OVERVIEW OF ARCHITECTURE FRAMEWORKS AND MODELING LANGUAGES FOR MODEL-BASED SYSTEMS ENGINEERING

Axel Reichwein
axel.reichwein@me.gatech.edu

Christiaan J. J. Paredis
chris.paredis@me.gatech.edu

Model-Based Systems Engineering Center
The G.W. Woodruff School of Mechanical Engineering
Georgia Institute of Technology
Atlanta, Georgia, 30332

ABSTRACT

Model-based systems engineering is a relatively new discipline. As a result, many system designers are not familiar with the main concepts of model-based systems engineering and how it can beneficially impact systems design. The paper presents an overview of architecture frameworks and modeling languages for model-based systems engineering as well as indicates future trends.

INTRODUCTION

While delays and budget overruns may not have been considered that important in the century-long construction of pyramids and cathedrals, they can be ruinous for any construction company in our current free-market society. In general, each contemporary organization involved in the design of complex systems needs to avoid delays and cost overruns in order to stay competitive. Although undesired, they frequently occur when the complexity of a system has been underestimated. Amongst other factors, a system’s complexity depends on its number of components as well as on its number of inter-component dependencies which can be multidisciplinary. These aspects are not specific to a particular project or product but apply for the design of a wide range of systems. Systems engineering emerged as a discipline in order to address problems related to the design of complex systems. It was initially used in the late 1940s by the US Department of Defense and has since been used by NASA and other organizations [6]. Its practical use was driven by advances in operational research, mathematical modeling and computer-based simulation [27].

Systems engineering is a broad activity which includes various aspects such as requirements, models, processes, analysis, optimization and decision-making. The use of computer-interpretable models for systems engineering, in contrast to paper documents, is called model-based systems engineering (MBSE). Benefits of model-based systems engineering over textual approaches include a higher level of support for automation and less ambiguous system representations [1]. As with any information format, standardization of system models is essential in order to ensure that different stakeholders from various disciplines can communicate about systems without ambiguity. As most engineers do not have a deep-rooted education in systems engineering, they can easily be confused by the multitude of systems engineering standards, representations and processes.

This paper provides an overview of architectural frameworks and modeling languages for model-based systems engineering. Chapter 2 introduces the basic definition of model-based systems engineering concepts while Chapter 3 presents past surveys of systems engineering standards. The characteristics of architecture frameworks and modeling languages are presented respectively in Chapter 4 and 5. Possible future directions in model-based systems engineering are identified in Chapter 6.

BASIC DEFINITION OF MODEL-BASED SYSTEMS ENGINEERING CONCEPTS

In order to ensure a good understanding of model-based systems engineering, it is appropriate to start with the basic definition of important systems engineering concepts. A general overview of key systems engineering terms can be found in various literature [3, 11, 49] as well as in several standards such as ISO/IEC 15288 [36], IEEE 1471 [30], IEEE 1220 [31] and EIA 632 [51]. As systems engineering is a relatively new discipline, different definitions for key systems engineering terms exist. Furthermore, as systems engineering has a broad scope, it is not surprising that its key concepts are often defined in general ambiguous terms. The following list of
systems engineering definitions is not an attempt to propose a standard nomenclature but to provide definitions that seem widely accepted.

**System**: The term system stems from the Greek “systema” meaning an organized whole. A system can for example include hardware, software, personnel and activities. The general notion is that a system is more than just the sum of its parts and that its functionality can only be achieved by the interplay of its parts. Due to the interconnections between system parts, the improvement of one part may compromise another part as well as the overall system’s purpose. Therefore, a trade-off often has to be performed between the system’s parts in order for the system to fulfill its purpose in the most cost-effective manner.

**System of systems**: In general, a system of systems is viewed as a complex large-scale system which entails interdisciplinary problems [37]. However, this term has caused some controversy since it appears unnecessary and redundant to the general term “system” which can equally be composed of further subsystems and also be of interdisciplinary nature. In addition, the legitimacy of the term “system of systems” is undermined by the fact that similar extended versions such as “system of systems of systems” appear meaningless.

**Systems engineering**: Systems engineering is defined by the International Council on Systems Engineering (INCOSE) as an interdisciplinary approach and means to enable the realization of successful systems. Systems engineering is both a technical and a management process [70]. The technical process is the analytical effort to transform an operational need into a system of proper size and configuration. The management process involves auditing the effort to ensure cost, schedule and technical performance objectives are satisfied. A list of definitions of systems engineering can be found in [6].

**Systems architecting**: While systems engineering involves quantitative assessments based on analytic tools, systems architecting is in contrast associated with qualitative estimations based on practical lessons learned [49, 69]. Systems architecting is performed when neither goals nor means are known with much certainty. While systems engineering is an approach to seek an optimal system design based on clearly defined objectives, systems architecting is a joint exploration of requirements and design. Systems engineering is therefore more regarded as a science while systems architecting is more viewed as an art [49].

**Model**: Models are the primary means of communication between system engineers, clients, builders and users. Models allow the specialization of labor as a system designer does not need to be the system builder. According to IEEE 610.12 [29], a model is “an approximation representation, or idealization of selected aspects of the structure, behavior, operation or other characteristics of a real-world process, concept, or system”. For simplification purposes, models usually suppress detail through abstraction which can be achieved through generalization, deletion and distortion [11].

**Model-based systems engineering (MBSE)**: The use of computer-interpretable models in systems engineering is referred to as model-based systems engineering. According to the INCOSE Systems Engineering Vision 2020 [32], it 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”. Several other acronyms such as model-driven development (MDD), model-driven engineering (MDE) and model-based design (MBD) are synonymous with MBSE. These acronyms are in fact until now more often used in the software engineering community in which MBSE then stands for model-based software engineering.

**View**: A view is according to IEEE 1471 “a representation of a whole system from the perspective of related concerns”. In other words, a view represents a system according to a specific viewpoint such as performance, cost or reliability.

**Viewpoint**: According to IEEE 1471, a viewpoint is “a template, pattern or specification for constructing a view”. A viewpoint usually informally specifies the purpose of a perspective and the concerns that stakeholders wish to address. A view is an instance of a viewpoint. As an example, a specific vehicle performance view would conform to a performance viewpoint.

**System architecture or system model**: A system architecture consists of several system views. It is also called alternatively a system model. According to IEEE 1220, a system architecture is “the composite of the design architectures for products and their lifecycle processes”, whereby a design architecture is defined as “an arrangement of design elements that provides the design solution for a product or a life cycle process”.

**Architecture description languages or system modeling languages**: Notations to represent system architectures are referred to as architecture descriptions, architecture description languages or system modeling languages. They can be semantically rich and ambiguous such as English, or semantically narrow and more formal such as the Rapide architecture description language [47].

**Architecture framework**: An architecture framework defines the structure and minimal required content of architecture descriptions as well as the range of activities that go into creating and using an architecture description. Architectural frameworks incorporate a broad consensus on best practices for system modeling [50]. Most architectural frameworks focus on the content to be captured in system architectures but do not specify a notation to describe system architectures.

**PREVIOUS SURVEYS OF MBSE STANDARDS**

Standards play an important role in systems engineering in order to ensure consistency across projects and organizations. Chang, Perng et al. [7] presented the evolution of systems engineering standards and guidelines, process models, and compliance assessment models from past to present. The overview of systems engineering standards ranged from the first MIL-499 standard in 1969 from the US military to current standards. Although INCOSE is working on harmonizing standards, it still lists several current systems engineering
standards on their webpage. The wide range of available standards has resulted in a framework quagmire [74] which reduces the likelihood of organizations seeking accreditation for a standard and also increases the possibility of quarrels over interpretation. The INCOSE Vision 2020 document [32] states that “currently, the MBSE process and methods are generally practiced in an ad hoc manner and not integrated into the overall systems engineering processes”.

Estefan [15] presented a survey on methodologies for MBSE, whereby a methodology is defined as a collection of processes, methods and tools. The survey refers to IBM Harmony [13], INCOSE Object-Oriented Systems Engineering Method (OOSEM) [23], IBM Rational Unified Process (RUP) [40], Vitech [8], JPL State Analysis [33] and Object Process Methodology (OPM) [12]. In addition, many organizations have internally developed processes and methods as well. Criteria to choose a methodology include its ease of use, its ability to address the range of systems engineering concerns, and the level of tool support [23].

Estefan [15] also presented a good overview of the breadth and depth of leading systems engineering process standards such as ISO/IEC 15288, EIA/ANSI 632 and IEEE 1220. While IEEE 1220 is focused on the development phase, ISO/IEC 15288 for example spans a broader range from the conceptualization phase to the dismantling phase of a system. Maier [48] described the process of system development as the iteration of models having a high level of abstraction to models having a high level of detail by combining several methods, including real-time structured analysis and design, metrics, performance modeling, and quality function deployment.

Boehm presented trends which will influence systems engineering processes over the next two decades [4]. They include “the increasing interaction of software engineering and systems engineering; increased emphasis on users and end value; increased emphasis on systems and software dependability; increasingly rapid change; increasing global connectivity and need for systems to interoperate; increasingly complex systems of systems; increasing needs for COTS, reuse, and legacy systems and software integration; and computational plenty. It also identifies two wild card trends: increasing software autonomy and combinations of biology and computing”. Boehm discusses the influences of these trends on systems engineering processes between now and 2025, and proposes an emerging scalable spiral process model.

According to the INCOSE Vision 2020 document [32], “many companies and organizations that attempt to define their internal processes according to these standards and models are doing so for the first time”. The lack of reports on how these methodologies perform and compare to each other makes the choice for an adequate methodology difficult. In addition, the INCOSE Vision 2020 document [32] mentions that “the formalism of systems engineering processes has also … created a perception of burdensome, heavyweight efforts, leading to unjustified cost and time overheads. The result is that the application of systems engineering in small and medium-scale enterprises, where the majority of engineering is conducted, remains weak”. System engineers who are process cynics refer to Feyerabend and his arguments against the use of method in science [18] which could equally be applicable against the prescription of a particular method for systems design [26]. Although it seems impossible to find a process that allows us to design systems in a perfectly rational way, Parnas finds the description of a rational idealized process useful as it gives designers guidance and helps them in assessing the progress that a project is making [67].

There is a lack of comparisons between system modeling languages. Most literature on systems engineering lists modeling languages without comparing their characteristics [6, 11, 49]. As a result, a system engineer can easily choose an unsuitable system modeling language which will over time not satisfy his systems engineering needs. A detailed comparison of architecture notations, which has for example already been performed for software engineering [52, 73, 78], has not yet been carried through or published for systems engineering. A reason may be that system modeling languages such as SysML [64] are very recent and that not enough feedback has been collected about their usage. Another possible reason for the lack of comparisons between system modeling languages is that the organizations which perform systems engineering are usually from the aerospace and defense industry and in general follow a very restricted information sharing policy. Experience reports on the usage of a system modeling language may therefore be kept within an organization and not shared publicly.

In contrast to architecture descriptions, there are several comparison studies on architecture frameworks. Shekerman for example lists the history, purpose, scope, principles, structure, guidance and compliance of many well-known architecture frameworks [71]. Emery and Hilliard [14] as well as Lankhorst [41] also provide a good overview of architecture frameworks. Leist and Zellner [43] present an evaluation of current architecture frameworks.

Next to the variety of architecture frameworks and system models, there is naturally also an abundance of mathematical models for systems engineering. The tricotyledon theory by Wymore [88] provides a mathematical foundation for discrete time system models as well as for system couplings and homomorphisms and is often cited in the context of systems engineering. Mathematical logic such as propositional or predicate calculus is frequently used to determine the consistency and validity of models [11]. Of economical importance is the mathematical treatment of decision-making in systems engineering [25, 45].

ARCHITECTURE FRAMEWORKS

Architecture frameworks define the structure and content of architecture descriptions and incorporate best practices to establish such descriptions. An architecture framework can therefore be understood as a basic skeleton or foundation based upon which an architecture description can be established. Recently, Nord et al. proposed a structured approach for reviewing architecture descriptions based on frameworks [55] in order to guarantee that stakeholders can understand and use
architecture descriptions in the way they were intended. Architectural frameworks differ in the addressed concerns, the resulting viewpoints and on the methods to create a view based on a viewpoint.

IEEE 1471, also known as ISO/IEC 42010 Recommended practice for architectural description of software-intensive systems [35], is for example a very high-level architecture framework which is domain-independent and does not impose any specific viewpoints [50]. However, it specifically identifies stakeholders and concerns and suggests that viewpoints should be chosen based on stakeholders’ concerns. As the standard is very generic, its main contribution is seen in the definition of architectural framework concepts. There is a common belief among organizations that being IEEE 1471 compliant is equivalent to having a good system architecture [78]. However, being IEEE 1471 compliant does not ensure that the system architecture was expressed in an appropriate notation or that any reasonable methodology was used.

The Department of Defense Architecture Framework (DoDAF) [84] is similar to IEEE 1471 but goes into more detail as to what kind of information needs to be captured in an architecture description. It is the successor of the Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance C4ISR framework [44]. Unlike IEEE 1471, DoDAF identifies specific viewpoints which belong to four categories: operational, systems, technical standards and “all”. It is important to note that some terminology in DoDAF is not consistent with IEEE 1471 as the IEEE 1471 terms “view”, “viewpoint” and “viewpoint set” respectively correspond to the DoDAF terms “product”, “kind of product” and “view” [78]. This is an example of the non-consistent use of terms across different systems engineering standards which can lead to confusion. A list of DoDAF weaknesses can be found in [49] including for example the lack of intraview consistency as well as performance views which are needed for acquisition decisions. DoDAF was first released in 2003. Examples of similar defense related architecture frameworks include the British Ministry of Defence Architecture Framework (MoDAF) [85], the French Atelier de Gestion de l’ArchiTecture des systèmes d'information et de communication (AGATE) [22], and the NATO Architecture Framework (NAF) [54].

An architecture framework which takes into account more business and organizational concerns is for example The Open Group Architecture Framework (TOGAF) [80]. TOGAF encompasses concerns related amongst others to business, application, data and technology. The centerpiece of TOGAF is the architecture development method which is iterative and forms a spiral-based process. Another non-defense related framework is the Reference Model of Open Distributed Processing (RM-ODP) [34] for the description of information systems which are running on independent heterogeneous nodes. RM-ODP defines five viewpoints including enterprise, information, computational, engineering and technology. RM-ODP goes a step further than other frameworks by also specifying the characteristics of the notation to describe these viewpoints. RM-ODP for example prescribes object-based notation concepts such as object, object template and interface [68].

Among other architecture frameworks, a famous enterprise architecture framework which is worth citing is the Zachman framework [89] from 1987 which popularized the notion of different viewpoints. The Zachman framework provides a classification of descriptive system representations in a simple matrix form. Each row of the matrix stands for a view with an increasing level of detail and each column stands for a view from a different perspective based on basic interrogative conjunctions such as why or what? Each cell within the matrix describes a system view at a specific level of detail according to a specific perspective. The matrix has 6 rows and 6 columns so the Zachman framework proposes 36 different system views.

In general, architecture frameworks do not prescribe any specific notation for architecture descriptions. This allows architecture frameworks to be used flexibly with different conforming architecture description languages. In some cases, the selection of a single architecture notation may leave out important details or be cumbersome. On the other hand, the selection of several architecture notations increases the required coordination effort.

An example for bridging the gap between an architecture framework such as IEEE 1471 and an architecture description notation such as UML [63] is presented by Kandé, Crettaz et al. [38]. Another example by Frankel, Harmon et al. [21] presents how the Object Management Group’s (OMG) Model Driven Architecture (MDA) [59] standard notations could be used for the different system views of the Zachman framework.

MODELING LANGUAGES

An architecture description consists of system views whereby each of them consists of a collection of models addressing related concerns. The activity of modeling has costs and benefits. In general, models fulfill several roles including documentation, performance prediction, generation of other artifacts such as software code and last but not least communication among clients, users and builders. However, the creation of models is time-consuming and expensive. In addition, keeping the cross-model consistency is unfortunately mostly an unproductive manual process. Designers therefore have to first decide what needs to be modeled at what level of detail before choosing a specific modeling notation. Different concerns will be modeled at different depths. In general, only the most critical system aspects will need to be modeled in great detail.

As models are abstractions of systems, they focus on capturing some aspects while leaving out others. As a result, three key concepts can be used to characterize models independent of a specific modeling notation: ambiguity, accuracy and precision. A model is accurate if it is correct whereas a model is precise if it is detailed. Ambiguity should ideally be avoided. A model is ambiguous if more than one interpretation is possible. According to the influential and timeless book on software project management “The Mythical
Unfortunately, the box-and-line diagrams that are often drawn on whiteboards are the greatest source of ambiguity. Although not a bad starting point, these diagrams suffer from ambiguity as it is not clear whether the boxes mean classes or objects or something else. Similarly, arrows can mean dependencies, data flows or something else [10]. The use of a standardized modeling notation is helpful in avoiding ambiguity as the meaning of symbols is then already defined. However, the definition of the meaning of symbols may have been specified in English and still be ambiguous. This has for example been observed with some parts of the mature and widespread UML modeling notation [56]. Developers can then waste considerable time resolving disputes over usage and interpretation of notation.

Many concepts for the description of system architectures are identical to concepts for the description of software architectures since both software and system representations are very abstract. An excellent overview of common notation-independent software architecture modeling concepts and styles is presented by Taylor, Medvidovic et al. [78] and Clements, Bachmann et al. [10]. They are similar to the modeling concepts for systems design presented by Oliver, Keliher et al. [58].

System architectures will usually describe both static and dynamic system aspects. Static aspects are usually easier to describe as they do not have to capture changes over time. Models of static system aspects are for example commonly composed of components and connectors. Components are building blocks with an encapsulated functionality while connectors are building blocks that regulate interactions between components [78]. In addition, Clements, Bachmann et al. [10] distinguish between three categories of architecture styles: module, component-connector and allocation. Common models of dynamic system aspects are behavioral models (describing the behavior of a component/connector over time), interaction models (describing the interactions within a set of components/ connectors over time) and data flow models (describing data flows over time).

Common graphical notations to describe single system aspects are for example Function Flow Block Diagrams (FFBD), Data Flow Diagrams (DFD), N2 Charts, IDEF0 diagrams [46]. On the other hand, there are also notations for multiple system aspects which can span several views and play an integrative role by tying multiple views together. These notations are termed modeling languages or architecture description languages. Modeling languages consist of a set of concepts and rules to describe single or multiple system views. A careful reader will notice that there is no one-to-one correspondence between the standardized systems engineering term “view” which can consist of several “models” and the notion of “modeling languages” which in contrast can span several “views” [49]. This is another example of a confusing inconsistency in the systems engineering terminology.

General-purpose modeling languages do not have any domain-specific modeling concepts and can therefore be used to describe a variety of systems. Examples of general-purpose modeling techniques are the use of a natural language such as English or PowerPoint-style modeling. These modeling techniques are however ambiguous and do not enable automatic consistency checks. More formal notations are therefore usually used to describe system architectures. The most well-known general-purpose modeling language is the Unified Modeling Language (UML) [63]. It resulted from the unification of several object-oriented modeling notations. Its main strengths are its large variety of modeling constructs and viewpoints, its extensive tool support and widespread adoption [78]. UML is definitely less ambiguous and more formal than PowerPoint but it does include purposefully some ambiguous modeling constructs such as the open-headed dependency arrow.

Additional semantics can be added to UML elements through stereotypes. This allows the UML to be extended with additional domain-specific semantics in order for example to remove the ambiguity inherent to UML or to support the automatic transformation of UML models into other domain-specific formats. UML extensions related to the same domain are grouped in a UML profile. An important extension of the UML is the Systems Modeling Language (SysML) [64] for systems engineering. SysML adds stereotypes to the UML as well as new diagram types to capture requirements and parametric relations while leaving out some UML diagrams such as component and object diagrams. Another example of a UML extension is the Unified Profile for DoDAF/MoDAF (UPDM) [61] to use the UML notation within the DoDAF/MoDAF architecture frameworks. SysML can, just as UML, also be extended. A recent example is the SysML4Modelica profile for the bidirectional transformation between SysML and Modelica models [65].

In contrast to general-purpose modeling languages, there are numerous domain-specific modeling languages which are specially adapted to the description of systems belonging to a specific domain. The modeling constructs are then domain-specific and not generic as in general-purpose modeling languages. This facilitates the modeling of domain knowledge and communication among domain experts as well as design synthesis [39]. Examples of domain-specific modeling languages include Koala [86] or AADL [17] for software design. Domain-specific modeling languages are usually defined through metamodels. Several workbenches are available for the creation of domain-specific modeling languages such as the Eclipse Modeling Framework [76] or MetaEdit [82]. Domain-specific modeling languages can be extended in a similar way as general-purpose modeling languages. Although domain-specific modeling is gaining momentum in software design, not many examples can be cited for the design of complex multidisciplinary systems.

**FUTURE DIRECTIONS**

Model-based systems engineering is still a relatively young discipline so its practice can be improved in many ways. Taylor
and van der Hoek [79] presented a list of future directions in software engineering which will most likely also apply for systems engineering as software and systems representations are very similar (e.g. UML and SysML). Another list of desiderata for architecture description languages can be found in Shaw and Garlan [73]. Next to improving the maturity of modeling languages as well as best practices for system modeling and tool support, important future directions in MBSE aim at improving decision-making during the system design process, ensuring consistency between heterogeneous models as well as an efficient synthesis of system architectures and system architecture families.

Before starting to model a system, designers need to consider the cost and benefits of modeling in order to decide which system aspects need to be modeled and at what level of detail. Designers are faced with a similar dilemma in system space exploration when having to choose between the creation and evaluation of many low-fidelity architecture models or only few architecture models with high fidelity [53]. Experienced designers will usually know which critical system aspects have to be modeled at a high level of detail. However, these decisions can also be subjective and not cost-efficient. Frameworks can assist designers in rationally choosing the required level of fidelity in modeling and therefore contribute to large cost savings. These frameworks could for example be inspired by real option theory [77], utility theory [19] or rational design theory [81]. In addition, modeling guidelines to better describe the rationale behind design decisions would provide more transparency to stakeholders and decision-makers [83].

Another challenge is keeping the consistency between heterogeneous architecture models. Models can differ in abstraction level and in domain focus. One approach is considered the “central” model approach [57] and consists of a system model spanning multiple discipline-specific views and ensuring consistency among them. This approach works well if domain-specific models can be mapped easily into a common system model. Although domain-specific models can be very different, they usually share common higher-level modeling constructs to support the creation of reusable components and component libraries. Based on these common modeling constructs, various domain-specific models can be mapped into a common generic system model. Extensions to the UML have for example enabled to create a common system model to ensure consistency between CAD and dynamic system models [24]. More recently, extensions to the SysML have allowed a SysML model to describe simulation models [28], dynamic system models in Modelica [65] and mathematical programming problems [72]. A system modeling language such as SysML might therefore evolve not only as a language to describe systems on a high level of abstraction but also as a language to glue heterogeneous models together.

Another approach to ensure consistency between heterogeneous information sources is based on ontologies. In addition to models which describe a system through a set of modeling constructs, ontologies can formally capture the semantics of modeling constructs. This allows ontology matching techniques to automatically detect links between different ontologies and ensure consistency between them. This approach is currently pursued on a large scale for the Semantic Web [42]. System models could for example be represented as ontologies in the Ontology Web Language (OWL) [87] and take advantage of ontology matching. However, ontology matching is an ongoing research effort [16] and it is not sure whether the promises of the extensively promoted Semantic Web will ever be met.

A different challenge in MBSE consists of efficiently creating a detailed system architecture based on a high-level system description, possibly only containing a set of requirements. The generative process can be performed through model-driven development. This technique consists of capturing design knowledge in model transformations which can then be invoked on initially abstract models to automatically create more detailed models or artifacts such as software code or documentation, thereby ensuring consistency throughout the design process. Model-driven development originated in software engineering in order to create code faster and with fewer errors. The new OMG standard to describe model transformations is Query-View-Transformation (QVT) [60].

France and Rumpe presented the model transformation challenges as well as a research roadmap on model-driven development of complex software [20]. The problems with applying model transformations in software design arise from having to bridge the wide semantic gap between abstract software representations such as UML and software code such as C++. However, this gap is usually narrower in systems engineering since system models as well as many analysis and simulation models share a similar high-level block-oriented notation. Model transformations were recently used in the context of systems engineering for the efficient synthesis of hydraulic circuits [39].

Another challenge in MBSE consists of not only modeling single system architectures but a family of system architectures. The idea is to easily reuse knowledge about an application domain for the creation of similar system architectures related to similar business needs. In general, a reference architecture, also called a base model, is used to represent the system aspects common to all system variants. The effort to create a specific system architecture based on a reference architecture should then be lower than when creating it from scratch. A lot of literature addresses this technique for the design of software families, also called software product lines [9, 66].

Recent developments suggest different approaches to describe the base model, the variations and to resolve both into a specific architecture [2]. An upcoming OMG standard to describe variability aspects independent of a specific reference architecture notation will be the Common Variability Language (CVL) [62]. In general, the efficient creation of a family of systems intends to substantially reduce costs in systems design. However, even a family of architectures has to evolve over time according to new business needs. Changes may then quickly
turn an initially cohesive set of separate base and variation models into an unmanageable set of disjoint models. Reminding of the problems faced in aspect-oriented techniques, and in addition, applying product lines to a broader set of multidisciplinary systems is more challenging than to a narrow set of software products.

**SUMMARY**

Model-based systems engineering is a broad activity. A system designer can therefore easily lose his or her orientation among the jungle of architecture frameworks and modeling languages. The intent of the paper was to provide an overview of major model-based systems engineering concepts in a summarized form. The paper first introduced the basic definition of common systems engineering terms as well as examples of the inconsistent use of some terms. Next, surveys were presented on standards related to processes, methodologies, architecture frameworks and modeling languages. The paper then showed the main characteristics of major architecture frameworks and modeling languages for model-based systems engineering. Finally, the paper indicated future trends in model-based systems engineering, which are all similar to current developments in software engineering.

**REFERENCES**
