cared. But now a business system may have to interoperate with another system halfway around the world; people who have never heard of each other write both. A failure to follow specifications can introduce errors that propagate around the world." [4]

The world has indeed changed, placing new, more stringent demands on software development teams. Rumbaugh describes 'e-world' that is distributed, concurrent and connected, and he is right. It is distributed, because information vital to a company's business can be located all over the world. It is concurrent, because business processes are no longer centralized and rarely simultaneous. As Rumbaugh points out that "Neither business decision making nor software programs can live with a single thread of control." Finally it is connected, because an action in one place can produce effects anywhere else within the organization today. Put succinctly, the basic computer systems, languages, and models of the past are simply inadequate for today's needs. [4]

The ever evolving software system and their complexity requires that developers need a better understanding of what they are building, and modeling offers an effective way to do that. In this complex e-world, Modeling brings the business and technical people close. Modeling helps to ensure that they are speaking the same language. It provides architects and others with the ability to visualize entire systems, assess different options and communicate designs more clearly before taking on the risks -- technical, financial or otherwise -- of actual construction. If you build a house, the customer needs to speak to an architect and the architect needs to speak to the builder.

Understanding requirements before developing the system is very essential part of any software development. Modeling helps taking customer requirements and putting

them together so that all parties can understand. Thus models facilities the communication between technical and business people.

Companies models complex software applications because in the long run, a detailed blue print saves time and money. It allows developers to consider alternatives, select the best option, work out details, and achieve agreement before anyone starts building the application. Using a model, those responsible for a software development project's can assure themselves that business functionality is complete and correct, end-user needs are met, and program design supports requirements for scalability, robustness, security, extendibility, and other characteristics, before implementation of code. A good model documents the application's structure and simplifies modifications. This is critical when you consider that 90 % of the costs involved in large applications occur when they are changed, extended and otherwise maintained. [6]

By modeling software, developers can:

- Create and communicate software designs before committing additional resources
- Trace the design back to the requirements, to ensure that they are building the system as per the requirements.
- Practice iterative development, in which models and other higher levels of abstraction facilitate quick and frequent changes
- Decrease development cost
- Manage risk of mistakes

Modeling is a viable and efficient way to create high-quality, adaptable applications that more closely align to business objectives.

1.5 Introduction to MDA

Model Driven Architecture is an evolutionary step in the development of the software field. The Model Driven Architecture (MDA) developed by the OMG is a framework for software development using a system modeling language. The MDA aims to enhance portability by way of separating system architecture from platform architectures. MDA focuses on the evolution and integration of applications across heterogeneous middleware platforms. It provides a systematic framework using engineering methods and tools to understand, design, operate and evolve enterprise systems. It promotes modeling different aspects of software systems at levels of abstraction and exploiting interrelationships between these models. The most significance of MDA approaches exists in the independence of the systems specification from the implementation technology or platform. These specifications will lead the industry towards interoperable, reusable, portable software components and data models based on standard models. The three primary goals of MDA are portability, interoperability and reusability through architectural separation of concerns [12].

Model is the most important concept of the model driven architecture. By separating technology dependent concepts from independent concepts, MDA limits the problems of platform dependencies and increases portability of the software. This separation is supported at model level to avoid platform dependencies in al phases of the life cycle. The primary components of MDA technologies are the Platform Independent Model (PIM), and the Platform Specific Model. Platform Independent Models describe the structure and function of a system, but not the specific implementation. MDA can also be visualized as an approach to system development, which increases the power of

models. It is model-driven because it provides a means for using models to direct the course of understanding, design, construction, deployment, operation, maintenance and modification.

MDA has the capability to define templates that map transformations from Platform Independent Models to Platform Specific Models. This facilitates the development of a system in abstraction, and simplifies implementation of that system across a variety of platforms.

A key aspect of MDA is that it addresses the complete software development life cycle, including analysis and design, programming, deployment and management. UML, XML, MOF and CWM are the four main components that affect the interchange of information between tools and applications.

For instance, an MDA Transform from PIM to a DDL will create DDL table elements from a class, whereas the same class transformed to an EJB Entity Bean will result in a package containing the class and interface elements required by EJB. Enterprise Architecture helps to manage such transformations and even write your own transformation rules for any language. It will also aid you in keeping as many Platform Specific Models as you need synchronized to a single Platform Independent Model. Enterprise Architecture has built in support for MDA transforms to C#, DDL, EJB, Java and XSD.

In essence, the foundations of MDA consist of three complementary ideas:

- **Direct representation:** Shift the focus of software development away from the technology domain toward the ideas and concepts of the problem domain. Reducing the semantic distance between problem domain and representation

allows a more direct coupling of solutions to problems, leading to more accurate designs and increased productivity.

- **Automation:** Use computer-based tools to mechanize those facets of software development that do not depend on human ingenuity. One of the primary purposes of automation in MDA is to bridge the semantic gap between domain concepts and implementation technology by explicitly modeling both domain and technology choices in frameworks and then exploiting the knowledge built into a particular application framework.
- **Open standards:** Standards have been one of the most effective boosters of progress throughout the history of technology. Industry standards not only help eliminate gratuitous diversity but they also encourage an ecosystem of vendors producing tools for general purposes as well as all kinds of specialized niches, greatly increasing the attractiveness of the whole endeavor to users. Open source development ensures that standards are implemented consistently and encourages the adoption of standards by vendors.

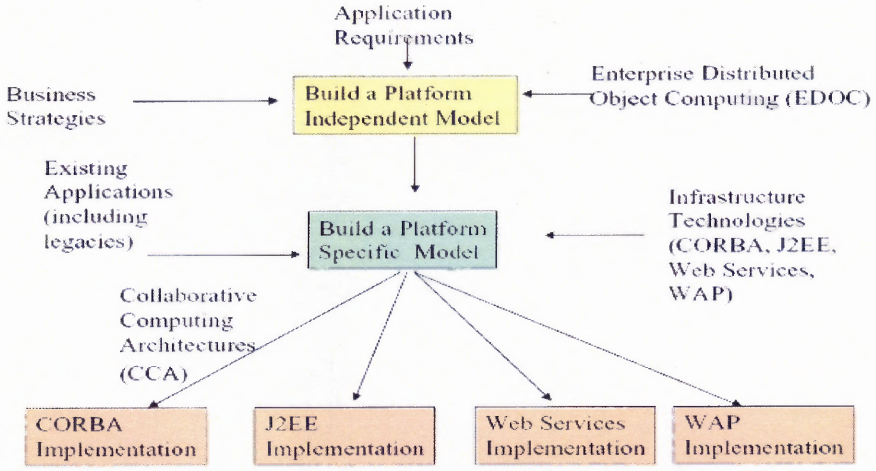


Figure 1.2 Overview of Model Drive Architecture

1.6 Models and Platform in MDA

A current trend in the development of distributed applications is to separate platform-independent and platform-specific aspects, by describing them in separate models. In MDA, the term platform is used to refer the technological and engineering details that are irrelevant to the fundamental functionality of the software components. Platform independence is a relative concept. It has meaning only with respect to some specified platform or platforms.

Platform-independence is a quality of a model that relates to the extent to which the model abstracts from the characteristics of particular technology platforms. The articulation of platform-independence is the most centric concept in MDA development.

MDA is all about transformation between models, each of which captures one or more subject matters and which are expressed in a language with a specific degree of abstraction. The MDA separates certain key models of the systems, and brings a consistent structure to these models. From MDA's point of view it is politically correct to think that there are two kinds of models

- **Platform Independent Model (PIM)** – This is the formal specification of the structure and function of the system that abstract away technical details. Business and modeling experts working together express the business functionality and rules undistorted by technology build these models. Because these models are independent of technology they retain their value over the years and require change only when business condition mandate.
- **Platform Specific Model (PSM)** – It provides a formal specification expressed in concepts of the specification models of the target platform. A platform

specific model is a view of a system from the platform specific viewpoint. A PSM combines the specifications in the PIM with the details that specify how those systems use a particular type of platform. A platform model provides a set of technical concepts, representing the different kinds of parts that make up a platform and the services provided by that platform. It also provides, for use in a platform specific model, concepts representing the different kinds of elements to be used in specifying the use of the platform by an application. [12]

Three important benefits of abstracting out the fundamental precise structure and behavior of a system in the PIM from implementation specific concerns in PMS are

- Simpler and more uniform models in PIM make it easier to validate the correctness of the model uncluttered by platform-specific semantics.
- In platform- independent terms integration and interoperability across systems can be defined more clearly then mapped down to platform specific mechanism.
- Defining business goals and policies in a computation independent manner make it easier to produce implementations on different platforms while conforming to the same essential and precise structure and behavior of the system.

OMG support different modeling standards for generating PIM and PSM models. The most commonly used standard is Unified Modeling Language (UML). Three keys OMG modeling technologies, based on UML, are MOF, CWM and XMI. In the next sections we will explore the above-mentioned key OMG modeling technologies.

1.7 Metamodels

Metamodels are the models of modeling language. They specify the concept of modeling languages that are used to create models. Metamodels simplifies the communication about models. We can view metamodels as the model whose instances are the types in other models or as mapping of meta-models elements to the modeling language's elements. This allows us to capture the other model and manipulate it. A well known meta-model is the specification for UML, which captures the classes in a developer's model. Metamodels may themselves be captured in meta-Metamodels. Metamodels facilitates the mapping and transformation between models.

1.8 UML – Unified Modeling Language

The Unified Modeling Language (UML) is a family of design notations that is rapidly becoming a de facto standard as software design language. OMG specification defines UML as "a graphical language for visualizing, specifying, constructing, and documenting the software intensive system. UML provides a variety of useful capabilities to the software designer, including multiple, interrelated design views, a semiformal semantics expressed as a UML metamodel, and an associated language for expressing formal logic constraints on design elements.

OMG's UML is based on common UML metamodel. Metamodel is in fact a class diagram and a set of semantics and syntactic rules that defines the core elements and relationship used in UML. In addition to core symbols, the metamodel contains supplementary symbols, called extensions.

UML Extensions are predefined set of Stereotypes, Tagged Values, Constraints, and notation icons that collectively extend and tailor the UML for a specific domain or process. An extensive package of stereotypes is referred to as a UML Profile. To specify the constraints on any diagram OMG has selected the Object Constraint Language (OCL)

1.8.1 Brief History

Back in late 80's there were different modeling methodology. Number of competing methodology appeared (Booch Rumbaugh, Shlaer-Mellor...). These approaches share many common features and also have arbitrary differences. The problem was that if different people were using different notations, somewhere along the line somebody had to do a translation. A lot of times, one symbol meant one thing in one notation, and something totally different in another notation. In 1991, everybody started coming out with books. Grady Booch came out with his first edition. Ivar Jacobson came out with his, and Jim Rumbaugh came out with his OMT methodology. Each book had its strengths as well as its weaknesses. OMT was really strong in analysis, but weaker in design. The Booch methodology was stronger in design and weaker in analysis. And Ivar Jacobson's Objectory was really good with user experience, which neither Booch nor OMT really took into consideration back then. [7]

In 1996 OMG announced it was interested in creating an open, standard object-oriented notation and called for proposals. Rational software submitted UML version 1.0 which had been developed by Booch, Rumbaugh, and Jacobson. Ultimately 21 other companies sent proposals. The OMG board approved the UML Version 1.1 specification resulted by blending the proposals of different companies like Hewlett Packard, IBM, Microsoft, Brest, France etc. and that covered most user and vendor needs. Since then,

OMG has managed UML as an open standard. An OMG task force gathers information about problems and improvements, and also schedule revisions. [6] OMG revision task force schedule minor changes frequently and major changes only at intervals that would enable developers and tool vendors to keep up with the changes and would also guarantee that the language evolved systematically.

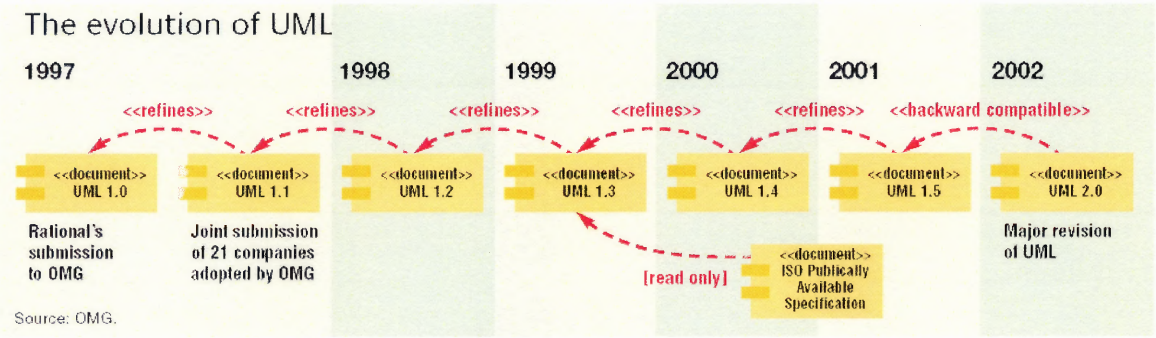


Figure 1.3 Evolution of UML

1.8.2 Goals of the UML

The primary goals in the design of the UML were as follows:

- Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
- Provide extensibility and specialization mechanisms to extend the core concepts.
- Be independent of particular programming languages and development processes.
- Provide a formal basis for understanding the modeling language.
- Encourage the growth of the OO tools market.
- Support higher-level development concepts such as collaborations, frameworks, patterns and components.

- Integrate best practices.

1.9 Meta Object Facility (MOF)

Meta Object Facility (MOF), OMG standard, is a model for specifying, constructing, managing, interchanging and integrating metadata in software system [8]. Interoperability of Metamodels across domains is required for integrating tools and applications across a development lifecycle using common semantics [9]. The main of aim of MOF is to provide a framework that support any kind of metadata and that allows the new kinds to be added as required. In order to achieve this goal MOF uses layered metadata architecture. The key feature of this architecture is the meta-meta-modeling layer that ties together the Metamodels and models. The four layers of metamodels architecture are

- M0 Layer – Information Layer defines the data of the application.
- M1 Layer – Model Layer contains the metadata that describes the data in information layer. This is the layer at which application modeling takes place.
- M2 Layer – Meta-Model Layer contains the meta-metadata that describes the structure and semantics of metadata. This is the layer at which CASE tools operate.
- M3 Layer – Meta-metamodel Layer comprised of description that defines the structure and semantics of meta-metadata.

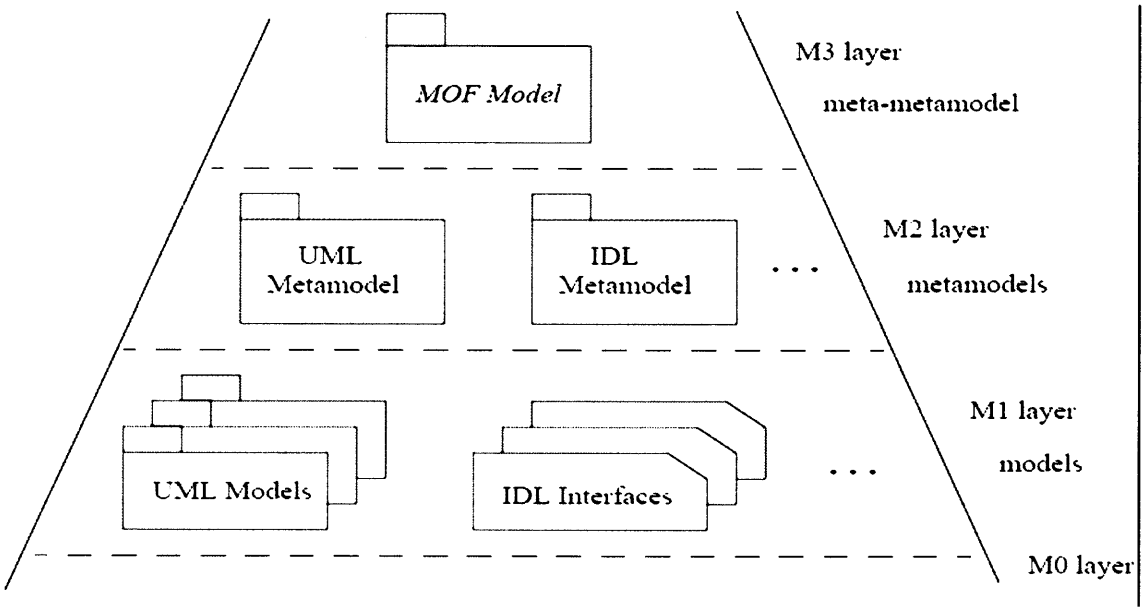


Figure 1.4 MOF Metadata Architecture [8]

The Key features of MOF are

- The MOF Model (the MOF's core meta-metamodels) is object-oriented, with meta-modeling constructs that are aligned with UML's object modeling constructs.[8]
- The MOF Model is self-describing. In other words, the MOF Model is formally defined using its own meta-modeling constructs.[8]
- The meta- levels in the MOF metadata architecture are not fixed. While there are typically 4 meta- levels, there could be more or less than this, depending on how MOF is deployed. [8]

1.10 XML Metadata Interchange (XMI)

XMI is a protocol that defines rules for deriving an XML Document Type Definition (DTD) from a MOF-compliant modeling language as well as rules for rendering a compliant model into a compliant XML document [9]. In short, XMI is mapping that expresses UML models into XML document. These XMI DTD rules that do the transformation are used like syntax for the construction of document. These rules are corresponds to metamodels of Layer M2 in MOF Metamodel architecture and the XML/XMI documents that are produce by the transformation corresponds to the Layer1 data. It is a standard interchange mechanism used between various tools, repositories and middleware [5]. XMI allows system developers to share models and metamodels over Internet on HTTP, IIOP or other wire protocols. XMI has the advantage of enabling exchange of models and metadata as files or as standard format based XML documents. UML visual modeling tools are currently adding XMI capabilities so that they can pass UML models from one tool to another [6].

1.11 Common Warehouse Metamodel (CWM)

CWM is the OMG defined standard language for data modeling, data warehousing, data transformation, and data analysis. It defines how the different data warehouse models relate to each other and enables exchange of data models, data transformation rule and data specification between tools from different vendors. CWM models are defined in terms of UML and their metamodels are defined in terms of MOF. Vendors like IBM, Oracle, Unisys Corp, Blue Bell, etc have already release their CWM complaint

warehouse. CWM database users can pass information between CWM complaint databases via XML because CWM is MOF complaint.

1.12 Summary

In this chapter we have explored the history and the current trends of software development. We established the benefits of standardization of the software development by using models. The contribution of Object Management Group is quite evident in establishing the modeling languages like UML that is generally used and accepted by software community. We also discussed the Model Driven Architecture and established its advantages. In the next chapter we will explore the software development process using Model Driven Architecture.

CHAPTER 2

MODEL DRIVEN ARCHITECTURE IN ACTION

2.1 Introduction

The software life cycle following the concepts of MDA consists of the following steps:

- Capturing requirements in a Computational Independent Model (CIM).
- Creation of Platform Independent Model (PIM) that represents the functional model of the system independent of specific technology.
- Mapping of PIM to one or more Platform Specific Models (PSM) by adding platform specific rules and code.
- Transforming the PSM to code
- Deploying the system in a specific environment

2.2 Capturing Requirements in CIM

Computational Independent Model (CIM) is used to model the requirements of a system. CIM captures the environment in which the system is actually going to working. This model is not concerned about the implementation details. It can be seen as a business model. It helps in setting the correct expectations of an enterprise system. It gives a common shared vocabulary to be used across the complete software development life cycle.

Capturing of requirements is the most fundamental aspect of the project design. The PSM model's foundation is laid on CIM. We do not need an expert in UML modeling. This can be documented with average skill programmer. The requirements

capture the general business process flow of the system along with the specific needs of the business.

2.3 Creation of Platform Independent Model

The next step is to create a model from the requirements. Platform independent model describes the system without showing its details of its use on a platform. This model presents the complete system without looking into its implementation details. It gives business functions a name and separates them as class in a model. We can use any kind of development environment like the ones that support complete MDA process or a visual modeling tool that does not support transformation but allows us to export the PIM model to a standard tool. An example from MDA guide illustrates the concept further:

A PIM is prepared using a platform independent modeling language. The architect chooses model elements of that language to build the PIM, according to the requirements of the application. These mappings may also specify mapping rules in terms of the instance values to be found in models expressed in the PIM language. Examples

- If the attribute "sharable" of class "entity" is true for a particular PIM model instance of type entity, then map to an EJB Entity, otherwise map to a Java Class. These kinds of rules may also map things according to patterns of type usages in the PIM.
- If pattern exists where an instance of class "entity" has a "manages" association to an instance of class "document", whose attribute "persistent" is set, then map the "entity" instance to an EJB Entity that manages whatever is mapped from the "document" instance identified by the pattern.

The system architects then chooses a platform for which the system will be modeled. The tools are used with the chosen platforms templates to generate the PIM models. We can capture PIM model by using UML, a graphical tool or also by OMG's XMI that is a text-based tool. Also they use different methods they capture the same model semantics.

2.4 Mapping of PIM to PSM

This intermediate layer is introduced by MDA to separate the decisions related to choice of deployment technology, programming language, protocols and operating system from code generation. Tools are used to distinguish and apply patterns to convert PIM to PSM. We need to give these tools a platform target and then they use the templates for the said target to develop a PSM.

One such tool, which can be used for conversion, is "OptimalJ". PSM can include database specific attributes and relationships and lists specifics about data types for each entity.

Mapping is the key issue in converting PIM to PSM. The choice of the platform decides which transformation maps should be used to convert platform independent model to the platform specific model. The architect of the system has documented many types of mapping whose choice depends on the type of platform chosen. Two examples, taken from MDA guide illustrate different approaches:

A platform model for EJB includes the Home and Remote Interface as well as Bean classes and Container Managed Persistence.

- Example: A UML PIM to EJB mapping provides marks to be used to guide the PIM to PSM transformation. It may also include templates or patterns for code generation and for configuration of a server. Marking a UML class with the Session mark results in the transformation of that class according to the mapping into a session bean and other supporting classes.

2.5 PSM to Code and Deployment:

The next step in the development life cycle of MDA project is to generate the code implementation from PSM and then the deployment of the generated code. The PSM to code generation is analogous to PIM to PSM generation. Again we can use tools to do the process. Here we may want to support different environments like development, test, staging and production. Each environment will have its own specifications and database connectivity's. Using the tool the team can then deploy the code in the application server.

2.6 Roles Defined in MDA Process

In a MDA process development lifecycle the roles of the people can be defined as under:

- Architects concentrate towards validating models and on create transformations to convert one model into another. They are also responsible for maintaining a health of the models. .
- Developers study the requirements and then crate Platform Independent models for the same. They then use these PIM's to crate PSM's by choose appropriate transformation.

- Programmers implement platform specific code for business rules that a PIM cannot express or a PIM generated by a transformation

2.7 Summary

In this chapter we discussed the development of software by using the approaches as laid down by MDA. We learned the system design guidelines as established by the MDA framework. MDA is slowly being widely accepted in the software development world. In the next chapter we will explore the industry support, which is being offered in favor of adoption of MDA as a standard framework.

CHAPTER 3

INDUSTRY SUPPORT AND FUTURE COURSE OF MDA

3.1 Industry Views about MDA

MDA gives the option of code reuse and this is a very attractive proposition for the companies. They will not be wasting their resources in re-engineering the code whenever there is a hard ware shift or there are changes to business. MDA gives the ability to work at the model level and generating code automatically, software teams will be able to keep the model in step with the debugging process. "You won't see the tendency to toss the model away in the middle stages of the project" said Cris Kobryn, co-chairman of the OMG's analysis and design task force and chief technologies at tools supplier Telelogic. The above advantages have eluded many companies to adopt the OMG standards and follow the software development guide lines as laid in MDA development.

Sam Greenblatt, Senior Vice President, Systems Strategy, Computer Associates says that "OMG's new Model Driven Architecture fits our needs, integrating with our software that manages e-business, and Computer Associates sees this as key to its infrastructure that will enable our clients over the next several years." Many "gurus" of the industry have raised similar opinions about MDA.

Software development productivity is the main essence of using new technologies to implement a solution. The determinants of productivity can be broadly classified as framework, tools and development methodologies. To this end MDA's emphasis on modeling provides acknowledged benefits, including long term flexibility to incorporate changes to a PIM, update or create new PSM's and deploy to multiple platforms without requiring substantial code rewrites. MDA also have its critics who are concerned about its

impact on the software development process as well as its reliance on UML for automatic code generation. Mr. Michael Jesse Chonoles the Chief of Methodology of Lockheed Martin Advanced Concepts Center says "By taking a modeling-centric approach, MDA gets us much closer to that ultimate goal of platform independent development and transparent reuse -- and it finally looks feasible and soon."

MDA supports the full life cycle by not only generating code from the initial model but also by allowing changes and regenerating code. The models and transformation languages used by MDA meet the open standards. Use of open standards allows the organizations to customize these languages to suit their environment.

One of the offspring benefits of MDA is that it allows the merging of work from related fields because it uses higher level of abstraction. In its current state MDA may not be the golden bird but there its definitely is a silver lining in the horizon of software development which can make the software development task much more streamlined and aliened as a manufacturing assembly.

Development of tools that enable automatic code generation from UML models in the next step of this paradigm shift of software development. The standardization teams will have to tighten up the semantics of UML to achieve consistent code across different tool suppliers. UML 2 was a major project of OMG in this direction. The development of tools that support MDA is also essential to generate a repository of models that will further reduce the development time of the new systems as they can leverage from the previously developed system.

3.2 Tools Supporting MDA Modeling

Over the past year or so, a number of tool vendors and service providers have extended their support to MDA. There are at least 40 tools that incorporate at least one of the major aspects of MDA: UML-based modeling; transformation between the app's overall design models and the models that are specific to the underlying computing architecture (.NET, EJB and so on); and the generation of code in a specific language.

Iona, InferData, Codagen Technologies, Eltegra, Hewlett-Packard and IBM are just a few of the companies that are either developing MDA tools or adopting and promoting the use of MDA. While evaluating or selecting MDA complaint tools, it has been found that tools are still not matured enough to support the complete MDA process. In most cases, it would be necessary to modify the generated source code or to write the code manually. Fortunately as more companies are supporting MDA, specifications will evolve and the tools will mature.

Some commonly used tools and their features are described below

3.2.1 IBM Rational Rose

IBM Rational® Software Architect is an integrated design and development tool that leverages model-driven development with the UML for creating well-architected applications and services. With Rational Software Architect, unify all aspects of software design and development.

Main Features:

- UML 2.0 modelling support for analysis and design using Use Case, Class, Sequence, Activity, Composite Structure, State Machine, Communication, Component, and Deployment diagrams.

- Support for the visual modelling with content-assist.
- Apply and author patterns and transforms.
- UML Class diagram editing for Java, Enterprise Java Beans, and Database objects.
- Support for the UML Sequence diagram editing for Java.
- Java method body visualization using UML 2.0 Sequence diagrams.
- UML Class diagram editing for C++.
- Uses transformations to generate Java, C++, or EJB code.
- Asset Browser for accessing reusable assets.
- Establish Traceability links from requirements through implementation.
- Automatically detect patterns and anti-patterns (ex: design, OO, structural, and system) in Java code.
- Template based rules for monitoring and enforcing application structure.
- Enterprise class IDE powered by Eclipse technology.
- WS-I compliant Web services and service oriented architectures.
- Rapid application development tools and wizards.
- Drag-and-drop UI components, point-and-click database connectivity.
- Automated tools for coding standards enforcement; component testing of Java, EJB, Web services; and multi-tier runtime analysis.
- Built-in Crystal Reports tools.
- C/C++ development environment with syntax highlighting editor and customisable build and debugger framework.

- Requirements perspective for browsing requirements in Requisite Pro and creating links to model elements.
- RUP configuration for Software Architects with context-sensitive and dynamic process guidance.
- Open API to support customizing and extending the modelling environment. UML profile creation and editing to customize the properties stored in UML models.
- Generate HTML, PDF, and XML reports from UML designs.
- Generate Javadoc with detailed design diagrams.
- Scripting support with Java.
- Team support with multi-model support, compare merge, and SCM integrations.

3.2.2 IBM Rose RT

Rational Rose Real Time is a comprehensive visual development environment that delivers a powerful combination of notation, processes, and tools to meet these real-time challenges. Through the industry standard UML, real-time design constructs, code generation, and model execution, Rational Rose Real Time addresses the complete lifecycle of a project; from early use case analysis, through to design, implementation, and testing.

Main features of Rose RT are:

- UNIFY your teams by describing your real-time embedded systems using the Unified Modeling Language, the industry standard notation championed by Rational Software

- Optimize your software development by generating complete, high-performance executables directly from UML design models -targeted to real-time operating systems
- Simplify tool-chain complexity by providing seamless integration to leading real-time operating systems, compilers, symbolic debuggers, and other market-leading Rational Software products.
- Executable models let you compile and observe simulations of your UML designs
- Model execution encourages early design refinement and continuous validation.
- Complete, deployable executables can be generated directly from UML design models - targeted to real-time operating systems.
- Automated generation of complete C++ applications eliminates the need for manual translation and avoids costly design interpretation errors.
- Improve communication between all members of your team through the power of the UML.
- Capture your architecture more effectively and make it part of the implementation.
- Software Configuration Management end Version Control tool integration allows you to use products like Rational Clear Case to even more effectively manage your UML application development.

3.2.3 I-Logix Rhapsody

Rhapsody is the industry's leading Model-Driven Development environment based on UML 2.0 and SysML for systems, software, and test, and has the unique ability to extend

its modeling environment to allow both functional and object oriented design methodologies in one environment.

Model-Driven Development (MDD) technology enables you to achieve unparalleled gains in productivity over traditional document driven approaches by enabling you to specify your systems and software design graphically, simulate and automatically validate the system as you build it, and ultimately produce full production code from the model for the embedded system.

- Seamless Environment for Systems and Software Development
- Advanced Graphics Engine to allow Domain Specific Modelling
- White Boarding (free sketch)
- Custom Bitmaps
- Advanced Layout and Ergonomics
- Profile Formatting "skins"
- Requirements Modelling and Traceability
- Full Behavioural Model Simulation
- Model Driven Test Generation
- Requirements Based Testing
- Automatic and Customizable document generation
- Model Execution on Embedded Target
- Directly Deployable C, C++, Java, and Ada Code Generation
- Code Visualization and Reverse Engineering

3.3 Summary

This chapter emphasizes the importance of MDA in current software industry by showing the industry support for MDA. We have also explained the various tools available for the designer to develop the software systems using MDA. A brief description of the common tools was also characterised. All the above discussion in the previous chapters is to design a software system. In next chapter we will discuss the Enterprise Architecture in the light of MDA.

CHAPTER 4

ENTERPRISE ARCHITECTURE AND MDA

4.1 Overview

Enterprise is defined or viewed as a complex system with a defined boundary and consists of differentiated and interdependent components. It is surrounded by an external environment which influences the enterprise operations and provides the various inputs that are transformed by the enterprise components to produce the output in the form of products and services that are returned to the external environment. [35]

Enterprise Architecture is a framework or "blueprint" which describes the linkage between the components of an enterprise and defines how an enterprise achieves the current and future business objectives. It analyzes the key business, information, application, and technology strategies and their impact on business functions. Each of these strategies is a separate architectural discipline and Enterprise Architecture is the glue that integrates each of these disciplines into a cohesive framework as shown in Fig. 4.1.

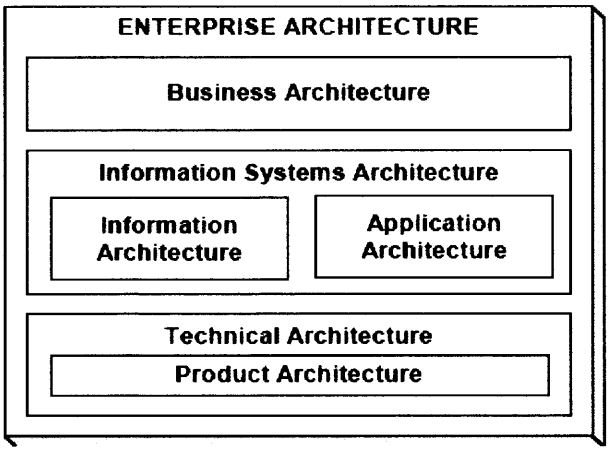


Figure 4.1 Enterprise Architectural Relationships

Enterprise Architecture is a logical link between enterprise business, information, and technical architectures and thus acts as a planning, structuring and integrating guideline for creating and maintaining the enterprise-wide information systems.

The EA is a top-down, business strategic driven process in a sense that analysis begins by looking out the window at the new market, competitive and other environmental forces that affect the organization. Creation of EA is an iterative process, which involves refinement of various artifacts that represents the holistic view of the enterprise's key business, information, application, and technology strategies and identifies the gap between the current and future state of an enterprise

EA provides the basis for organizing the information management resources and enterprise-wide review and oversight mechanism for different policies and projects. The hierarchal linkage from business strategy level to IT implementation level enable organizations to align business goals and IT investment plans and facilitates communication and decision making between Business strategy and IT investment groups. EA has been recognized as an approach that drives both business and technology strategies.

In 1987 John A. Zachman, an IBM researcher, proposed what is now popularly called the Zachman Framework, a way of conceptualizing what is involved in any information system architecture. [36] It is an analytic model that organizes descriptive representations without describing an implementation process and is independent of specific methodologies.

The Object Management Group's Model Driven Architecture (MDA), as described in earlier chapters, is an approach to create models, and generate code from

models. The MDA approach also includes technology to facilitate the transport of models and code from one implementation to another, and the ability to reverse engineering code into models. It is a generic approach that can be used with any existing methodologies including Zachman Framework.

The objective of this chapter is to establish the importance of enterprise architecture and its design principles, Zachman Framework and it's mapping with MDA.

4.2 Importance of Enterprise Architecture

Maturity of Industrial age and globalization of industries brought a paradigm shift in data access. Information is no longer being accessed by just few top executives of the enterprise but is accessed globally through out the enterprise. Data needs to be captured once and reused throughout the enterprise in different processes and applications. The concept of the local market is defunct. Products and services have to be produced and integrated to fulfill customer requirements. This results in large and complex distributed enterprise systems enabling business to engage in global market place. Enterprise requires extensive automation to compete in this global information age. Systems are no longer discretionary support for the enterprise they are mandatory.

The customization of information and services provided to the customer is becoming the key of the success of the business world. This brings us the requirement of changing or modifying the existing systems following the fastest route and having the minimum impact on different components in the chain of the enterprise systems. To accomplish this we'll need enterprise architecture that describes the enterprise, as it exists at a given point in time and helps to focus the efforts in the area that require immediate

attention or a change. The impacts resulting from any change in one area are more readily discerned when a blueprint of a business is available for the analysis.

EA provides the readily available documentation of the enterprise. These documents helps the shareholders to see the enterprise not only as it is but as it is envisioned to be, assuming its owners wish to bring about changes and need to communicate a common understanding of them to stakeholders who are not only affected by the changes but are also expected to participate in bringing them out. [15]

For large organizations, it is impossible for people to retain and work with so many variables to bring about meaningful change unless information about them is documented through EA. [15] In the past, the development of IT systems was more ad-hoc. There was no common language among the designers of the enterprise in the meeting rooms. This made the development of different systems disparate and non inter-portable. The development using the framework of EA, leverage the idea of templates and avoid difficulties that used to arrive due to absence of reference model for processes data and technology .The concept of reference models gave a pivotal point that can be used by the designers of the system.

From our previous discussion, we can deduce the following advantages of Enterprise Architecture

- EA harmonizes the linking of strategic and business planning to business architecture, from BA to IT architecture, and from IT architecture to IT implementation. EA forces this because EA requires all this knowledge to be made explicitly visible and to be used as a basis for approving IT investments. [15]

- It defines the mechanism to seamlessly integrate all the artifacts of an enterprise to achieve an effective information flow.
- It enables an integrated vision and a global perspective of informational resources. [8]
- It enables the discovery and elimination of redundancy in the business processes reducing information systems complexity. [15]
- It contributes to having information systems that reflect common goals and performance measures for all managers, to encourage cooperation rather than conflict, and competition within organizations. [38]
- It becomes the bridge between the business and technical domains. [13]

4.3 EA Design Principles

The architectural design of the enterprise wide information system is the integral element in the success and prosperity of the company. A sound and well thought design of the system will give a vision to the designers to develop the system in conjunction with the business requirements. The root of information system lies in building a sound Enterprise information technology architecture (EITA).

Architectural design is usually a complex process as it has to consider a vast number of parameters and it lays foundation of the design of multi-vendor and heterogeneous systems. The two critical and important tasks that the designers of EA have to accomplish prior to building enterprise architecture are

- To identify the process which will be best suited for the enterprise. The process must be flexible to accommodate and accept a wide range of architectures and functional areas. It must also be able to handle multiple iterations for refinement.
- To identify an architectural framework on which enterprise architecture will be build upon. Designers can build the framework either from the s either from scratch or they can use the existing framework. Designing from scratch takes time, effort, energy and money. A good alternative is to leverage upon existing frameworks and customizing them to achieve the vision of the company.

After identifying the process and the framework for an enterprise we can follow the following steps iteratively to develop the complete enterprise architecture.

- **Initiate the Process**: We start the process by defining the scope of the project and getting together the architecture team. This step will also initiate the process of identifying and influencing the stakeholders, encouraging participations and involvement in discussions about the project. This also helps in overcoming resistance to change and creating readiness for architecture.
- **Characterize the baseline architecture:** The next steps involve establishing the baselines by describing the current platforms. Expectations are established. All the stakeholders are also briefed about the reasons for the changes and what will be its valued outcome after the development is complete. The infrastructure view, functional views and informational views are base lined by conducting user surveys.
- **Develop the Target Architecture**: We will establish the target architecture. It helps creating the vision of the system and also generates enthusiasm among the

stakeholders. Many models are discussed to generate a feasible future model of the enterprise system. The target architecture views base lined in step 2 are defined. Iteration between step 2 and 3 refine the gaps in the baseline architecture views.

- **Plan Architectural Transitions**: Next step is to develop the transition plan and execute the target architecture. In this phase a sound management structure is put into place and support for the architect is established.
- **Plan Architecture Implementation**: This phase maps the resources budgets schedule, people and products to the choices made in previous steps. The program management plan is defined and updated as per the availability of resources.

4.4 The Zachman Framework

One of the objectives of the enterprise architects is to identify a framework as described in the previous section. Zachman framework is the common choice among enterprise architects. In this section we'll explore Zachman more into detail.

Zachman defines "Architecture as the set of design artifacts or descriptive representations that are relevant for describing an object such that it can be produced to requirements as well as maintained over the period of its useful life." He suggested that an organization does not have a single architecture, but has, instead, a whole range of diagrams and documents representing different aspects or viewpoints and different stages. Figure 4.2 provides an overview of current Zachman Framework. [36]

Abstractions (Columns) →

Perspectives (Rows) ↓

The Zachman Framework	DATA *What (Things)*	FUNCTION *How (Process)*	NETWORK *Where (Location)*	PEOPLE *Who (People)*	TIME *When (Time)*	MOTIVATION *Why (Motivation)*
SCOPE (Contextual) *Planner*	List of things important to the business *Entity = Class of business thing*	List of processes the business performs *Function = Class of business process*	List of Locations in which the business operates *Note = Major business location*	List of Organizations Important to the Business *People = Major organizations*	List of Events Significant to the Business *Time = Major business event*	List of Business Goals/Strategies *Ends/Means = Major bus. goal/Critical success factor*
BUSINESS MODEL (Conceptual) *Owner*	Semantic Model *Ent = Business entity* *Rein = Business relationship*	Business Process Model *Proc = Business process* *I/O = Business resources*	Business Logistics System *Node = Business location* *Link = Business linkage*	Work Flow Model *People = Organization unit* *Work = Work product*	Master Schedule *Time = Business event* *Cycle = Business cycle*	Business Plan *End = Business objective* *Means = Business strategy*
SYSTEM MODEL (Logical) *Designer*	Logical Data Model *Ent = Data entity* *Rein = Data relationship*	Application Architecture *Proc = Application function* *I/O = User views*	Distributed System Architecture *Node = I/S function (Processor, Storage, etc.)* *Link = Line characteristics*	Human Interface Architecture *People = Role* *Work = Deliverable*	Processing Structure *Time = System event* *Cycle = Processing cycle*	Business Rule Model *End = Structural assertion* *Means = Action assertion*
TECHNOLOGY MODEL (Physical) *Builder*	Physical Data Model *Ent = Segment/Table, etc.* *Rein = Pointer/Key*	System Design *Proc = Computer function* *I/O = Data elements/ sets*	Technology Architecture *Node = Hardware/ System software* *Link = Line specifications*	Presentation Architecture *People = User* *Work = Screen format*	Control Structure *Time = Execute* *Cycle = Component cycle*	Rule Design *End = Condition* *Means = Action*
DETAILED REPRESENTATIONS (Out-of-Context) *Sub-Contractor*	Data Definition *Ent = Filed* *Rein = Address*	Program *Proc = Language statement* *I/O = Control block*	Network Architecture *Node = Addresses* *Link = Protocols*	Security Architecture *People = Identity* *Work = Job*	Timing Definition *Time = Interrupt* *Cycle = Machine cycle*	Rule Specification *End = Sub-condition* *Means = Step*
FUNCTIONING ENTERPRISE	Actual Business Data	Actual Application Code	Actual Physical Networks	Actual Business Organization	Acutal Business Schedule	Actual Business Strategy

Figure 4.2 Zachman Framework

Zachman states, "The Framework for Enterprise Architecture is a two dimensional classification scheme for descriptive representations of an Enterprise."[2] The rows of the framework are known as the "Perspectives" of the model and defines the views of the IS participants who uses the models contained in the cells. The top row of the framework represents the most generic perspective of the organization while the lower rows are more concrete. The Six perspective as show in figure 1.2 are described as under

Scope: (Contextual) The Planner's Perspective: This row defines the models, architecture and representation that define the scope and boundaries of the enterprise system. It provides the high level contextual view of an organization and its interaction to the outside world as perceived by the senior executives.

Business Model: (Conceptual) The Owner's Perspective: This row is used to identify the facilities, objects and the assets of the system. The business process owners describe the models in this row like the semantic model, business process model, logistics system or the workflow models. These models help to allocate responsibilities and focus on the characteristics of the enterprise system.

System Model: (Logical) The Designer's Perspective: This describes the models and architectures used by technical architects and engineers. They decide the feasibility and the desirability of the required system. The owner perspective addresses the desirability of the system and the designer perspective lays down the practical possibility of achieving it technologically.

Technology Model: (Physical) The Builder's Perspective: This describes the models, architectures and descriptions used by technicians, engineers and contractors

who design and create the actual product. The emphasis is given to the identification of the constraints and the actual construction of the system.

Detailed Representations (Out of Context Perspective): This describes the actual elements or parts that are included in, or make up, the final product. Using the construction metaphor, Zachman refers to it as sub-contractor's perspective, and this makes sense to software developers when the design is implemented with modules of components acquitted from others.

Functioning Enterprise: This is the functional model of the enterprise in the real world. The cell of this row defines the actual components, which can be combined to make the enterprise system work.

The horizontal dimensions or the columns of the framework define the types of abstraction for each perspective. They are built upon the frequently asked questions while designing an enterprise application. The horizontal columns are data, function, network, people, time and motivation.

Data: - The perspectives of this column concentrate on the material composition of the enterprise system. Zachman goes on to elaborate model for each. For the Data column he follows the model Thing-Relationship-Thing.

Function: (How) - The main focus is on the functionality of the system and how to achieve the required. The Zachman model is: Process- Input/Output – Process.

Network: (Where) - This focuses on the physical locations of the entities of the system. It also defines the communication about different modules of the system. The Zachman model is: Node-Line-Node

People: (Who) - This defines the distribution of tasks among the people. It lays down the responsibilities of the people involved in the enterprise system. The Zachman model is: People-Work-People

Time: (When) - The focus in this column is on the event sequence. It defines the schedule for all events. The Zachman model is Even- Cycle-Event.

Motivation: (Why) - This defines list of goals and strategies for each perspective. It says why a particular item is being done and what can be achieved by it. Zachman model is End-Means-End.

We can characterize the Zachman Framework as:

- **Simple** - It's easy to understand non-technical and pure logical.
- **Comprehensive** - It addresses the enterprise in its eternity.
- **A language** - It helps to think about complex concepts and communicate them precisely with few, non-technical words.
- **A planning tool** - it helps to make better choices, as you are never making a choice in vacuum.
- **A problem solving tool** - It helps you to work with abstractions and define simple modules without losing the vision of the complex enterprise architecture.
- **Natural** - It is not defined based on any tool or methodology.

The framework for Enterprise Architecture is not the answer. It is a tool for thinking. If it is employed with understanding, it should be great benefit to technical and non-technical management alike in dealing with the complexities and dynamics of the Information Age Enterprise. [37]

4.5 Mapping of MDA and Zachman Framework

In this section we'll explore how MDA can be used to capture and use the information defined by Zachman Framework in the section 1.4. Zachman Framework describes all the architectures, models and representation that managers and developers need to keep track of and the MDA approach is designed to support the creation and management of enterprise architecture. [38]

The goal of MDA is to create an enterprise architecture modeling capability that analyst and developers can use to describe a company's business and software assets. [38] As discussed in earlier chapters MDA describes how an IT group can derive models from software descriptions and business processes. MDA models are classified as Computational Independent model (CIM), Platform Independent Model (PIM) and Platform Specific Model (PSM). (Refer Chapter 1 & 2 for more details). Computational Independent Model (CIM) describes and captures the business requirements of the enterprise. This model reflects the business owner perspective and maps to Business Model Row of the Zachman Framework. Similarly, Platform Independent Model represents the perspective of technical designers and map to System Model row and Platform Specific Model describes the view of system builders as per Zachman Framework These models mappings to the rows of the Zachman Framework is shown in fig 4.3.

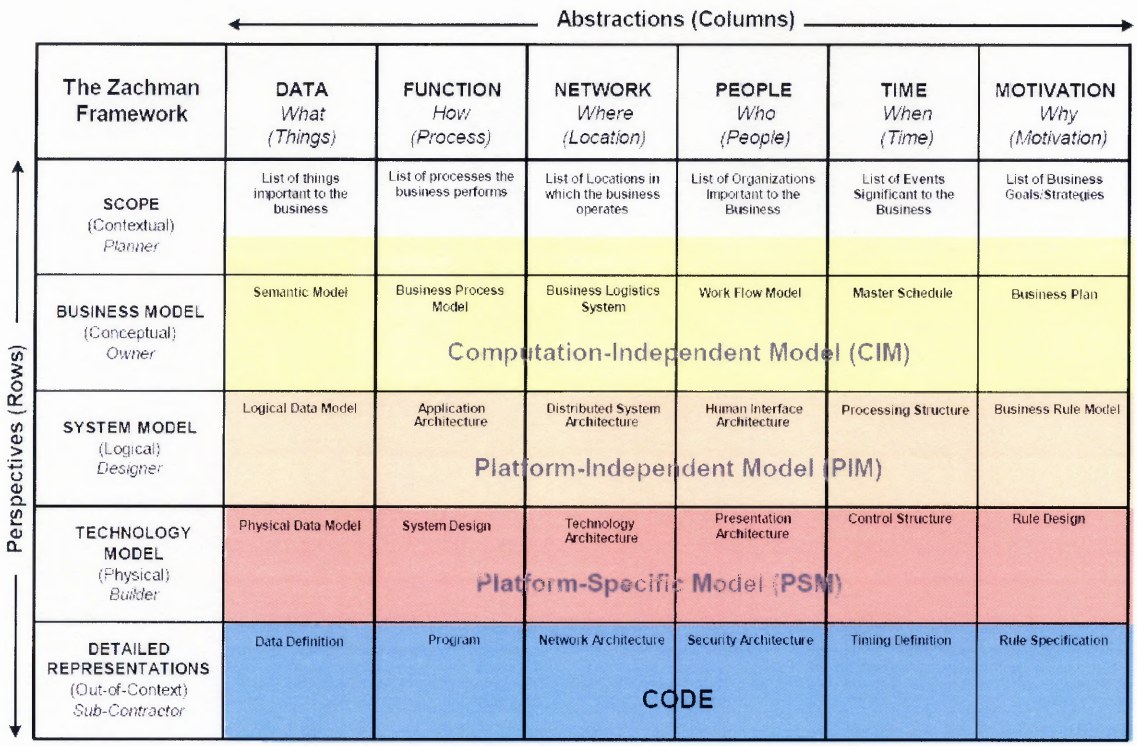
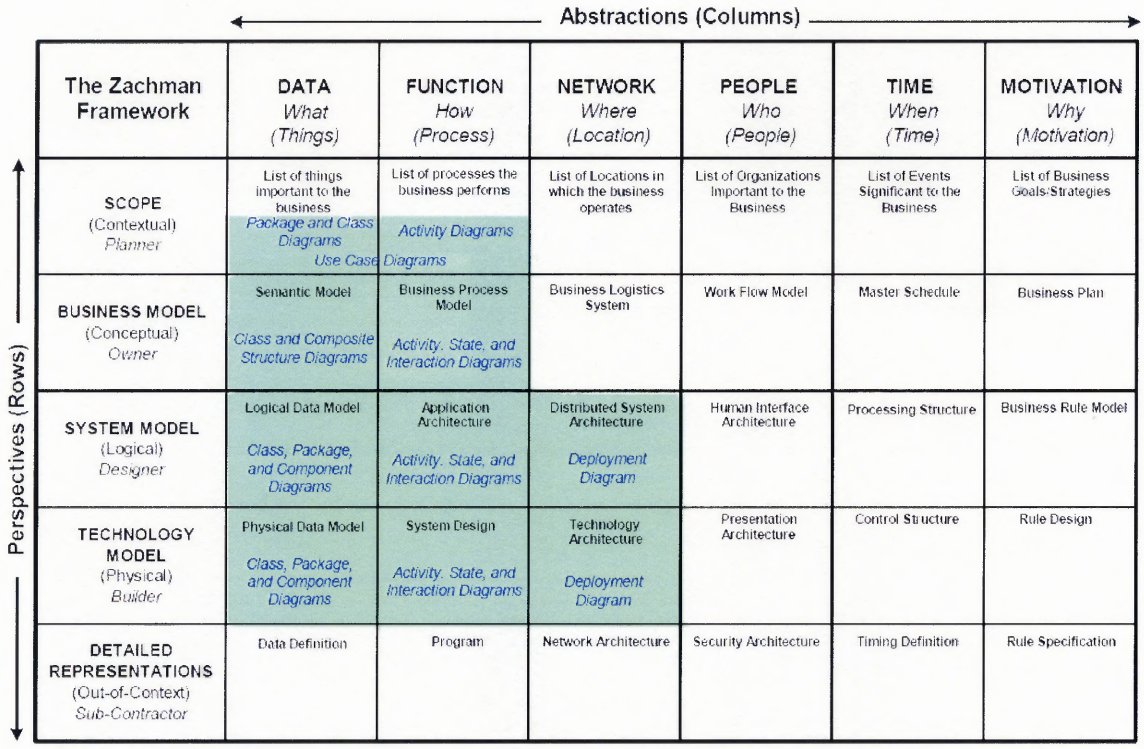


Figure 4.3 Mapping of MDA Models to Zachman Framework

Model Driven Architecture is supported by number of UML models and standards .All these models are derived from a very abstract Metamodel know as MetaObjectFacility (MOF). These UML models map to Zachman Framework as show in Fig 1.4. An architect or an analyst using UML would probably capture information from different Zachman cells using simpler UML diagrams and then add details to turn the initial diagram into a more complex diagram to achieve the specific purpose. For example a single activity diagram can incorporate elements that are described within different cells on the Zachman Framework or a very general class diagram depicts the important concepts to the business and entity relationship diagram represent entity important to the business.



The Zachman Framework	DATA *What (Things)*	FUNCTION *How (Process)*	NETWORK *Where (Location)*	PEOPLE *Who (People)*	TIME *When (Time)*	MOTIVATION *Why (Motivation)*
SCOPE (Contextual) *Planner*	List of things important to the business *Package and Class Diagrams* *Use Case Diagrams*	List of processes the business performs *Activity Diagrams*	List of Locations in which the business operates	List of Organizations Important to the Business	List of Events Significant to the Business	List of Business Goals/Strategies
BUSINESS MODEL (Conceptual) *Owner*	Semantic Model *Class and Composite Structure Diagrams*	Business Process Model *Activity, State, and Interaction Diagrams*	Business Logistics System	Work Flow Model	Master Schedule	Business Plan
SYSTEM MODEL (Logical) *Designer*	Logical Data Model *Class, Package, and Component Diagrams*	Application Architecture *Activity, State, and Interaction Diagrams*	Distributed System Architecture *Deployment Diagram*	Human Interface Architecture	Processing Structure	Business Rule Model
TECHNOLOGY MODEL (Physical) *Builder*	Physical Data Model *Class, Package, and Component Diagrams*	System Design *Activity, State, and Interaction Diagrams*	Technology Architecture *Deployment Diagram*	Presentation Architecture	Control Structure	Rule Design
DETAILED REPRESENTATIONS (Out-of-Context) *Sub-Contractor*	Data Definition	Program	Network Architecture	Security Architecture	Timing Definition	Rule Specification

Figure 4.4 Mapping of UML Diagrams to Zachman Framework

Zachman framework can be implemented in many different ways but implementing it by MDA approach offers the breath and consistency that allow managers, architects and developers to use the resulting framework in a consistent manner over the course of year.

CHAPTER 5

EMERGENCY RESPONSE SYSTEM

5.1 Overview

In the previous chapters we learned about development of information systems using modeling techniques and developed an understanding of Zachman Framework. The modeling techniques described in the previous chapters are very effective in developing an elaborate and complex system like Emergency Response System. In this chapter we will discuss Emergency Response System and develop the major requirements and design principle of such system.

Emergency response system is the integral part of human response to any crisis situation. The response of the team will have far reaching implications on every aspect of human civilization, depending on the severity of the crisis. A good information system can dramatically improve the degree of preparedness for, respond to and recover from disaster across geographical domains.

The recent emergency crises like hurricane Katrina, terrorist attack of 9/11 have brought great sorrows and taken their toll on the economy and life. These crises have highlighted the major flaws in our emergency response. There were three key issues in the response to disasters that are recurring in jurisdictions across the nations. Firstly, there was too much reliance on voice-oriented communications. Secondly it was observed that there is limited awareness of the situation among the emergency personals that hindered in their judgment and decisions. Thirdly there was lack of interoperability. There was no defined process for interdepartmental communications either. In addition, technological limitations impeded interoperability, situational awareness, and rapid

coordination of activities between emergency response departments resulting in confusion, inefficiency, and costly response time and reduced quality of care. Emergency response communications require an alternative means of communication and information sharing flexible enough to support the dynamic nature of emergency response. [41]

The inefficiency of the emergency response system, which came to light by the recent past, has put forth the need for modeling and developing an emergency response system that will rectify the current deficiency of the Emergency Response System. The Committee on Science and technology for Countering Terrorism of the National Research Council identified "systems analysis, modeling and simulation" as the first of the seven crosscutting challenges to be addressed to counter the terrorism threat. The report states: Systems analysis and modeling tools are required for threat assessment; identification of infrastructure vulnerabilities and interdependencies; and planning and decision making particularly for threat detection, identification and response coordination. Modeling and simulation also have great value for training first responders and supporting research on preparing for, and responding to, biological, chemical and other terrorist attacks. (National Research Council (2002))

The emergency response system should target two types of audiences. First, who participate in decision-making, second, the people who are first responders. The focus will be on systems and data that dramatically improve coordination, planning, situation awareness, and decision making of geographically dispersed multi-organization response teams before, during, and after a disaster.

An ideal emergency response system can also be used for training and exercising members of the emergency operations unit, simulating the most vivid and lively situations happening in real world. They are also increasingly used to simulate crises requiring humanitarian assistance, natural disasters or man make distracters. Emergency response training tool should be able to train emergency responders for dispersion of radiological, chemical and bio chemical agents. It also should be able to effectively handle a terrorist attack, fire outage, storm breakout etc. Most important to the success emergency response system is the capability to share information and results across simulation tools and across different agencies in real life scenarios. This can be achieved by leveraging the technological tools offered by the computer science in today's era. Such system should improve the procedures used in a crisis situation by leveraging the past information and using it as the intelligence to the current emergency crisis.

5.2 Types of Emergency Events

Emergency response agencies have to respond to a number of man-made and natural disaster events. Some of areas of man-made system where emergency may arise either due to natural calamities or human errors can be classified as follows:

- Nuclear and radiological plants.
- Human and agricultural health systems.
- Toxic chemicals and explosive materials.
- Information technology, telecommunications.
- Energy systems.
- Transportation systems.

- Cities and fixed infrastructure.
- Response of people to terrorism.

The natural emergency situation can be listed as:

- Earthquakes
- Storms
- Fire
- Building collapse

The nature and extend of the catastrophe will decide the extent of the response of the team. It can be a small isolated incident or it can be a widespread phenomenon engulfing a wide geographical area and affecting many people.

Most of the emergency situation will not need the widespread communication between different agencies. This ideal emergency response system should be capable of handling very small emergencies, involving only one or two agencies, to big catastrophe, which may involve multiple agencies.

5.3 Requirements of an Ideal Emergency Response System

The characteristics of the ideal emergency response system can be classified as the following:

5.3.1 System Response

An emergency response system should be able to quickly adapt it self to the requirements of the situation and tailor its objects around people, actions, relevant information and discussion. It should also learn and use the knowledge of similar incidents in the past to leverage the current situation with past intelligence.

Emergency response system requires development of simulated environments, which integrates the potential human subject issues, and has a well defined objectives and metrics involving cross functional teams participating in the possible scenarios.

5.3.2 System Integration

The integration of the distributed environment of the all the agencies is the core of this emergency response system. The interaction and acknowledgment of the current development to the authorities during crisis is of one of the most important consideration that accounts for reduction in of the casualties during a catastrophe.

We will leverage the technology to support a reliable network between various servers of different agencies and personal computers, mobile devices and laptops of the field agents to coordinate the actions. The integration will also allow reports to flow through the system and all agencies are apprised of the current real field situation. The system should support communication with multiple media centers like radio, TV, phone, Internet, e-mails, pagers etc.

5.3.3 System Training

The emergency systems are not like regular systems that are used on a day-to-day basis. This makes even more important to train the emergency personal to assist transfer of learning from classroom to job. This is different from normal training, as the training has to be practiced in a crisis situation where many factors are against normal scenarios.

The system should be easy to learn and it should be complex enough to create possible situations of a given scenario. This will allow the people to understand their

roles and responsibilities in an emergency situation. It should have the ease of use that can be utilized by non-technical staff handling the crisis situation.

5.3.4 System Intelligence and Planning

A fast and easy search of the database should be available to extract the relevant intelligence by using the past date of the similar occurrence of the situation. It should extract the organizational memories, event memories of the past and build upon them to create scenarios during training and to offer alternatives during real life situation.

The roles and responsibilities cannot be decided in a pre-defined manner. The system should be able focus on a concise and self-evident design demanded by the small screen orientation and the need to minimize learning.

The interfaces of the response should be spontaneous and it should be able to handle bulk load of data. This should support planning, training, evaluations and system updates between crises.

Tapping the correct information to avoid any information over load during the emergency situation is a critical feature for any emergency systems. This avoids over burning of the emergency personals are they are already over worked during crisis.

It should have templates to fit into situation that can be utilized during a crisis situation. We should remember the fact that during emergencies nothing is normal and the system should be able to generate data due to unusual human responses.

5.3.5 System Interface

The emergency response system will be connected to various other systems of different agencies. The integration between various systems is an essential characteristic of the emergency response system.

It should create the event logs and notification that can be shared by all the involved agencies. The interface of the emergency personals to the information system should be robust which allows interaction between all agencies, depending on the severity of the crisis.

The system should be able to communicate the requirements of the field agents and then get them approved and processes. The system should be able to assign the right privileges dynamically so that it's not waiting for taking an action by the authorized user only.

The system can have some templates for the communication that can be automatically defined by the system by the nature of crisis. These notifications can then be further utilized to the responses of the team.

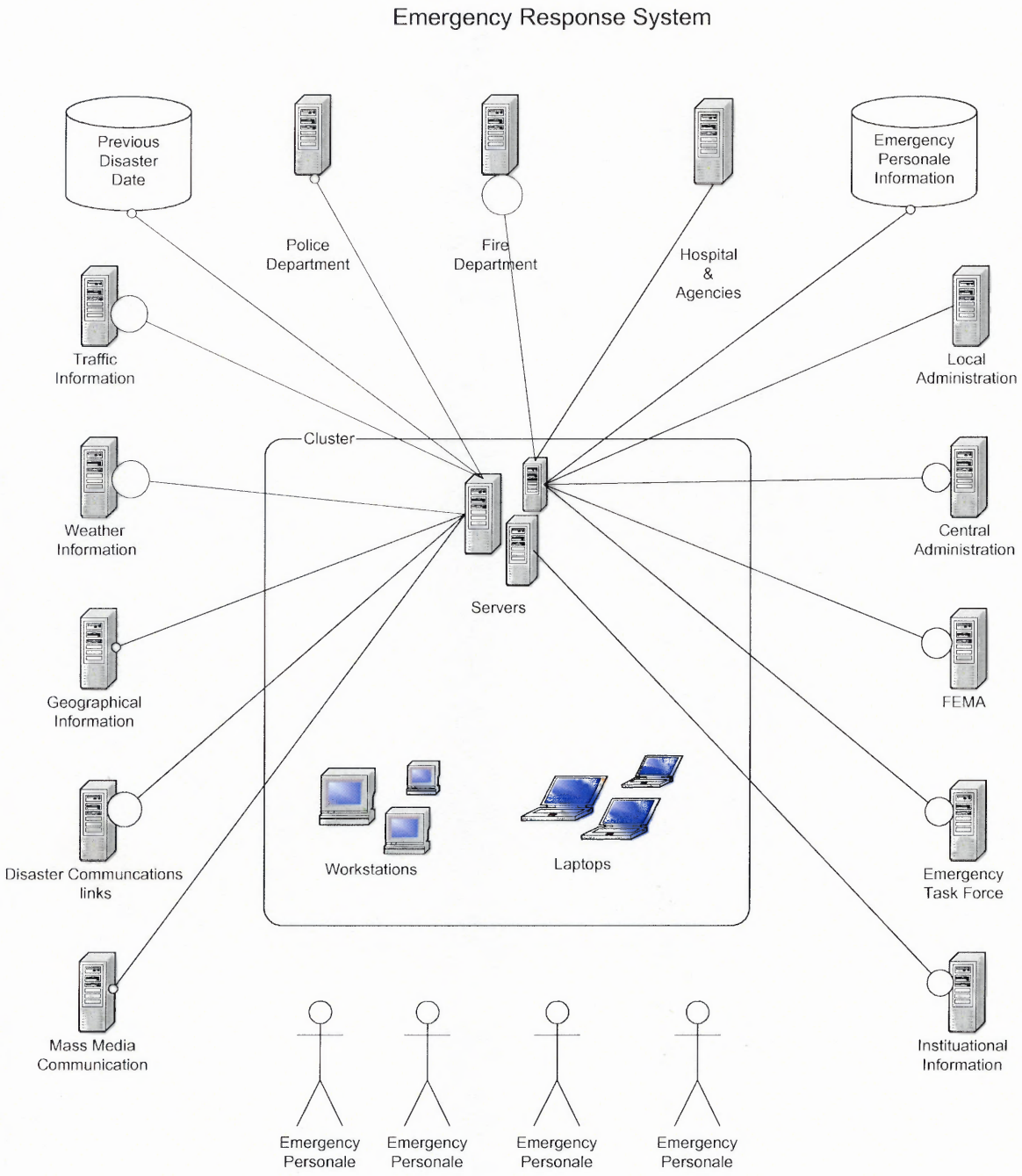


Figure 5.1 Emergency Response System

5.3.6 System Communications

An emergency situation would require communication and coordination among different individual agencies and also many independent aspects of the situation. A typical emergency system should be capable of communicating with police department, fire department, hospitals, local administration, central administration, FEMA, emergency task force, emergency planning institutes, traffic control, weather control center, geographical information center, Telecommunication center, Mass media communication center and mass transport center.

It should be able to support and alert the parties which are drafted as the best resource by the planning section of the emergency response system. According to the After Action Report, communication at the scene was an impediment to efficient emergency response. "Radio traffic overwhelmed the system to the extent that foot messengers became the most reliable means of communicating. Radio communications inside the Pentagon were, for the most part, impossible. Where line of sight could be achieved, 'talk around' was minimally effective" [Titan Systems Corporation, 2002]. Emergency Medical Service (EMS) providers stated they did not reply on radios to transmit information because "ambient noise sometimes made it hard or impossible to talk on the radio" [Titan Systems Corporation, 2002]. Parallel to that, cellular telephones were not useful as calls jammed local towers. The report also noted that deployment information from the Emergency Communications Center (ECC) to emergency response units was delayed and incomplete due to jammed voice oriented communications. It is recommended, "Every firefighter and EMS responder should have a pager to receive dispatch notices both on and off shift" [Titan Systems Corporation, 2002]. It was also

recommended that the EMS units should be equipped with mobile data terminals to transmit and receive information. The After-Action report also suggested that the Emergency Operations Center (EOC) should incorporate computer-based communications that would enable "roistering, automated notification, operators checklists and journals, action tracking, and report generation" [Titan Systems Corporation, 2002]. [40]

5.3.7 Information Management

Conduct the exercise and gather performance assessment of the data. This data can be used to identify the weakness of a particular response and then can be analyzed to improve the response time by incorporating the lessons learned in each exercise.

The system should be able to do meaningful data analysis that can be readily used by the emergency personals during the crisis. It should also analysis the "memories" of individual's actions and alerts them of their common mistakes. This may help the people to respond in a better "learned" manner to the situation.

The system should also be able to identity the best response team for a particular crisis. This can be done with the help of the vast database of the system that stores data about the actions of the people involved in past crisis. Getting the subject matter expert is equally important. The identification and notification upon approval can improve the time required to contact the SME. Such type of response team building can improve the chances of having the best knowledgeable team available.

5.3.8 Role Based Access

The system should avoid information over load to the field officers. There has to be a balance between the information access and the information available to the officers. If

there is less information then it will hinder in making the right decision and if there is more information then the officer might over look the relevant information. The communication system should be able to give role-based access to the information. For example, a fireman would receive not only information relevant to firefighters, but also information on police and EMS activity. This could cause the firefighter to be overloaded with information and possibly cause relevant information to be overlooked. [40] There are multitude of information sources and destination. The role based access with also add the added security required in such big information sharing system.

Once information segregation is introduced in an emergency response system, a need for administration arises. In such a system, there must be an administrative interface, which allows a user or users to determine the necessary groupings of information, and allow/deny access to these groups based on need. The administrative interface must provide both a means for monitoring current group configurations, user privileges, and subsequently changing them.

5.4 Design Principles

In the paper design of DERMIS Dr. Turoff describes the design principles that take care of all the requirements of an ideal Emergency Response System. In this subsection we will describe the design principles as described by Dr. Turoff and discuss them with our view also.

Design Principle 1 - System Directory: The system directory should provide a hierarchical structure for all the data and information currently in the system and provide a complete text search to all or selected subsets of the material. [39]

Dr. Turoff explains a very vivid directory structure for Emergency Response system. He states

A possible structure for the system we have been describing is:

Directory

- People

    Background and Expertise

    Group Memberships

    Conference Memberships

    Bulletin Board Editorships

    Roles

        Responsibilities

            Log Event Creation Privileges

            Current active log events

            Completed log events

            Notifications

            Resource Concerns

        Authorities

    Roles

    Events

    Groups (e.g. medical, firefighters, volunteers, etc)

    Conferences (topic discussions)

    Bulletin boards (Policies, Plans etc)

    Databases (resources, information, local, national, etc)

Learning materials an scenario game generators

Other Emergency Systems

Clearly their needs to be a way to form specialized groups that are focused around certain areas of concern and to have supporting group conferences and message list for these groups. Bulletin Boards represent the semi-static material that a small group of people is responsible for updating.

There is lot of opportunity in this system for smart software to aid the members of the system:

- Letting individuals know who is the subgroup concerned at some point in time with the same situation.
- Finding information that a given individual is not aware of but should be.
- Helping the user to adapt their linkage filters to meet a changing situation and requirements.

The long term success of the system is clearly dependent on features like "smartness" being evolved as part of an on going development process with feedback from real users and real applications [39]

Design Principle 2 - Information Source and Timeliness: In an emergency it is critical that every bit of quantitative or qualitative data brought into the system dealing with the ongoing emergency be identified by its human or database source, by its time of occurrence, and by its status. Also, where appropriate, by its location and by links to whatever it is referring to that already exists within the system. [39]
We learnt from the recent Katrina disaster the importance of transfer of information across the entire group to enable a better coordination about the working groups. The

system should be able to determine the source of information. This information can be used in resource planning by the core team. The allocated resource is answerable for the data it has shared and also can be queried for any updates specifically. Hence the system should be able to trace the data path and the resources at all time. It should be able to generate report on an ad-hoc basis.

Design Principle 3 - Open Multi-Directional Communication: A system such as this must be viewed as an open and flat communication process among all those involved in reacting to the disaster. [39]

The duration and type of emergency is unpredictable. The system should be able to exchange information between any of its modules. It should be able to transfer data from one resource to another. If the emergencies go for longer time then the responsibilities of individual have to be rotated. The system should be capable of transferring the data between individual irrespective of location and type of interface used by the emergency personals. The system should allow greater decision power to more people and should not be following hierarchical order. In emergency situations it's observed that it's not prudent to have a hierarchical decision tree as it takes longer to complete critical functions.

Design Principle 4 - Content as Address: the content of a piece of information is what determines the address. [39]

The information needs to be duplicated and shared among many resources. The system should be able to send the information to the required destinations depending on the type of event that generated the data. This will enable all the responsible personals to be updates of the latest events and information to reach a decision. The system should be capable enough to share the information on its own and intelligently make decision about

the stakeholders of an event. One-way in which a computer system adds a different dimension to data and information that is difficult to duplicate with other forms of communication (Hiltz and Turoff 1978; Turoff 1993). The user should be able to do text searches as and when required to retrieve the required information.

Design Principle 5 - Up-to-Date Information and Data: Data that reaches a user and/or his/her interface device must be update whenever it is viewed on the screen or presented verbally to the user. [39]

System should be able to synchronize data between the master copy of the information and all other systems using it. The information with all the sub-systems should be up-to-date and emergency response system should be capable of adding or deleting new clients on the run.

This is a form of what might be termed "dynamic" linking in that all data exists as a master copy located somewhere in the system which also tracks where in the network of users it also resides. [39] The user does not have time to search for an event of concern and a change of status in an event of concern should just be delivered and presented. [39]

Design Principle 6 - Link Relevant Information and Data: An item of data and its semantic links to other data are treated as one unit of information that is simultaneously created or updated. [39]

The system should be able to link and correlate the information to have a meaningful use. The concept of linking data is critical to the emergency response operation. Any single item of data is associated with numerous attributes and other pieces of data. The user cannot spend the time to contemplate and devise complex search queries. [39]

Design Principle 7 - Authority, Responsibility, and Accountability: Authority in an emergency flows down to where that action is taking place. [39]

The hierarchical structure of authority is not advisable in emergency situation as this creates lot of delays due to trickling of the information from top to down. The field personals should have authority for taking some actions but at the same time they should be accountable for their decisions. The system should give them updated information to help them to make decisions. The system should keep track of the decisions taken by each personal. This data will also help in future learning. There ahs to be clear accountability of who is taking what actions and it should also be clear to all involved when a conflict occurs and how it is being handled. In disaster situations authority is always flowing downwards (Dynes and Qarantell 1977).

Design Principle 8 – Psychological and sociological factors: Encourage and support that psychological and social need of the crisis response team. [39]

Emergency crisis are nothing about normal situations. The unforeseen circumstances and unusual activities bring lot of pressure on the field officers. The system should assist in encouraging social activities by allowing people to know each other and relax. The system must allow for a "team spirit" to develop. People must get to know one another well enough so that they have no qualms about handling over their role to another person. [39] There should be feeling of trust among the persons.

When an emergency system is employed as a dispersed virtual command center, this consideration becomes critical. There is a strong need to be able to rely on one another and to accept frankness in viewpoints as the common norm. The user should be

able to adapt the system to his or her method of cognitive problem solving and not inhibit creativity or improvisation in unique problem situations.

Design Principle 9 – Notification and correlation: System should have multiple ways of sending the notifications to the emergency personals. It should support addition of new ad-hoc networks.

Notifications are the pivotal point in any emergency situation. The system should support sending notifications to the field officers in multiple ways. If one type of communication fails then there should be a fail over means to communicate to field officials. The notifications should be send using pagers, cell phone, text messages, voice messages, emails and it should also make sure that it's not using the same network to send all the messages as there is a danger of information over flow at the communication towers if all info flows thru them.

5.5 Summary

In this chapter we studies about Emergency Response System. We outlined and discussed the requirements of such a system. We established that the communication is one of the most important concepts of any Emergency Response System. The design principles were also formulated at a conceptual level. We tried to club the information established in various papers at one place to present a complete picture of the requirements and design of Emergency Response.

CHAPTER 6

FRAMEWORK

In the previous chapter we had discussed the overview of an emergency response system and had also established the functional requirement and design principle of an ideal Emergency Response System. In this chapter we will propose a framework; we will call it as ERSD Framework (Emergency Response System Design Framework) for designing an Emergency Response System based on our study of Model Driven Architecture and Zachman Framework.

We'll map the major activities that will be required for the design of the emergency response system to Zachman Framework. This new ERSD framework will act as a checklist for designers of emergency response systems.

6.1 Design Process

The most common activities that a designer need to perform while designing and developing any software system is shown in fig 6.1.

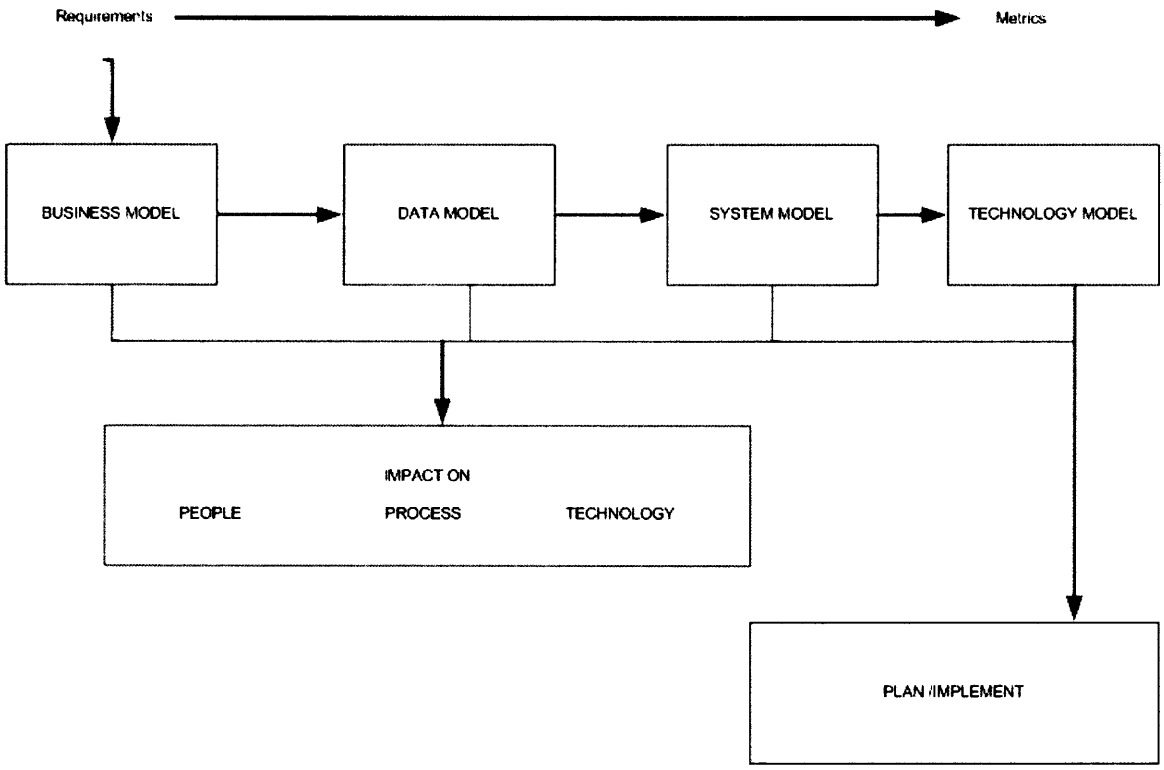


Figure 6.1 Design Activities

A designer will enlist the domain characteristics and gather business requirements. He may do it with some help of artifacts for e.g. Use Cases. From the business use cases developed during the phase of requirement gathering he can then design the business model that will identify the key business processes and the internal and external entities that are important to an enterprise. Next logical step is to analyze the data that will be processed by the system under design. From our Business and data model we can derive the design of the system model that will be the technical view of the system. System models are independent of the technology and implementation details. The view from the above models can be used to design the actual implementation framework of the system. These are the high level activities for designing a system as

defined by the MDA process. In the next section we will break these high level system design activities as per Zachman's framework for an emergency response system.

6.2 ERSD Framework

The ERSD framework is the bird's eye view of the artifacts required to develop an emergency response system. This framework will act as the "list of things" to implement such a system. Table 6.1 highlights the activities that one need to perform while designing an emergency response system.

Cell Definitions

Column 1: The "What" or "Data" Column

Row 1: List of Business Things

Zachman's view: This is simply a list of things that the enterprise is interested in – the "universe of discourse" relative to things. [42]

ERSD view: The universe of Emergency Response system revolves around information like Demographic information, System Information, Emergency personnel information, Historical information, Real-time event information.

ERSD FRAMEWORK	DATA	FUNCTION	NETWORK	PEOPLE	TIME	MOTIVATIO N
CONTEXTUAL MODEL	Past event memories, Authority details	System capabilities	Commands Center Locations	Emergency personnel	Type of Disasters	Criteria of Activating the systems
BUSINESS MODEL	Directory Functional Model	Use Case Model (Requirements)	Interconnection s details	Roles and their relationships	Service Level Agreements /ETA	Requirements
SYSTEM MODEL	Directory informational and naming structure	Data Flow diagrams or sequence charts or activity flow diagrams	Network Diagrams	Interface Design	Event Transitions	Design Principles
TECHNOLOGY MODEL	LDAP / X.500	Detailed Design based on Specific technology	Hardware & software specifications & Types of links	Interface Technology like pagers	Event processing cycles	Design principles / domain patterns
Data Model	Directory schema	Implementation or Low level Design	Network Map, Node Address & link protocol	AAA specifications	Response Time	Design patterns

Table 6.1 ERSD Framework.

Row 2: Semantic Model

Zachman's view: This is model of the actual enterprise things that are significant to the actual enterprise. [42]

ERSD view: This will contain the directory structure of emergency response system. More specifically this will contain the functional model of the directory of the system. An example of such a view is shown in the figure 6.2 below:

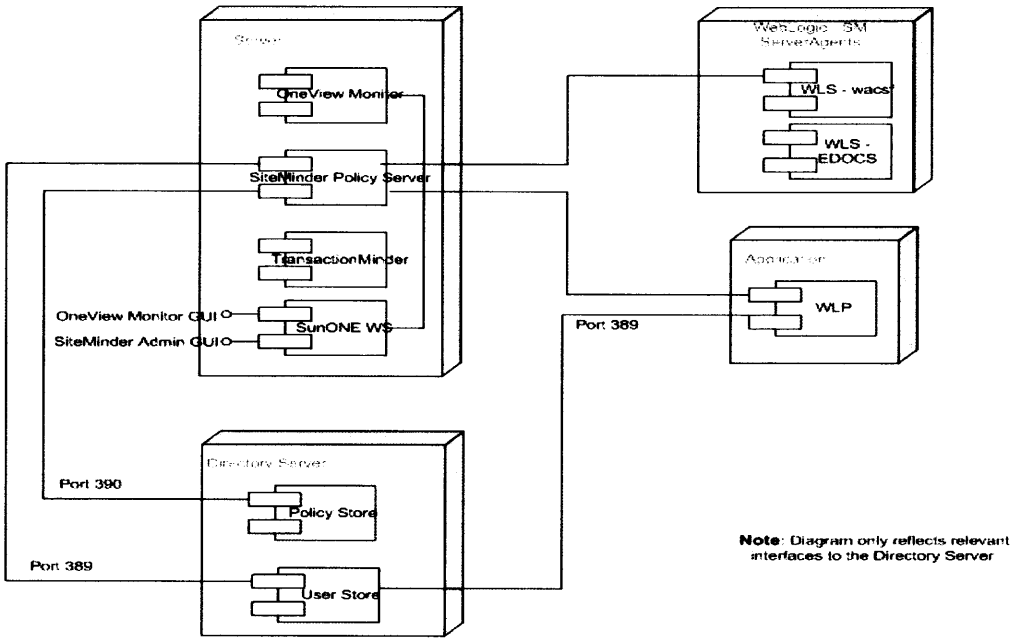


Figure 6.2 Directory's Functional Model

Row 3: Logical Data Model

Zachman's view: This is the technology neutral logical representation of the things of the enterprise about which it records information. [42]

ERSD view: This will represent the directory information tree structure and also describes the information model of the emergency response system central directory. Fig 6.3 shows one such example of directory naming and information model.

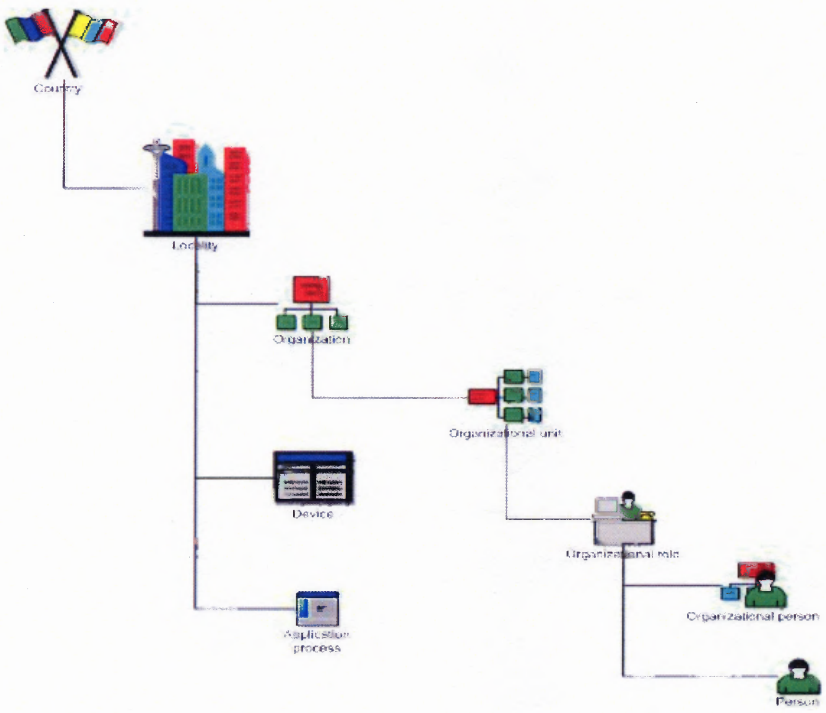


Figure 6.3 Directory Information Model

Row 4: Physical Data Model

Zachman's view: This is technology constrained, or physical representation of the things of the enterprise. [42]

ERSD view: This cell will contain technology specific design of directory. Directory can be based on LDAP or X.500 standards. Both of these standards will have their own constraints that a designer need to follow while designing technology specific DSA and their distribution model like replication details. Fig 6.4 shows one such example

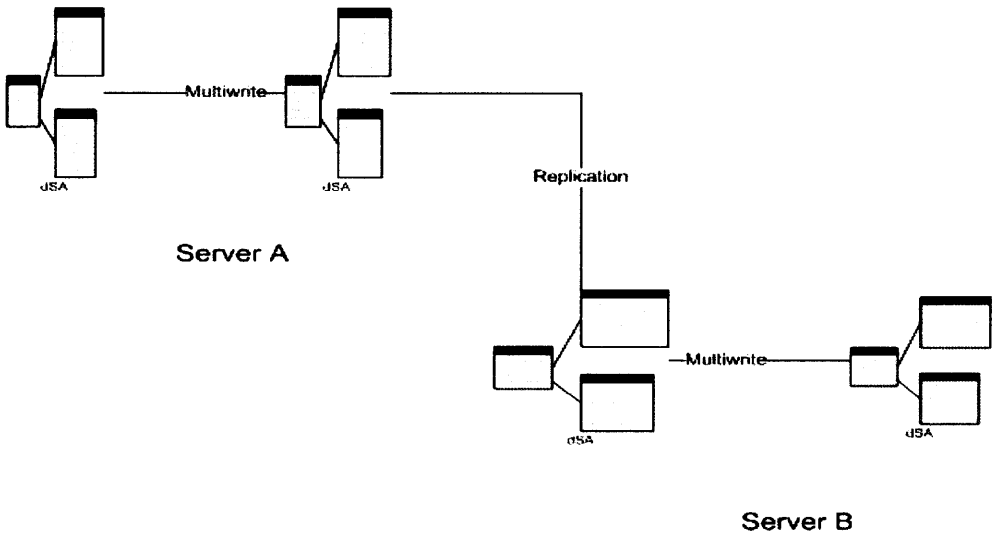


Figure 6.4 Directory DSA Design

Row 5: Data definitions

Zachman's view: This would be the definitions of all the data objects specified by the physical data model and would include all the data definition language required for implementation. [42]

ERSD view: This cell will contain directory schema definition.

Column 2: The "How", or "Process" column

Row 1: Semantic Model

Zachman's view: This is simply a list of processes (or functions) that the enterprise performs – the " universe of disclosure" relative to process, the " transformation" of enterprise " inputs" or "outputs".[42]

ERSD view: This cell will contain the main capabilities of the emergency response systems. Example few of such business processes are sending and receiving notifications , remote database searching capabilities, self learning capability of the system, simulation of events for training purposes, seamless communications between

different departments involved in emergency. These functionalities are described in detail in the previous chapter under the section of characteristics of ideal emergency response system

Row 2: Business Process Model

Zachman's view: This model of the actual business processes that the enterprise performs, quite independent of any "system" or implementation considerations and any or organizational constraints. [42]

ERSD view: This will define the use cases for the required functionality described in row 1 column 2 of the framework. For example in case of fire emergencies who all will receive notification. How will system respond to the search started by a field agent? Such kind of use cases should be elaborated in this cell.

Row 3: Application Architecture

Zachman's view: This is a model of the logical (implementation –technology neutral) "systems" implementation (manual and or automated) supporting the business processes and would express the "human/machine" boundaries. [42]

ERSD view: This cell will elaborate the data flow for the application functions and established architecture for each function. For example, this cell will describes how a notification will be received and transmitted to all the required parties. This will show how a particular notification will decided its destinations by the help of the sequence diagrams. Such sequence diagrams should be developed for all business processes described in use cases in row 2 column 2 of the framework.

Row 4: System Design

Zachman's view: This is a model of the logical (implementation – technology neutral) "systems" implementation supporting the business processes and would express the "human/machine" boundaries. [42]

ERSD view: This will decide the technology implementation for the functions described in previous cells. For example how an existing wireless network will be leveraged to send the data / voice notifications to the remote user. An interface build in java using J2ME APIs will be modeled to provide a simple interface on the mobile devices. The name of the APIs should be identified in this cell.

Row 5: Program

Zachman's view: These would be the programs that derive from the "Action Diagram" – style or Object-style specifications for the implementations. [42]

ERSD view: This cell will contain actual implementation design.

Column 3: The "Where" or "Network" column

Row 1: List of Business Locations

Zachman's view: This is simply a list of locations in which the Enterprise operates, or relates to-the "universe of discourse" relative to location. [42]

ERSD view: For emergency response system this cell defines the command center location, field locations and back office locations. For example city halls, moveable command center, locations of police stations, fire stations and hospitals.

Row 2: Business Logistics System

Zachman's view: This is a model of the locations of the enterprise and their connections whether the connection are voice , data , post or truck ,rail , ship.[42]

ERSD view: This cell will describe the nodes and the hierarchal structure of each location. The emergency command center locations and their linkages are identified and established. Each node is given a priority for service in case of emergency.

Row 3: Distributed System Architecture

Zachman's view: This is a logical model of the system implementation of the business logistics system depicting the types of the systems facilities and controlling software at the node and lines. [42]

ERSD view: Here the architecture is drawn linking each node established in previous steps. The network architecture describes the type of storage and connectivity. For example a nodal diagram is drawn for the servers of police department and fire department. Protocols are established for the communication between them.

Row 4: Technology Architecture

Zachman's view: This is the physical depiction of the technology environment for the enterprise showing the actual hardware and system software at the nodes and the lines and their systems software including operating systems and middleware. [42]

ERSD view: This cell will contain the detail specification of each hardware and software that will be the part of the system. For example server at police headquarter has how much, memory, processing speed, operating system details and so on. It will also specify the link details whether it's T1 or T3 or VPN link over public network and so on.

Row 5: Network Architecture

Zachman's view: This is the specific definition of the node and the line identification. [42]

ERSD view: This cell will contain the address details of each node for example the MAC address and IP address of each server located at the police headquarter. It will also specify the protocol of each link whether its ATM or Frame Relay or IP.

Column 4: The "Who" or "People" Column

Row 1: List of Business Organizations

Zachman's view: This is simply a list of organizations to which the Enterprise assigns responsibility for work – the "universe of discourse" relative to people. [42]

ERSD view: This cell defines the list of all the departments and people who will interact during an emergency response.

Row 2: Work Flow Model

Zachman's view: This is the model of the actual Enterprise allocations of responsibilities and specifications of work products. [42]

ERSD view: This cell lays down the structure in which the departments and people interacts which each other. In emergency system it recommended not to follow the conventional hierarchal structure for communication.

Row 3: Human Interface Architecture

Zachman's view: This is the logical "systems" expression of work flow which would include the specifications of the "roles" of responsible parties including management, administration, knowledge-worker, engineering, marketing etc. as well as the logical specification of the work products like, voice, text, graphics, video, etc . [42]

ERSD view: This view will explain vividly the roles of each personal during the emergency. The authority of people is established and they are joined to network with their specialty.

Row 4: Presentation Architecture

Zachman's view: This is the physical expression of work flow of the Enterprise including the specific individual and their ergonomic requirements and the presentation format of the work product. [42]

ERSD view: This view describes the privileges of each person and the type of interfaces which they will be using to access the information.

Row 5: Security Architecture

Zachman's view: The "out-of-context" specifications of work flow would be the identification of the individual accessing the system and the specification of the work or job they were authorized to initiate. [42]

ERSD view: This view will describe Authentication, Authorization and Accounting details of all the nodes. It will also specify how the role base access requirements will be met and implemented

Column 5: The "When" or the "Time" Column

Row 1: List of Business Events

Zachman's view: This is simply a list of events to which the Enterprise responds – the "universe of discourse" relative to time. [42]

ERSD view: This view will enlist the events what that will trigger the emergency response. Like in the event of major fire a different subnet of emergency response system will be invoked.

Row 2: Master Schedule

Zachman's view: This is a master of the business cycles that is comprised of an initiating event and an elapsed time. [42]

ERSD view: The timing is very critical in any emergency response system. The cell defines the timing definition for triggering each event.

Row 3: Processing Structure

Zachman's view: This is the logical systems specification of points in time and lengths of time (processing cycles). [42]

ERSD view: This cell will specify how the triggering event changes the state of the emergency system. This can be elaborated by State Charts.

Row 4: Control Structure

Zachman's view: This is the physical expression of system events and physical processing cycles, expressed as control structures, passing controls from one to another processing module. [42]

ERSD view: This cell will describe the event processing cycle of an emergency response system.

Row 5: Timing Definition

Zachman's view: This is the definition of interrupts and machine cycles. [42]

ERSD view: This cell will define the calculated response time of the emergency response system with respect to an event.

Column 6: The "Why" or "Motivation Column"

Row 1: List of Business Goals /Strategies

Zachman's view: This is simply a list of major business goals that are significant to the Enterprise and defines the "universe of discourse". [42]

ERSD view: This cell contain the list of the disaster events that can happen both natural and man made.

Row 2: Business Plan

Zachman's view: This is a model of the business objectives and the strategies of the enterprise that constitute the motivation behind Enterprise operations and decisions. [42]

ERSD view: This cell will contain the requirements of the ideal system that has been proposed in the previous chapter.

Row 3: Business Rules

Zachman's view: This is a logical model of the business rules of the enterprise terms of their intent and the constraints. [42]

ERSD view: This cell will contain the design principles outlined in the previous chapter.

Row 4: Rule Design

Zachman's view: This is a physical specification of the business rules. The rules are not presently factored out from their implementations and therefore are found as cardinality and optionality in the data models, as procedural code or as policy specification. [42]

ERSD view: This cell will contain the design principles and also the design patterns.

Row 5: Rule Specification

Zachman's view: This will be the "out-of-context" specification of the business rules. [42]

ERSD view: This cell will contain the low level design patterns.

CHAPTER 7

CONCLUSIONS

We have described a software development landscape. In this discussing we laid down the advantage of abstraction in a software development lifecycle. Modeling is powerful concept to visualize any system. It is established why modeling is important and how can it be leveraged to achieve a more coherent soft wares. Unified Modeling Language is a set of specification defined by Object Modeling Group. We have defined various profiles which derived from UML.

Model Driven Architecture defines modeling the enterprise around system independent model and system specific model. The requirements are captured in Computational Independent Model and then platform independent model is created. The next step is to map the platform independent model to a platform specific model. Here we have tool support which can model a particular technology independent model to a technology specific model. We learnt that we have lots of industry support for MDA. Various modeling tools like IBM Rational Rose, Rose RT, and I-Logix Rhapsody. There are standard profiles defined by OMG to achieve such modeling goals.

We established the importance of enterprise architecture and how it can be achieved. We learnt that MDA is a very important concept in achieving today's enterprise architecture as it helps us separate the business logic from the implementation methodology in a clear way. Enterprise Architecture design principles were describes to develop the appreciation of the architecture. Zachman's framework is one of the most commonly used frameworks in today's industry. We discuss the reasons to its popularity and its guidelines to develop and enterprise architecture were also laid. The framework

was established which can use the artifacts described by MDA to describe the cells of the Zachman's framework in an architecture.

We did a case study of developing an Emergency Response System Design Framework. This framework can act as a checklist for development of any emergency system. An ideal emergency response system was established and then the MDA artifacts and Zachman's framework was used to develop an ERSD Framework.

CHAPTER 8

CONTRIBUTIONS

Emergency response system has always been an important aspect of human life in case of crisis. The recent disastrous events like the 9/11 and hurricane Katrina have brought the importance of emergency response system to the forefront. Leveraging the current technologies to efficiently develop reliable emergency response systems with high availability can be very beneficial and help saving many lives as we can achieve well coordinated efforts in less time.

Based on the literature review we know that well defined architectures and enterprise architectures in particular are critical to the successful development, deployment and maintenance of complex, scalable integrated systems with multiple users, responsibilities, and goals. We have studied the state of the art in defining enterprise architectures. We have further examined the current approaches to defining and designing Emergency Response Systems and the application of MDA approach to support this task and have defined an Emergency Response System Design Framework, which can provide guidance when architecting, developing and deploying an Emergency Response System. Our ERSD Framework is a check list of views, recommended models and best practices that allow for a comprehensive definition of an Emergency Response System.

REFERENCES

1. Grandy Booch, MDA: A Motivated Manifesto and Software Development, Software Development Magazine, August 2004.
2. John Daniels, Modeling with a Sense of Purpose, IEEE Software, February 2002.
3. David S. Frankel, BPT Column, MDA Journal, September 2003.
4. Anders Lidbeck, Super Models, CBR Research, October 2002.
5. Gray Crenosek, The Value of Modeling, Rational Developers Works, Oct 2004.
6. Paul Harmon, UML Models E-Business, Softwaremag.com, January 2005.
7. Terry Quatrani, Introduction to UML, Rational Software, June 2003.
8. OMG: Object Management Group, Meta Object Facility Specification V1.4, April 2002.
9. Stephen J. Mellor, MDA Distilled: Principles of Model-Driven Architecture.
10. Ruiz Francisco, Using XMI and MOF for Representation and Interchange of Software Processes, IEEE, 2003.
11. Caruso Francesco, Architectures to Survive Technological and Business Turbulences, Information Systems Frontiers, September 2004.
12. Joaquin Miller, MDA Guide Version 1.0.1, OMG Press, June 2003.
13. Frank J. Armour, A Big-Picture Look at Enterprise Architecture, IEEE IT Pro, Feb 1999.
14. Frank J. Armour, Building Enterprise Architecture Step by Step, IEEE IT Pro, August 1999.
15. Tony Brown, Value of Enterprise Architecture, www.zifa.com.
16. Michael Chung, Enterprise Architecture, Implementation, and Infrastructure Management, IEEE Conference, 2002.
17. Kaisler, Enterprise Architecture: Critical Problems, IEEE Conference 2005
18. Frank J. Armour, A UML-Driven Enterprise Architecture Case Study, IEEE Conference, 2002.

19. Robert France, Bernhard Rumpe, In Search of Effective Design Abstractions, Springer-Verlag 2004.

20. Zudin Dzafic, Mevludin Glavic, and Sejid Tesnjak: A Component Based Power System Model-Driven Architecture. IEEE Transactions on power systems Vol 19 Nov.2004.

21. James Skene and Wolfgang Emmerich, A Model Driven Approach to Non-Functional Analysis of Software Architectures, Proceeding of the 18th IEEE International Conference on Automated Software Engineering 2003.

22. Jana Koehler, Rainer Hauser, A Model Driven Transformation Method, Proceedings of the Seventh IEEE International Enterprise Distributed Object Computing Conference 2003.

23. Kevin L Moore and John K Abraham, An Architecture for Intelligent Decision Support with Applications to Emergency Management, IEEE 1994.

24. Barry M Horowitz, Stephen D Patek, Integrated Peer-to-Peer Applications for Advanced Emergency Response Systems Par II Technical Feasibility, Proceedings of the 2003 Systems and Information Engineering Design Symposium.

25. Leo Frishberg, Looking Back at Plan AHEAD: Exercising User Centered Design In Emergency Management, CHI 2005.

26. Murray Turoff, Assuring Homeland Security: Continuous Monitoring, Control and Assurance of Emergency Preparedness, http://web.njit.edu/~turoff/#a5, Oct. 2005.

27. Sisi Zlatanova, Proposed System Architecture for Emergency Response in Urban Areas in Direction Magazine Oct 2005.

28. J. T. Ryan and C. N. Dillard, A Computer Model of the Total Emergency System, ACM 2002.

29. Ingmar Rauschert, Pyush Agrawal, Rajeev Sharma, Designing a Human-Centered, Multimodal GIS Interface Support Emergency Management, ACM 2002.

30. Yufei Yuan & Brian Detlor, Intelligent Mobile Crisis Response System, Communications of the ACM February 2005.

31. K. K. A. Zahid, L. Jun, M. Matsumoto, IP Network for Emergency Service, ACM International Conference Proceedings, 2004.

32. Murray Turoff, Past and Future Emergency Response Information Systems, Communications of the ACM April 2002/Vol. 45, No. 4.

33. J.A.Zachman. "A Framework for information Systems Architecture," IBM Systems Journal, Vol26, No.3, 1987.

34. Published in the electronic book *The Zachman Framework*. The book is available at www.zachmaninternational.com, 2005.

35. John A.Zachman, A Framework for Information Systems Architecture, IBM Systems Journal, Vol. 26, No. 3, 1987.

36. John A.Zachman, The Framework of Enterprise Architecture: Background, Description and Utility, www.zifa.com, Oct. 2005.

37. David S. Frankel, The Zachman Framework and MDA, OMG White Paper, Sep 2003.

38. Murray Turoff, Design of DERMIS, JITTA, http://web.njit.edu/~turoff/#a5, Oct. 2005.

39. Altaf S. Bahora, Integrated Peer-To-Peer applications for Advanced Emergency Response System-Part1: Concepts of Operations, IEEE 2003.

40. Altaf S. Bahora, Integrated Peer-To-Peer applications for Advanced Emergency Response System-Part2: Technical Feasibility, IEEE 2003.

41. John A.Zachman, The framework of Enterprise Architecture Cell Definitions, www.zifa.com, Oct 2005.