INCOSE Italian Chapter Conference on Systems Engineering (CIISE2014)
Rome, Italy, November 24 – 25, 2014

A Methodological Template for Model Driven Systems Engineering

P. Bocciarelli, A. D'Ambrogio, E. Caponi, A. Giglio and E. Paglia
Dept. Enterprise Engineering, University of Rome "Tor Vergata"
{paolo.bocciarelli,dambro,andrea.giglio,emiliano.paglia@uniroma2.it}

Copyright © held by the authors.

Abstract. The advent of formal modeling languages (e.g., UML and SysML) and system architecture frameworks (e.g., DoDAF and MODAF) has given systems engineers the ability to effectively describe the requirements as well as the behavior and the structure of systems. Approaches founded on the use of modeling languages and frameworks are grouped under the banner of MBSE (Model Based Systems Engineering). The basic idea is that a model evolves over the system development life-cycle, until it becomes the built-to baseline. In this paper, we consider a modeling approach based on the use of a metamodeling architecture that focuses on the use of models as the primary artifacts of system development. We specifically address the use of MDA (Model Driven Architecture), which allows to increase the level of automation when evolving models from the very abstract representation of a system down to the system implementation, thus making easier (i.e., at reduced cost and effort) the analysis, development and testing activities. By applying MDA concepts and standards to MBSE approaches we obtain what we refer to as MDSE (Model Driven Systems Engineering). The paper illustrates a methodological template for MDSE and shows its application to the development of a software-intensive system.

1. Introduction

Systems engineering (*SE*) focuses on producing and maintaining successful systems that meet user's requirements and development objectives. In order to achieve this capacity, the systems engineer must balance superior performance with affordability and schedule constraints. In this respect, an important subset of systems engineer's activities is *systems architecting*. This term refers to the translation from the operational requirements defined by the stakeholders into a system's model that must be validated and defined as the built-to baseline. The strong cohesion between systems architecting and modeling has allowed the extensive use of architectures in large, complex system development programs. Initially, these architectures were technically accurate but diverse in their structure.

In order to standardize the architectures, many organizations developed and mandated the use of *architectural frameworks*, such as DODAF [1] and TOGAF [2], which prescribe a structural approach and principles for developing a system architecture. Such frameworks use models to represent aspects, perspectives and views of the system. Traditional models, like standard block diagramming techniques, are based on top-down decomposition of a system. These methods are typically functionally based and are formed into a hierarchy of models representing attributes of system in increasing levels of detail. With the evolution of systems architecting, during the nineties, the need for a formal modeling language was recognized as beneficial to establish a consistent standard. The International Council of Systems Engineering (INCOSE) commissioned an effort in 2001 to develop a standard modeling language. The attainment of this effort was the *Systems Modeling Language* (*SysML*) [3], a systems engineering extension to the well-known and widely used *Unified Modeling Language* (*UML*) [4].

Architectural frameworks, UML and SysML can be grouped under the banner of what is referred to as *Model Based Systems Engineering* (*MBSE*) [5]. The basic idea behind MBSE is that a model evolves over the system development life-cycle, until it becomes the built-to baseline. Early in the life-cycle, the models have low levels of fidelity and are used for decision making. As the system is developed, the level of fidelity increases until the models can be used for design. Finally, they are transformed yet again into the build-to baseline. Therefore, the MBSE methodologies provide the required insight into the analysis and design phases, enhancing the communication between the different participants and enabling effective management of the system complexity.

Unfortunately, the MBSE methodologies have limitations in terms of lack of practices and tools that automate the models development. In order to overcome these limitation, this paper exploits principles and standards introduced in the *model-driven engineering* field.

Model-driven engineering supports systems development activities through the automated transformation of abstract models to operational components and applications. The most used incarnation of model-driven engineering has been provided by the Object Management Group (OMG) through the *Model Driven Architecture* (*MDA*) [6], which emphasizes the role of models as the primary artifacts of software-intensive systems development.

This paper illustrates a methodological template for applying MDA principles and standards to MBSE approaches, thus obtaining what we refer to as *Model Driven Systems Engineering* (*MDSE*).

The rest of the paper is organized as follows: in Section 2 we provide a brief summary of MBSE and MDA. In Section 3 we discuss the proposed MDSE template, which is then applied to the analysis and development of a software-intensive system in the naval electronic warfare domain, as shown in Section 4.

2. Background

The following subsections overview the basic concepts at the basis of MBSE and MDA, respectively.

2.1 MBSE

Model-based systems engineering (MBSE) is defined as the *"formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases"* [7].

The main purpose of MBSE is to provide a *methodology*, which can be defined as a collection of related processes, methods, and tools.

In [8] a *Process (P)* is defined as a logical sequence of tasks performed to achieve a particular objective, a *Method (M)* is the set of techniques for performing a task and a *Tool (T)* is an instrument that, when applied to a particular method, can enhance the efficiency of the task.

A *methodology* is essentially a “recipe” and can be thought of as the application of related processes, methods, and tools to a class of problems that have something in common [9].
The environment dimension can be added to the aforementioned ones. An *Environment (E)* consists of the surroundings, the external objects, conditions, or factors that influence the actions of an object, individual person or group [8].
The traditional approach identifies these factors as the "PMTE" elements (Process, Methods, Tools, and Environment) of a methodology. Furthermore, it suggests to consider also the capabilities and limitations of technology when setting up a systems engineering development environment, as stated in [8].
Taking these considerations into account, and with the aim to get a step forward, in our work we present an extension of this methodology (see Section 3), which also includes other dimensions related to techniques and languages relevant to the model-driven engineering domain.
As regards the state of the art of the main MBSE methodologies, the INCOSE MBSE Focus Group has identified the following list of methods [7]:

- Telelogic’s Harmony – SE
- INCOSE’s Object Oriented Systems Engineering Method (OOSEM)
- IBM’s Rational Unified Process for Systems Engineering (RUP-SE)
- Vitech’s MBSE Methodology
- Jet Propulsion Laboratory’s (JPL) State Analysis (SA).

For a more detailed discussion the reader may refer to [5],[7].

2.2 MDA

Model-driven engineering (MDE) is an approach to software design and implementation that addresses the raising complexity of execution platforms, by focusing on the use of formal models [10],[11]. According to this paradigm, a software system is initially specified by use of models at an high level of abstraction. Such models are then used to generate other models at a lower level of abstraction, which are used in turn to generate other models, until stepwise refined models can be made executable. One of the most important incarnation of MDE principles is the Model Driven Architecture (MDA) [6], the OMG's incarnation of MDE.
MDA-based software development is founded on the principle that a system can be built by specifying a set of model transformations that map the elements of a source model, which conforms to a specific metamodel, to elements of another model, the target model, which conforms to the same or to a different metamodel.
To achieve such an objective, MDA introduces the following standards:

- *Meta Object Facility (MOF)*: for specifying technology neutral metamodels (i.e., models used to describe other models) [12];
- *XML Metadata Interchange (XMI)*: for serializing MOF metamodels and models into XML-based schemas and documents, respectively [13];
- *Query/View/Transformation (QVT)*: for specifying model transformations [14].

The relationship among MDA standards is summarized by the scenario depicted in Figure 1. Model M_A and model M_B are instances of their respective metamodels, namely metamodel MM_A and metamodel MM_B, which in turn are expressed in terms of MOF constructs. A model transformation, specified at metamodel level using QVT, allows one to generate model M_B from model M_A. Both model M_A and model M_B can be serialized using XMI rules, to obtain the corresponding XMI documents, describing such models in XML language. XMI rules can also be used at metamodel layer to serialize metamodels and obtain XMI schemas for XMI document validation.

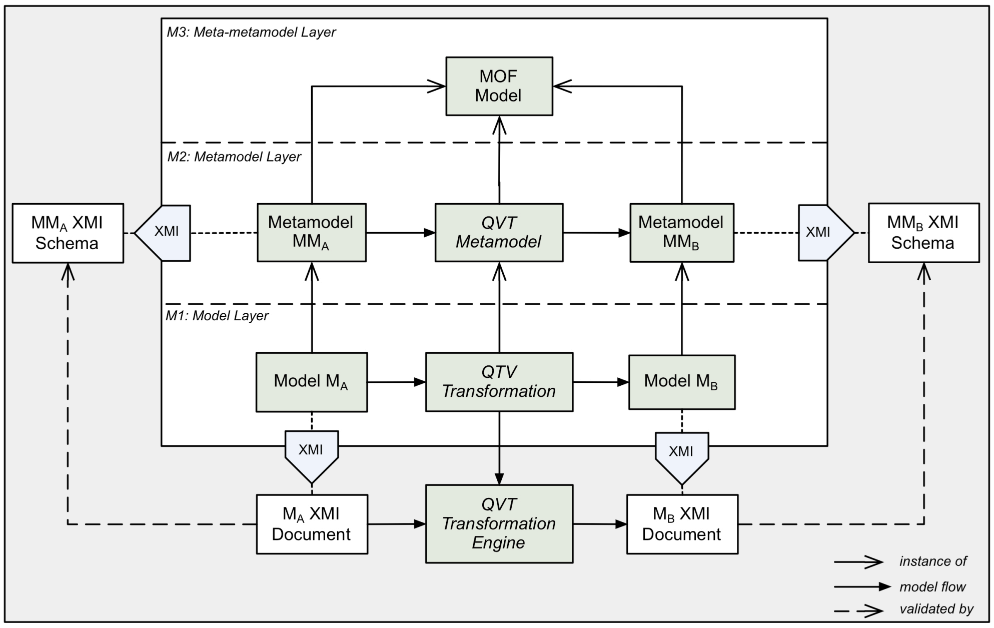

Figure 1. Overview of MDA standards

3. Methodological template for MDSE

This section illustrates the proposed methodological template for introducing MDSE approaches, which are strongly based on the productive use of models.

For such a reason, in order to distinguish the proposed approach from the one defined in Section 2.1, we introduce the MDSE template that helps to organize the relevant areas of knowledge. It is defined as quintuple of the form <*Em, Pm, Mtm, Tm, Lm*> where:

- *Environment (Em)* is the set of semi-formal models that assist the team participants in grasping the abstract concepts related to SE.
- *Process (Pm)* is the set of the processes that describes the evolution of a particular new system. The stepwise evolution of each process is referred to as the *system life-cycle model*, which subdivides the system's development process into a set of basic steps and phases.
- *Method (Mm)* is the set of activities that are iterated in each phase of the process. We refer to this dimension with the term *system methods model.*
- *Technique (Tm)* is the set of standards that support processes and methods;
- *Language (Pm)* is the set of languages that ensures the formal and correct manipulation of the models.

The following subsections describe the proposed MDSE template dimensions.

3.1 Environment dimension

In this dimension, we shall use models to grasp the abstract concepts related to SE efforts. One of these is related to the domain of responsibilities for the systems engineer. Informally, a systems engineer must attain a broad knowledge of the several disciplines involved in the development of a complex system. Clearly, it cannot be as deep as the knowledge possessed by the specialists in these areas. However, this definition raises the question: how much broad and deep is the knowledge for a systems engineer? For the purpose of illustrating the typical scope of a systems engineer's responsibilities, it is useful to create a specific model for the complex systems. By its nature, it consists of a number of major interacting elements, generally called subsystems, which themselves are composed of more detailed entities, and so on, down to primitive entities usually referred to as parts. In our work, each complex system can be subdivided into five levels:

- *System (L1)*, a set of interrelated elements that work together to achieve a specific objective;
- *Sub-Systems (L2)*, the major portions of the system that perform a closely related subset of the overall system functions;
- *Components (L3),* the middle-level entities of system that perform a specific functionality;
- *Sub-Components (L4)*, which perform elementary functions and are composed of several parts;
- *Parts (L5)*, which perform functions only in combination with other parts.

For this reason, our model is also known as *5-Level Hierarchical Model*. The knowledge domains of a systems engineer extend from the highest level, the system and its environment, down through the middle level of components. At the same time, the design specialist’s knowledge extends from the lowest level of parts up through the components level, at which point their two knowledge domains “overlap”. This is the level at which the systems engineer and the design specialist must communicate effectively, identify and discuss technical problems and negotiate workable solutions.

3.2 Process dimension

The evolution of a particular new system from the time when a need for it is recognized, through its development and operational use, is a complex effort, that is called system development process. Typically, it requires several years to complete, it is made up of many tasks and has a specific schedule and budget. Furthermore, the introduction of a new technology inevitably involves risks, which must be identified and resolved as early as possible. All these factors require that the system development must be conducted in a step-by-step manner, in which the success of each step is demonstrated, and the basis for the next one validated, before a decision is made to proceed to the next step.
The term *system life-cycle model* is commonly used to refer to the organization of the stepwise evolution of a new system, from concept through development, operation, maintenance and disposal. The set of lifecycle models subdivide the system life into a set of basic steps that separate major decision milestones. Therefore, the derivation of a life-cycle model is a useful tool to organize the building of complex systems.
The second dimension for our MDSE Template is embodied by the *SE life-cycle model* [15], which consists of three stages and eight phases. The first two stages encompass the developmental part of the life-cycle, while the third the maintenance part. Such stages will be referred to as *(1) concept development* stage, which is the initial stage of the formulation and definition of a system concept to best satisfy a valid need; *(2) engineering development* stage, which covers the translation of the system concept into a validated physical system design meeting the operational, cost and schedule requirements; and *(3) post-development* stage, which includes deployment and support of the system throughout its useful life.

The *concept development* stage, as the name implies, embodies the analysis and planning that is necessary to establish the need for a new system. SE plays the lead role in translating the operational needs into technically and economically feasible system concept. This activity, referred to as *systems architecting* is illustrated in the MDSE example application discussed in Section 4.
The *engineering development* stage corresponds to the process of engineering the system to perform in the operational environment the functions specified in the system concept.
Finally, the *post-development* stage starts soon after the system successfully undergoes its acceptance testing and is released for operational use.

3.3 Method dimension

While each phase of the process is designed for specific problems, it is possible point out a set of activities that tends to iterate from a given phase to the next one. In this paper we use an iterative set of activities that will be referred to as the *systems engineering (SE) method model*. It can be thought of as the systematic application of the scientific method to the engineering of a complex system. It consists of the following basic activities: *requirements analysis, functional definition, physical definition* and *design validation*.
Such four activities are carried out in each phase of *SE life-cycle model* and vary in their specifics depending on the type of the system and the phase of its development.

3.4 Technique dimension

In this paper, model-driven engineering is considered a special technique that uses the model as the primary artifact to support the construction of complex systems. The term "technique" is used to emphasize the study and the description of the *Tm* dimension related to the *Mm* dimension. As aforementioned, the goal of model-driven engineering is to increase the level of automation of the processes and facilitate the interoperability between the implemented system. In recent years, this technique has been adopted with success in the development of various complex systems (military, space, automotive, etc.). Consequently, we have witnessed the emergence of several ad-hoc customizations, depending on the application domain, processes and business methods applied. This paper specifically focuses on the OMG MDA incarnation [16], which defines the guidelines for the implementation of software-intensive systems through the use of models. MDA must be intended as a family of technology standards, through which it provides higher levels of automation and better interoperability among the various systems technologies adopted.
The typical "heterogeneity" of the technology stacks is dealt with introducing the concepts of metamodeling and model transformation, as illustrated in the 3-layer architecture of Figure 1.
Metamodeling is an architectural abstraction that provides the foundations for construction, manipulation and validation of models.
Model transformation is related to a fundamental task associated with the productive and automated use of models. In the modeling stage, the designer has the task of building a so-called PIM (platform independent model), which is the representation of the application domain in a technology-neutral way. Subsequently, the PIM is transformed into one or more PSMs (platform specific models), which are models that contain information specific to a given technology stack, and eventually into the implementation code of the software components.

3.5 Languages dimension

In this section, we introduce the languages that support the *Tm* dimension. UML is the premier language at the basis of the MDA metamodeling architecture. *UML* is a general purpose language that allows the construction of hardware models and software for the representation of applications and systems of different nature. UML is the base language that has been extended by use of the *MOF-based profiling* mechanism to provide *SysML*, a lightweight extension that has rapidly become the standard modeling language in the SE domain.

As mentioned in the previous sections, a core concept of MDSE is provided by the automated transformation of models. *QVT* (*Query/View/Transformation*) is the MDA standard that deals with the specification of model transformations [14]. Queries define the procedures to traverse a model and extract parts of it. Views are instances derived from the models, while transformations deals with the automated generation of models by use of appropriate mappings that can be specified through declarative (i.e., QVT/Relations) or imperative (QVT/Operational Mappings) languages.

Finally, XMI (XML Metadata Interchange) [13] and OCL 2.0 (Object Constraint Language) [17] are side standards commonly used in a MDSE effort.

XMI is used for defining, exchanging and integrating processing data objects in XML format. XMI specifies an open information exchange model that enables the exchange of object oriented data structures in a standard XML-based way, enabling interoperability between applications.

OCL is a formal specification language that allows the definition of constraints over model elements through a syntax based on the first-order predicate calculus. Through this mechanism it is possible to describe assertions that bind the values of attributes, as well as define preconditions and post-conditions that must be satisfied by modeling elements.

4. MDSE example application

This section illustrates an example application of the proposed MDSE methodological template to the development of a software-intensive system in the naval electronic warfare domain.

The example makes use of the SE *life cycle model* as process and the *SE method model* as method. The main effort has focused on the *concept definition* phase, by iterating the following activities: requirements analysis, functional analysis, physical allocation and testing/validation. All such activities can be traced back to a model-based approach, which, although highly structured, is managed manually by the system designer. Furthermore, the concept definition phase also introduce the transition from the system design view to the related software-oriented design view. Consequently, it is necessary to build models that emphasize the software application building. These issues were dealt with by using a model driven approach. Considering a SysML model that represents the system view, the systems engineer and the software design specialist agree on a set of transformation rules from SysML to UML. These rules are properly encoded in the QVT language, allowing the automatic generation of an UML model, which represents a complete baseline to proceed with the engineering activities.

3.1 MDSE: Template Instantiation

The first step is the instantiation of the MDSE methodological template, introduced in the previous sections. The <*Em, Pm, Mtm, Tm, Lm*> quintuple has been instantiated as follows:

- **Environment (*Em*):** the example application considers a software-intensive system, which may be associated to the naval electronic warfare domain. This term refers to the use of various weapon systems in order to prevent the enemy attack operations against its own marine equipment. Using the hierarchical model, the following three architectural levels have been identified:
    - **System (L1)**: *CMS (Combat Management System)* and *EWS (Electronic Warfare System)*
    - **Sub-System (L2)**: *EWS* is divided in:
        - *RESM (Radar Emitter Support Measure)*
        - *RECM (Radar Emitter Counter Measure)*
        - *DLS (Decoy Launch System)*
        - *EWMU (Electronic Warfare Management Unit)*
        - *EWMU proxies*
    - **Component (L3)**: *EWMU* related software elements.
- **Process (*Pm*):** as aforementioned the example application focuses on the *concept definition* phase related to the SE life cycle model.
- **Method (*Mm*):** the method is a mix of the OOSEM and Telelogic Harmony methods, which are used to drive the transitions between activities.
- **Technique (*Pm*):** The technical model *Tm* coincides with the family of the MDA technological standards discussed earlier. Such standards are fully supported by the tool Topcased Version 5.1.0 [18]. Specifically, SysML/UML models are edited by use of Topcased version 2.0, the editing of profiles is supported by Papyrus UML 0.9 [19] and finally, the specification and execution of model transformations is managed through the QVT-Operational plug-in, version 3.0 [20].
- **Languages (*Lm*):** The choice of the languages addresses the use of SysML [3] and UML [4] to perform modeling tasks and the QVT-Operational Mappings language to support the transformation operations.

3.2 MDSE: Concept Definition

In the concept definition the functional requirements into a model, intended as a baseline for the engineering activities. The following activities have been carried out:

1. *Requirements analysis*: errors have been eliminated and the relative quantification was performed. Finally, we provided the traceability to the operational requirements (system capacity);
2. *Functional definition*: use cases and related execution scenarios have been defined. The features have been identified and allocated to SysML blocks related to the sub-systems;
3. *Physical allocation*: the number of EWMU software components and their allocation has been determined;
4. *Testing and validation*: tests to validate the interfaces between the EWMU and other sub-systems have been planned and carried out.

For the sake of brevity, Figure 2 illustrates the resulting SysML block definition diagram only.

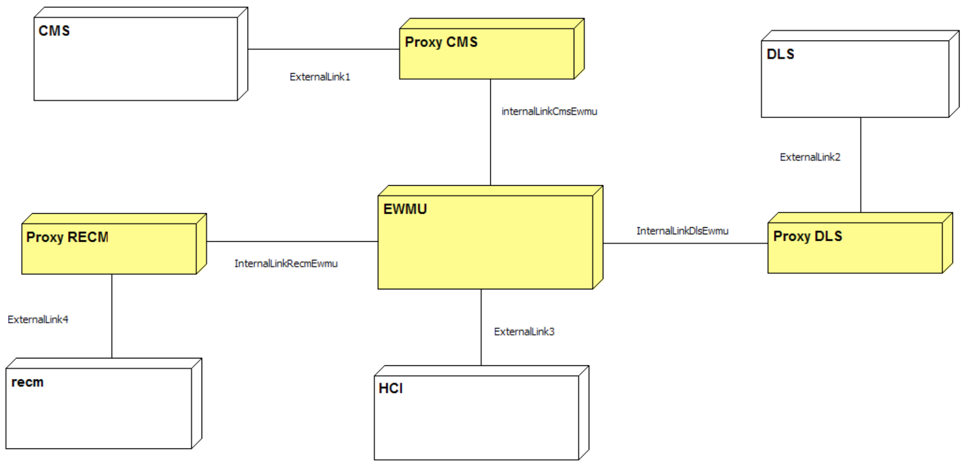

Figure 2. Block Definition Diagram for the example case study

3.3 MDSE: SysML/UML Bridge

The execution of the concept definition phase has highlighted two critical issues that can make the process management more difficult and less efficient, the former being the use of manual modeling activities, which are effort and time consuming, and the latter being the derivation of software models from system models, which can be error-prone and not easily traceable.
The adopted solution consists in the specification of a *SysML/UML bridge* based on an automated PIM-to-PSM transformation coded in the QVT/Operational Mappings language. Although SysML is defined as a UML profile, in this context the UML model is considered platform-specific due to the fact that the transformation refines the SysML model with details related to the software "platform" described by use of a UML model.
As an example, Figure 3 illustrates the specification of a transformation rule that is executed to decouple sub-systems and proxies by use of a *publish/subscribe* paradigm.

```
mapping Block::getComponentOperations() : Set(Operation){

    init{ result := self.ownedOperation.map operation2operation()->asSet()->

    union(self.ownedPort>select(isStereotypeWidth('PublishPort')).map
        getPublisherOperation()->asSet())->
    union(self.ownedPort>select(isStereotypeWidth('SubscribePort')).map
        getSubscriberOperation()->asSet());     }

}
```

Figure 3. Example transformation rule coded in the QVT language

4. Conclusions and future work

The application of the proposed MDSE methodological template has provided various insights that give useful guidelines for future design work. The analysis and the production of a complex system must be supported by tools and mature technologies. Each phase of the development process must be highly structured, so it is necessary to define the documents that must be produced or the tests that are to be planned and executed. Finally, it is essential to adopt solutions that can be widely recognized, from both scientific and business viewpoints. The limitations identified in the example application can be traced back to the adopted methodological template. At process level, the model-driven approach has been applied in two stages, but can also be profitably extended to the engineering step. Furthermore, the object-oriented method alone is not sufficient and also functional analysis and predictive performance analytics should be carried out. Moreover, MDA techniques alone are not sufficient, as they focus their efforts on aspects of design models. In software intensive applications, techniques for model-checking correlated with the validation of the models are frequently used. These aspects suggest interesting challenges that may be faced as future work: the evolution of the template in a methodological framework that can fully support the production activities.

References

[1] DoD Architectural Framework, version 2.02, available from http://dodcio.defense.gov/Portals/0/Documents/DODAF.pdf.

[2] TOGAF Version 9.01, available from http://pubs-opengroup.org/architecture/togaf9-doc/arch.

[3] Object Management Group, System Modeling Language (SYSML), version 1.2, June 2010.

[4] Object Management Group, Unified Modeling Language (UML) Infrastructure, version 2.1.2, November 2007.

[5] Jeff A. Estefan “Survey of Model-Based Systems Engineering (MBSE) Methodologies”, May 25, 2007.

[6] Baker, Loyd, Clemente, Paul, Cohen, Bob, Permenter, Larry, Purves, Byron, and Pete Salmon, “Foundation Concepts for Model Driven System Design,” white paper, INCOSE Model Driven Systems Design Interest Group, International Council on System Engineering, Jul. 15, 2000.

[7] International Council on Systems Engineering (INCOSE), Systems Engineering Vision 2020 INCOSE-TP-2004-004-02 Version 2.03, September 2007.

[8] Martin, James N., Systems Engineering Guidebook: A Process for Developing Systems and Products, CRC Press, Inc.: Boca Raton, FL, 1996.

[9] Bloomberg, Jason and Ronald Schmelzer, Service Orient or Be Doomed!, John Wiley & Sons: Hoboken, New Jersey, 2006.

[10] MLT-STD-498 Application and Reference Guidebook, 31 January 1996.

[11] ANSI/EIA 632 Standard, Process for Engineering a System, 7 January 1999.

[12] Object Management Group, Meta Object Facility (MOF) Core Specification, version 2.4.1, June 2013.

[13] Object Management Group, XML Metadata Interchange (XMI) Specification, version 2.4.2, April 2014.

[14] Object Management Group, Meta Object Facility (MOF) 2.0 Query / View / Transformation Specification, version 1.0, April 2008.

[15] A. Kossiakoff, W.N. Sweet, S.J. Seymour and S.M. Biemer. System Engineering Principles and Practise, 2end Edition. Wiley – 2011.

[16] Object Management Group, MDA Guide, version 1.0.3, 2003.

[17] Object Management Group, Object Constraint Language, version 2.4, 2014.

[18] Topcased version 5.1.0 , available from http://www.topcased.org.

[19] Papyrus version 0.9, available from http://www.eclipse.org/modeling/mdt/downloads/project/papyrus.
[20] QVT Operational Plug-in version 3.1.0 , available from http://projects.eclipse.org/projects/modeling.mmt.qvt-oml.