SOME KEY PRINCIPLES FOR CREATING GOOD VISUAL MODELS

Anthony Spiteri Staines
Department of Computer Information Systems
University of Malta
MSIDA MSD 2080
MALTA
toni_staines@yahoo.com

Abstract: - This paper discusses the importance of key principles for creating good visual models. Sections I to II discuss software and system development, different types of techniques and modeling approaches are used. Visual modeling is suited to developing systems because these notations are understandable by different stakeholders. Visual modeling can make use of graphs. The paper is presented as follows.
1. Introduction: Information and software systems require the use of methods and notations for proper representation. Unfortunately many users do not appreciate the need for creating suitable and aesthetically good diagrammatic notations 2. Background: Different modeling notations can be used to model systems. The key principles behind these notations are based on principles of tidiness, neatness, constructability and the level of detail. Different works and findings are presented. 3. Problem Formulation/ Problem Statement: This section presents various problems with visual modeling. In reality the use of good principles for creating the models are not necessarily identified, considered and adhered to when diagrammatic notations are used. 4. Proposed Solutions: several key principles are used as solutions. These are : i) abstraction, ii) universality, iii) aesthetics, iv) correct sequence and v) patterns. Their importance is explained and it is indicated how these can improve and solve the overall diagrammatic modeling approach. 5. Some Toy Examples: This part shows the ideas presented in the proposed solutions being applied in practice. Models can have several characteristics and still be useful and offer good representation. 6. Discussion and Existing Problems: Explains the validity of the toy examples and how this can be extended to other models. However several problems still remain and these are not straightforward to solve. These issues are explained in this part. 7. Conclusion: Summarizes the paper and explains other issues that can be tackled in the future.

Key-Words: -Diagrammatic notations, Software engineering, Requirements engineering, Visual modeling.

1 Introduction

Information systems analysis, software, hardware design and computer engineering make use of many different forms of system representation. Many formal methods, notations and design methods have been created. Some examples are: the UML, software design methods and techniques. Several graph based diagrams are used to do this job [1]-[10], [12]-[15].

Extensive literature and courses have been developed to support this knowledge [1]-[7]. The design prior to coding principle is a mainstream approach in software development. Modeling is not only used for good system design but can be applied to many other computer related areas. Several reasons exist for modeling. These are: i) understanding a system, ii) simplifying the representation of a system, iii) component identification, iv) scalability, v) representing the architecture, vi) communicating to stakeholders, vii) verification, viii) analysis and design, ix) creating architectural specifications etc. [1]-[3], [12]-[15].

Modeling has become a topic of fundamental importance for the design and analysis of computer systems. It is increasing becoming a problem to select the right model for the job [1],[2]. Some models are never used because of their complexity or oversimplification. Models provide and give insight to different scenarios. However there are cases where models are not used for various reasons.

Models used in computing range from formal models and mathematical models to different forms of visual models [1]-[12].

In this work the focus is on visual modeling. Visual models are easier to understand at a glance and appeal to a wide audience [3], [5]-[7]. This paper is concerned with visual models mainly based on graphs and their representation.

Modeling principles and skills are often left to the user to develop and are not taught as a separate and important topic. Abstraction and representation lie at the very core of system development.

Many existing notations used in literature are modification of block diagram notations and graphs [2],[3],[5]. These are simple and straightforward. Because of their effectiveness they play a crucial role in software modeling. Currently many compositional structures that are used for representation are based on modifications of these models [2],[3],[5].

It is imperative to educate students and IT professionals from the onset about key modeling principles for good system representation [9],[12]. Good notations and designs should support the process of creating better and more robust systems and they support the intuitive creative thinking process which is fundamental in requirements engineering and software engineering [2], [12]-[15]. Creating models also has an artistic and expressive side to it which is very often ignored.

Modeling is symbolic in nature [1], [13]-[14]. In literature there are several approaches that have been created for visual modeling. The mainstream approaches can be shown as follows: the i) UML notation [6], ii) Fundamental modeling concepts (FMCs) [2], iii) Technical architectural modeling (TAM) [15],[20], and different approaches that are graph based, formal or semi-formal [16],[19]. Some graph based approaches could be different classes of Petri nets or plain graphs like those used in [3],[5],[8],[10],[11],[20],[21]. The key principles presented here are an integral part of FMCs and TAM. The FMC approach places a lot of emphasis on aesthetics, node harmonization and layout of the visual models. The same ideas apply for TAM. Unfortunately when using UML and graph notations these principles might not be considered. The emphasis in the UML is to provide a large unified repository of notations for system representation. Other works focus mainly on formal representation, which is a very important aspect. The ideas in the problem solution section related to good diagrammatic notation can be extended to all visual modeling domains.

This paper is organized as follows: 2 Background, here the motivating ideas for creating visual models are presented and compared with other works. 3 Problem formulation, this part presents the problems associated with creating good visual models for software and requirements engineering. 4 Proposed Solutions, identifies some ideas and solutions to the problems presented in 3. Section 5 Some toy examples, illustrates the main ideas behind the solutions proposed in 4. Section 6 Discussion and existing problems explain the validity of the solutions and examples given and consider other open problems that are not easily understood. Section 7 Conclusion closes this paper.

2 Background

In this work modeling can be defined as the activity of creating valuable structures that capture the behaviour or composition of a system. This is an intellectual activity that adds value and involves some sort of effort. Capturing the details of essential structures that either i) exist or are ii) planned will create more knowledge and value about them [1]-[15].

The main requirements of systems must be precisely understood by designers and other specialists [1],[2]. Information and knowledge about a system might have to be exchanged between different persons. Models serve as exchange mechanisms for knowledge that could otherwise go missing. Drawing models and understanding them are two separate activities.

Diversity in notations used for representing modern software development happens because of the different types of requirements and complexity issues. New problems and challenges are being created all the time. Certain modeling structures that are universally accepted and understood are normally used. Class diagrams are an example of this [6], [7]. However new structures might have to be created for a particular scenario.

Requirements of real time systems are very different from those of an e-commerce application. It is impossible to reconcile these because the nature of the end product is entirely different. Each system requires the use of diverse modeling notations and

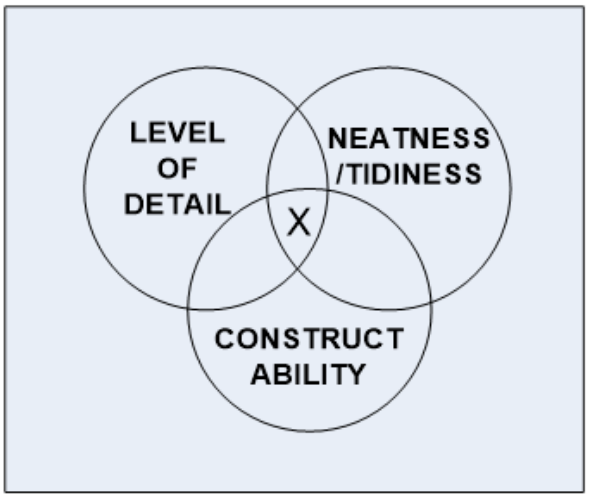

Fig. 1 Key Attributes of System Models

techniques. A delicate balance has to be maintained between using too few or too many models.

Modern systems could be diverse as distributed systems and web-based applications. At the extreme side, embedded applications could be stand alone. Any system can contain a finite number of subsystems depending on classification. Sub-systems can exhibit diversity.

A single simple system can have multiple configurations excluding those that not necessarily accounted for. Diversity, configurability, distributedness, new technologies and platforms etc. require proper management.

Creating good software does not only imply successful coding but involves several other structures and sound modeling principles [2]. These need to be considered in order to deal with complexity issues. Modeling issues tend to be neglected and are not properly understood by many stakeholders in development. This is obvious when good programs that are created require to be developed again from scratch because some important issues were not implemented. Even in modern technologies it is possible to get some high end device that has a poor UI layout or software that is too tedious to operate or gives unforeseen errors in some state that was unaccounted for.

Good models should promote reuse. It should be descriptive, suggestive and self-explanatory. For comprehension it should contain a sufficient level of detail. Tidiness implies that it is neatly laid out and drawn up using appropriate scales. The model would be suitable for visualization.

3 Problem Formulation

Several reasons exist to explain why modeling is neglected: i) there is a lack of uniform modeling foundation, ii) so many diverse notations exist. Which should be chosen? iii) there is a lack of training in requirements engineering and modeling, iv) it is time consuming to apply notations. When should we stop applying them? How much is enough? Which should be selected? v) there is a lack of education and training about the importance of models. For more details refer to [1]-[21].

The skill of modeling is certainly acquired as one goes along solving certain problems and developing systems. It would be beneficial if this skill is systematically taught or initially presented.

In many occasions the use of models is overlooked. There is a lack of instruction at different levels on how to produce suitable models. Analysts and developers are in a rush to get done with the application and solve problems. Thus the result of having a quick implementation comes at the expense of proper comprehension of the problem domain. It might happen than an improper solution has been formulated. A possible reason not to use notations is that they could divulge too much information and be used for reverse engineering by competitors to their advantage.

In many cases detailed diagrams of the system components and their functionalities are not provided for specific obvious reasons.

The lack of proper representation is one reason of the large software costs. This leads to the problem of not doing something right at the first attempt so many other attempts need to follow. The use of good notations would contribute to the solution of this problem.

The following main problems can be identified when constructing system models: i) precise and fundamental terminologies need to be used. ii) the model should be comprehensive enough to reflect about the structures of the system and the subsystem. iii) add value and be understandable by different stakeholders, iv) concise or retain simplicity but not at the cost of omitting important

details. v) be very presentable and simple to construct. vi) the models should promote reuse and transformation to other frameworks or applications [1],[2],[5],[6],[12]-[15].

The main issues presented here can be summarized into the following fundamental points. i) the model is too detailed or contains too little information. If the model contains too much information for the user it becomes unreadable and some of the information is not taken in by the user. On the other hand if the model is too compact, then there is insufficient information and knowledge that can be derived from the model.

ii) the model is not neatly drawn. This is a big problem with many mainstream modeling approaches [5]. The models could contain overlapping edges and nodes. The size of the nodes and the node arraignment setup give the model a look of untidiness. It could also imply that excessive nodes and edges are used [5].
iii) the model should be easy to construct. This implies that there should not be a high level of difficulty to construct the model.
These three main points summarize many of the problems with modeling systems and software. These are depicted in figure 1.

4 Proposed Solution

The following solutions are given for the problems that have been previously identified.

For i) too much detail, the solution would be to simplify the structures and on the other hand for ii) too little detail more information could be added.

The solutions cannot be implemented without the following qualities. The users need to comprehend the value of modeling: i) the user needs to have some idea or a rough idea of how the system works and its possible main components. ii) the user needs to have the ability to formulate or give shape to different classes of problems. I.e. have some explicit form of representation to put the concept into visible form. iii) knowledge of basic computer related structures is a pre-requisite and iv) one has to identify alternative structures or ways of representing.

The key principles presented can be used to solve many of the problems with visual modeling. These form an integral part of creating good visual structures for system and software development. Some of these principles have been already presented to some extent in the MDA, MDE literature, FMCs and TAM and the UML [6],[7],[15],[2]. In this work more details have been added. These have been identified and put together from experience in this area over a long period of time.

These key principles are:

i) Abstraction: this is the ability to describe conceptual architectural structures at different levels. Abstraction implies simplicity. Simplicity happens because the representation is decomposed. Description techniques should be restricted to very few elements and notations.

ii) Universality: a description albeit being simple requires to provide sufficient information that will cover a variety of situations and scenarios. Basic block diagram notations and certain graphs like undirected graphs or digraphs can be used to create useful models that are visually understandable. Many of the diagrams used in the UML like activity diagrams, class diagrams, sequence diagrams are based on graphs. Petri nets and FMC or TAM diagrams are similarly based on graphs [2],[3],[8],[10],[11],[15],[20],[21].

iii) Separation of concerns: there are important and trivial parts. Different parts of the system have different requirements hence the different parts (e.g. components or entities that collaborate together are fundamentally different).

iv) Aesthetics: Deals with the neatness or tidiness of the diagram. This implies that a structure should support a proper layout and presentation. The graphical patterns should be well formed and follow certain key principles that make them presentable. The aesthetical value of the diagram is often overlooked or ignored. The principle of aesthetical value can be extended even to formal notations or representation that are non-graphical. The actual process of creating structures for computer based systems can be attractive not just from the scientific and economical point of view but also from the compositional point of view. Thus it is possible to acquire a skill where designing becomes a sort of craft or art. Aesthetical design would imply the correct layout of nodes, and harmonization of the layout. This will imply proper graphical connections

and tidiness in the design. However these principles are not easily defined and it is important to be tied down to a particular approach.

v) Correct Syntax: This property deals with the correctness and validity of the model being created. Sometimes the syntax used is full of errors. The tool used to create the model could use some form of syntax that is not the proper one. This is commonly seen with the UML where different case tools support different syntax. However the model can normally be supported using formal methods to give it more rigor if this is required. Correct syntax is not necessary in the most important part of the model. Sometimes the aesthetical value of the model is more important.

vi) Patterns: A pattern refers to something that can be repeated. In the case of modeling the patterns refer to the building style properties of the model. E.g. if a form of representation is similar, in this case the pattern will also be similar. In the case of modeling having a pattern implies that for a particular representation in a diagram a similar layout or form is used. E.g. if a decision node is being modeled twice it should look similar.

5 Some Toy Examples

A simple control flow graph (CFG) is used to illustrate the ideas presented in the proposed solutions. The CFG represents the basic processing of a typical bank ATM (automated teller machine). This has been selected because the ideas can be easily understood. The processes or activities in the CFG are just used to explain the control flow and do not necessarily reflect correct ordering. This CFG can be refined and subgraphs can be added. There are several ways and many different notations used in software engineering that are used to represent CFGs.

The example of the CFG drawn using the principles presented here are constructed in fig. 3 and 4. These show exactly the same thing even though they look different. It is possible to find many other valid combinations for representing this example.

The most important principles presented in the previous section are used to create these models. The focus is on generating models that are usable and understandable. As regards aesthetics some might prefer the diagram in fig. 4 to that in fig. 3. For others the opposite might hold. In fig. 4 there is more abstraction and separation of concerns than in fig. 3. because the ATM menu has been properly separated from the main functioning of the ATM.

6 Discussion and Existing Problems

In this section some key principles that are useful for creating good models are indicated. These would help software engineers to come up with better designs. The principles presented can be easily implemented with minimum effort. Ideally, they should be a crucial part of software engineering. However this is not normally the case.

In section V, CFGs have been used as an example but the key points presented here can be used with any type of diagrammatic and graphical notation like those found in MDA [7], UML [6], data flow diagrams DFDs, class diagrams [6], block diagram notations [2], etc. The key principles can be used elsewhere.

The principle of aesthetics is not often considered. This concept could create a new field of modeling notations and software development models. Other principles like universality, syntax, patterns and abstraction are closely linked with aesthetics.

The suitability and usefulness of models is not always in agreement with good construction principles. This is because for good construction principles the following apply i) representation and ii) understanding. Representation does not necessarily guarantee that a model is easy or simple to translate into a given specific language. Why? It could be that the model cannot be translated automatically but requires additional information or additional human information.

System size and complexity are very difficult to estimate and describe. It is fundamental to know at what stage a model will be used. Will the model be used at the analysis stage, design stage, for system documentation or at any other stage?

A model to be used for input to a tool or another transformation needs to be complete and free from inconsistencies. However, there are several secondary issues for it to be communicable and usable at the stakeholder level. When teaching

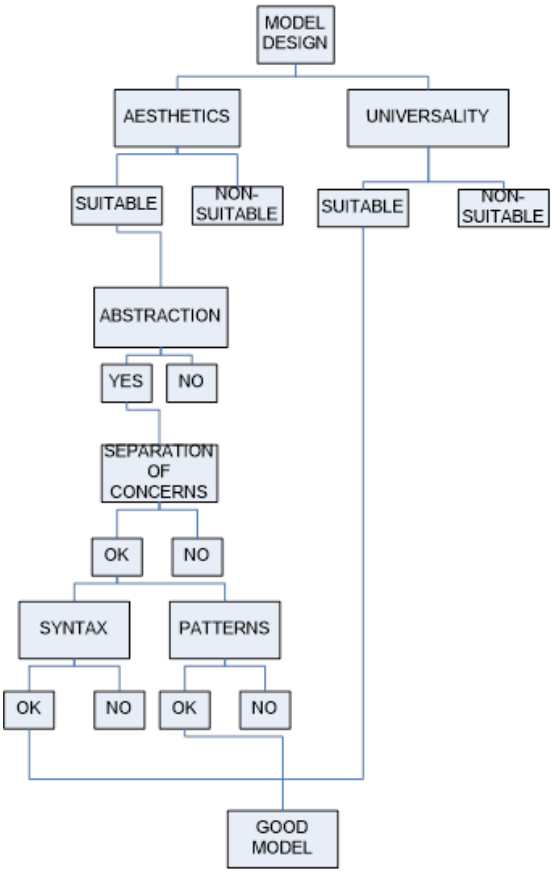

Fig. 2 Static vs Dynamic Composition

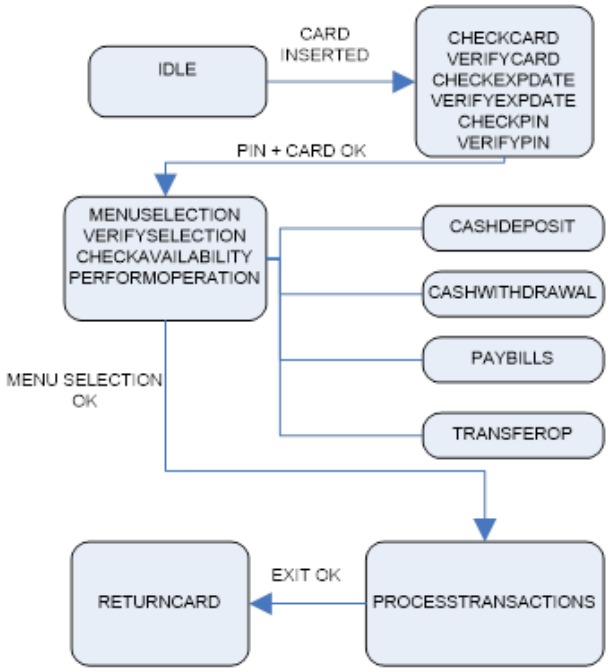

Fig. 3 Control Flow Graph 1 for a Bank ATM

software modeling in topics like requirements engineering and software engineering the layout and visual impact of the model might be of greater significance than its actual completeness. This is because the functionality of the model should serve its purpose as an agent of i) representation ii) communication and iii) pattern capturing. Ambiguity will not allow proper comprehension.

Unfortunately if MDA [7] and UML [6] driven approaches are considered, architectural models can become surprisingly complex and unmanageable. The UML representation can become too rigid. Abstraction does serve to reduce complexity however detail can be lost. It cannot be expected for MDA and UML along with other mainstream approaches to fully replace the need for creating simple system diagrams using block diagram notations. These serve to enable better thinking and give a holistic picture of the system.

It is observed that there does not exist a single notation that covers all the modeling needs for a system. There is a great amount of fragmentation and diversity in requirements engineering thus complicating issues related to its presentation. This happens because different stakeholders require different viewpoints.

Visual models created to represent information systems can be developed to create new forms of representation and understanding. The visual models can be combined with pictorial representation, other notations and mathematical expressions as required.

The expression of form through geometry, graphs, interacting shapes and symbols provides one level of the expression of knowledge and understanding, depending on the transformations carried out.

Graph diagrams and geometric shapes can undergo several processes of dissolution, expansion or contraction of their components. Graph drawing for visualization is a vast area in its own right. There are various algorithms that can be used to optimize the layout of the shape and the particular drawing. Several different layouts can be found to depict the shape. The concept of aesthetics can be expanded to implement ideas related to symmetry, size, vertices layering etc.

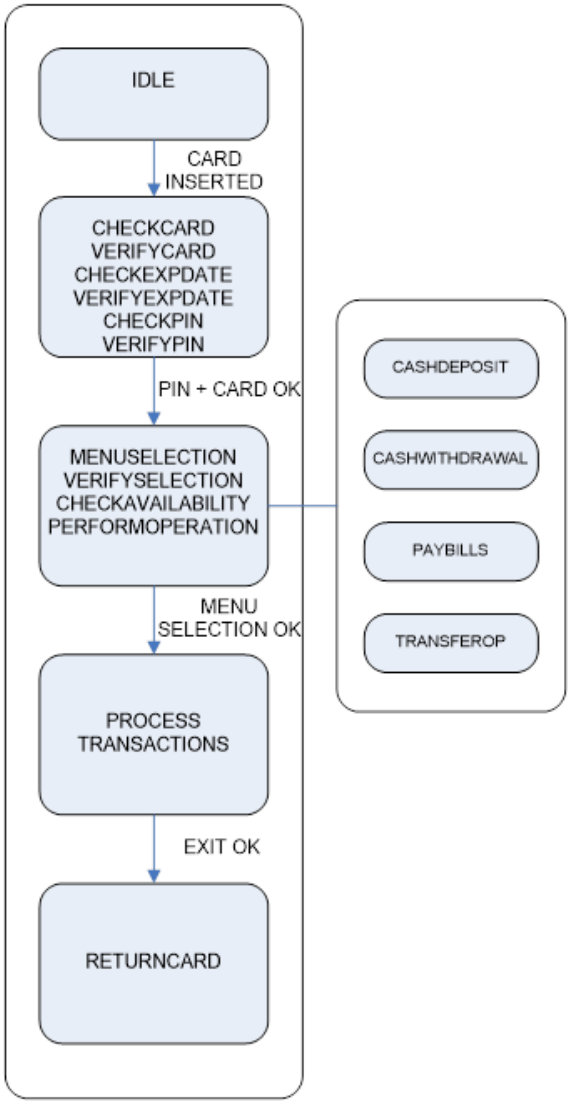

Fig. 4 Alternative Depiction of Fig. 3

Examining a diagram its connections and shapes immediately indicates how its components do relate to each other. A user can discuss various relationships of patterns, connections, layout, form and aesthetics. Liking or disliking the object model is to a large extent dependent on how we perceive the model as a whole.

This is similar to when we see a well laid out model. We experience various qualities based on the relationships between the entities, the size of the nodes, communication, form, aesthetics, layout etc. In this respect modeling is not just a scientific approach but it also has its artistic part. This part is still not properly understood in software engineering.

Software engineering has a variety of issues related to modeling and quality that remain unresolved till this very day. Ideally one must not become attached to a particular approach or notation. This could imply that under certain circumstances some rules and key principles mentioned might have to be broken in favor of others. This choice is left up to the user to decide what is best with ample room for experimentation and creativity.

7 Conclusion

This work deals with basic key principles for creating good system and software models. However it is far off from finding a solution to this problematic area. Requirements engineering and software engineering have several problems that have withstood the test of time and seem to remain unsolvable. The more complex and fragmented systems become, the greater is the significance and importance of sound modeling principles. As already indicated here, there is no one size fits all solution readily available and this important topic definitely requires more work in many different directions. Formal methods and formal notations could be combined with the visual diagrams for creating more robust models. The solutions and ideas presented can be easily integrated in other works and practices to improve the design of visual models.

References:

[1] H. Kaindl And J.M. Carroll, Symbolic Modeling in Practice, *Communications of the ACM*, vol. 42, No 1, 1999, pp. 28-37.

[2] A. Knopfel, B. Grone, P. Tabeling, *Fundamental Modeling Concepts*, Uk: Wiley, 2006.

[3] K. van Hee, *Information Systems: A Formal Approach*, Cambridge Univ. Press, 2009.

[4] C.B. Jones, *Systematic Software Development using VDM*, Pretence Hall, 1990.

[5] G. Di Battista, P. Eades, R. Tamassia, I.G. Tollis, *Graph Drawing Algorithms for the Visualization of Graphs*, New Jersey: Pretence Hall, 1999.

[6] OMG, *UML Super Structure Specification Documentation*, 2018,https://www.omg.org/spec/UML/2.4.1/About-UML/

[7] OMG, *Model Driven Architecture*, 2018, http://www.omg.org/mda/

[8] A. Spiteri Staines, Some Fundamental Properties of Petri Nets, *International Journal of Electronics Communication and Computer Engineering*, IJECCE, vol.4, Issue 3, 2013, pp. 1103-1109.

[9] T. Spiteri Staines, A Rational Perspective on Software Modeling, *Software Engineering and Applications, 9th ICSOFT- EA*, 2014, pp. 345-350.
[10] A. Spiteri Staines, A Triple Graph Grammer (TGG) Approach for Mapping UML 2 Activities into Petri Nets, *9th SEPADS conf., WSEAS* ,Cambridge UK, 2010, pp. 90-95.
[11] A. Spiteri Staines, Rewriting Petri Nets as Directed Graphs, *Int. Journal of Computers, NAUN*, issue 2, vol. 5, 2011,pp. 289-297.
[12] J. Osis, and E. Asnina, Topological Modeling for Model-Driven Domain Analysis and Software Development: Functions and Architectures, *Model-Driven Domain Analysis and Software Development: Architectures and Functions*, 2010, pp. 15-39.
[13] T.D. Kelly, Symbolic and Sub-Symbolic Representations in Computational Models of Human Cognition, *Theory& Psychology, Sage Publications*: Vol. 13, No. 6, 2003, pp. 847-860.
[14] D. Hofstadter, *Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought*, Great Britain: Penguin Books: 1995.
[15] www.fmc-modeling.org, *Standardize Technical Architectural Modeling Conceptual and Design Level*, SAP, http://www.fmc-modeling.org/download/fmc-and-tam/SAP-TAM_Standard.pdf
[16] S. Fleurke, Forecasting Automobile Sales using an Ensemble of Methods, *WSEAS Transactions on Systems*, WSEAS, Vol. 16, 2017, pp. 337- 345.
[17] Agostino Poggi, Developing Scalable Applications with Actors, *WSEAS Transactions on Computers*, WSEAS, Vol. 13, 2014, pp. 660-669.
[18] L. Pace, P. Maggiore, Model-Supported Verification of Space Systems, *WSEAS Transactions on Systems*, WSEAS, Vol. 16, 2017, pp. 64-68.
[19] V. Kasyanov, T. Zolotuhin, A System for Structural Information Visualization Based on Attributed Hierarchical Graphs, *WSEAS Transactions on Computers*, WSEAS, Vol. 16, 2017, pp. 193-201.
[20] SAP, How to communicate Architecture – Technical Architecture Modeling at SAP, 2015, https://blogs.sap.com/2015/02/11/how-to-communicate-architecture-technical-architecture-modeling-at-sap-part-4/
[21] A. Spiteri Staines, Matrix Representations for Ordinary Restricted Place Transition Nets, *WSEAS Transactions on Computers*, WSEAS, Vol 16, 2017, pp. 23-9-29.