Overcoming Mechatronic Design Challenges: the 3+1 SysML-view Model

Kleanthis Thrambouidis, Member, IEEE

Abstract—There has been an increasing interest of the research community in mechatronics over the last years since current practices are unable to address the complexity of today’s mechatronic systems. New methodologies are being proposed to address the challenges in the mechatronics domain. However, the communication gap, which exists between the various disciplines involved in mechatronic systems, makes the task of defining new methodologies very difficult. Moreover, there is no commonly used terminology, which makes the task of comparing or unifying these methodologies hard. In this paper, we refine our approach for synergistic integration in mechatronics and we attempt to establish a basic terminology and framework in this domain. It is claimed that main challenges in mechatronic system development including synergistic integration, size and complexity, reuse, as well as requirements handling and traceability, support for decision making, and maintaining consistency, are successfully addressed by the 3+1 SysML-view model approach. It is argued that the proper integration of Model Integrated Mechatronics (MIM) with SysML, on which the 3+1 SysML-view model is based, is a promising platform for a solid framework for mechatronic systems development.

Index Terms—Mechatronics development process, SysML, Model Driven Engineering, requirements handling, consistency, 3+1 SysML-view model.

I. INTRODUCTION

Mechatronics is the engineering discipline concerned with the construction of systems composed of mechanics, electronics and software. The current practice in the development of these systems is characterized by a subsystem based approach by which integrated systems are built from technology homogeneous subsystems [1]. Moreover, current mechatronics engineering practices have a propensity to develop application-specific controllers, reducing reuse and increasing per-unit cost [2]. In this context, the traditional approach for the development of mechatronic systems, according to which their constituent parts, i.e., mechanics, electronics, and software, are developed independently, and then are integrated to compose the final system, does not address the requirements of the development process of today’s mechatronic systems. As an example, traditional automated manufacturing systems are not capable of responding rapidly to changes in demand and supply and they do not deliver the level of agility that is highly imposed by current trends in the development of goods [3]. This is why manufacturers work to streamline and standardize system decomposition, in order to improve modules and subsystems reuse, furthering these benefits, and reduce per-unit costs [2]. The mechatronic systems developer should be able to evaluate different design alternatives as well as simultaneous changes in the three discipline parts, during the development process. Due to this there has been an increasing interest of the research community in mechatronics over the last years. New methodologies and tools are being proposed to address the current challenges in mechatronic system design. Among these challenges we discriminate synergistic integration, size & complexity, reuse, requirements handling and traceability, support for decision making, and maintaining consistency.

Research groups have already presented various methodologies. However, questions are, even today, more than answers in this domain [4]. Moreover, there is a great gap in the terminology used in the various methodologies. The lack of a commonly used terminology by the various researchers in the area of mechatronics is evident. This is one of the factors that makes it hard to compare, and even more understand, the findings and contributions of the different methodologies proposed by various research groups. In [5] the importance of mechatronic domain models towards the digital plant engineering is discussed. The authors define the term “mechatronic information object” to refer to the collection of the information of the disciplines involved in the mechatronic system over its whole lifecycle. Authors use the mechatronic information object as a representative for the physical mechatronic system and the physical mechatronic component. In [6], authors describe a methodology that has been produced by aggregating existing methodologies in various domains. They use, a) the term “mechatronical unit” to refer to the real world entity, and b) the term “its describing information” to refer to the information that refers to the mechatronical unit derived during the early development phases. Authors in [7] attempt to define the basic concepts on which a mechatronical approach should be based. They use the term “mechatronical information object” (MIO) which they define as “an engineering artifact of information object type that combines the modeling of mechatronical units of a manufacturing system with its different characteristics like signals, electrical drawing, function blocks or devices in one object.” They consider MIO as the information representation of the mechatronical unit in the engineering process.

Kleanthis Thrambouidis is with the Electrical and Computer Engineering Department, University of Patras, Patras, 26500 Greece (e-mail: thrambo@ece.upatras.gr).
In this paper, several research approaches are considered in order to define a solid terminology for mechatronics systems development. Based on this, the 3+1 SysML-view model [8] is discussed regarding its ability to address the aforementioned challenges in the development process of mechatronic systems. This model, which addresses the synergistic integration of the constituent parts of Mechatronic systems, is based on four views of the system. The main view is the SysML-view which corresponds to the mechatronic layer of the Model Integrated Mechatronics (MIM) Architecture [9]. This view captures the system model, which is the one constructed by the Mechatronic System (MTS) developer. Each one of the other three views is used to describe the system from the perspective of the corresponding discipline, i.e., mechanical, electronic and software. Specific tools of every discipline are exploited for the model execution and analysis of the SysML models of the mechatronic system. The 3+1 SysML-view model is extended in [10] to support the development process of safety critical Cyber physical systems. The MTS V-model is adapted to propose an efficient integration of system engineering with safety engineering starting from the early phases of system development.

The Festo-MPS example application is used as a running example throughout the paper. Festo-MPS, which processes cylindrical work pieces, is composed of three units, the distribution unit, the testing unit and the processing unit. Festo-MPS is considered as a composition of mechatronic components in contrast to [11] where the mechatronic system is composed of components from various disciplines.

The remainder of this paper is organized as follows. In Section 2, background and related work is given. Section 3 discusses our proposal for the synergistic integration in mechatronic systems development and briefly presents the basic concept of the Model Integrated Mechatronics paradigm. Section 4 presents a realization of the MIM approach based on the use of the Systems Modeling Language (SysML). Section 5 discusses mechatronic challenges and how they are addressed by the 3+1 SysML-view model approach. Concluding remarks are drawn in the last Section.

II. BACKGROUND AND RELATED WORK

The terms “Mechatronic Object” and “Mechatronic component” have been used in the development process of manufacturing systems also by other researchers, e.g., [12] and [13]. However, in [12] authors mostly focus on the software part of the component; they do not address the whole development process, and they do not provide an architecture for the concurrent engineering of all constituent components, i.e., mechanical, electronic and software. Authors in [13] use the term mechatronic component in the title, but they do not define the meaning of this term, neither do they use the term in the manuscript. Instead, they use the term physical component and mechanical component to refer to mechanical units of the controlled process [14]. The term mechatronic component is also used, e.g., [11], to refer to any constituent component of a mechatronic system. These components are usually monodiscipline.

The term Mechatronic Module is used in [15] as a construct that utilizes more than one different domains (disciplines) of mechatronics, merging the respective domain-specific components. According to the authors a mechatronic module designates the “smallest” indivisible mechatronic subsystem within the set of mechatronic sub-systems since this is the only construct of a mechatronic system that can be decomposed into domain-specific, i.e., non-mechatronic components. This means that a mechatronic module cannot be decomposed into other mechatronic modules or mechatronic system components.

The term mechatronic information object (MIO) is used in [5] to refer to the construct that gathers the information from all disciplines that describe a mechatronic system or component. Authors in the same paper use it as the representative of the physical mechatronic system or component in information technology. They consider it as an information container for all the engineering phases: it includes the design specification but it also gathers information regarding the maintenance of the physical mechatronic object. This definition raises several questions since a component represents during the different phases of the development time the same thing but in different levels of abstraction. Therefore, the real world mechatronic object is an implementation model of what the corresponding mechatronic object represents at the design time. The term MIO is also used in [7]. It is defined as the information representation of a mechatronical unit within a mechatronical engineering process. A mechatronical unit is defined as a mechatronical system that can be described by its functionality and is composed of software and hardware objects. Authors in [16] discuss the shortcomings of current practices, methods and tools in order to identify the most important challenges. They also propose potential solutions and directions where further work is required. From the above, it is evident that a solid terminology that would allow the convergence of the various methodologies is missing.

III. THE INTEGRATION PROBLEM IN MECHATRONIC SYSTEMS DEVELOPMENT

A. Concurrent Engineering: One more step forward

The software community confronted many years ago the challenge of designing algorithms and data structures. There were several approaches that focused on each one domain separately and independent of the other. Some approaches defined first the data structures and then the algorithm, while others followed the reverse direction. Today it is widely accepted that in the development of software systems, algorithms should be designed concurrently with the corresponding data structures they use. If the corresponding data is not properly organized, the results are not optimal even with the best algorithm. Concurrent engineering of data structures and algorithms is the prerequisite for the optimal solution.

A similar problem was confronted a few years ago in the embedded systems domain, regarding the design of software and hardware. It was found and it is now widely accepted that
co-design, i.e., the concurrent design of hardware and software, is the approach that leads to the optimal solution.

We believe that this model will be adopted very soon in the development process of mechatronic systems and it will also be extended to include the mechanical part. The current approach is based on the initial partitioning of the mechatronic system in three parts, one for each discipline. Moreover, the software development starts when the development of mechanics and electronics has already been completed. This results in very strong design constraints in the software development, which in many cases excludes the optimal solutions from the development process. Fig. 1 represents very well this situation even though this picture was originally used fifteen years ago in [17] to highlight the software crisis. One should consider SYSTEMS, in Fig. 1, to represent the mechanics and electronics of the mechatronic system, and SOFTWARE to represent the software part that has to fill the rhombus whole. Electronic and mechanical solutions are represented in models during the software architecture definition and are used during the software system analysis through simulation, but they are outside of the system context. Therefore, modeling and simulation are used to perform trade-off analysis only between various design alternatives in the software domain. Examples include among others functional requirements allocation and non-functional requirements apportionment, e.g., control-loop performance.

![The Software Crisis](image)

Fig. 1. Software Engineers have to fix all the hardware architecture problems. Problems that should have been solved in the mechanical and electronics domains are solved in the software domain.

As claimed in [8] “there is no joint development process, no joint tool usage, no joint modeling formalism and no joint analysis.” One reason for this is the distinction of the development process into various stages or phases, including requirements analysis, functional analysis, design synthesis, and system analysis, which was imposed by the corresponding process models created during the early steps of software and system engineering. The other reason is that the joint tool usage of the various phases has not been addressed successfully. The concurrent engineering of the three discipline parts at the system level, (see Fig. 2a), is proposed as a solution to this problem that will lead to mechatronic designs that optimally satisfy functional and non-functional requirements. Moreover, the partitioning in the three discipline parts, which is used to address complexity in each discipline, should be synchronized with the partitioning on the other disciplines. Authors in [2] claim that this synchronization will improve time to market and system quality and will reduce warranty costs. However, this concurrent engineering of the three discipline parts at the system level introduces more complexity in the development process. To address this complexity MIM has introduced the concept of Mechatronic Component (MTC) [9]. MIM adopts another type of partitioning as shown in Fig. 2b. The MTC is used as a new construct aiming to partition the system. Based on the concept of MTC, the MTS is considered as a composition of already existing MTCs that properly collaborate to offer the required by the system behavior. An MTC is defined as a reusable element of mechatronic systems with well defined interactions with its environment. It has an implementation of its provided interfaces and their Quality of Services (QoSs), based on the proper synergistic integration of mechanics, electronics and software. Therefore, an MTC is a self-contained part of a mechatronic system with specific responsibilities, functional and non-functional, which are implemented adopting a synergistic integration of mechanics, electronics and software. This means that the MTC fully encapsulates mechanics, electronics and software required for the component to offer its required services. The MTC provides the unit of reusability in the development of MTSs. Concurrent engineering regarding the three disciplines is applied at the level of primitive MTC, as shown in Fig. 2b. The Mechanical part is concurrently designed with the electronic and software ones of the MTC, leading to optimal designs that satisfy the required responsibilities, functional and non-functional. This trend can be expressed by the following claim “We need to abandon the traditional mono-disciplinary development process to avoid a mechanical solution when an electrical or software solution is better. But also in opposite direction: avoid complex software by utilizing smart mechanical or electrical (re-) designs.”

![Partitioning and concurrent engineering](image)

Fig. 2. Partitioning and concurrent engineering in mechatronic systems development: (a) Discipline partitioning at system level, (b) Discipline partitioning at primitive component level.

B. Model Integrated Mechatronics (MIM)

MIM [9] is a paradigm that was proposed to address the need for integrated development in mechatronic systems. MIM provides a framework for the model-driven development of MTSs through the evolution of models on the mechatronic layer, that is a new layer of abstraction introduced by MIM. The mechatronic layer, was defined to systematically address complexities in the model-driven development process of
MTSs. Key concept of the mechatronic layer is the MTC, which is the primary construct for defining MTSs on the mechatronic layer. MTS integrators work horizontally in the model evolution dimension of the MIM architecture on the mechatronic layer. They interactively compose the MTS using already defined MTCs without worrying on lower layers’ implementation details. MTS integrators go through a model-driven development process to build the MTS using descriptions of already existing MTCs. They mainly have to capture the application logic in application layer components, as well as to identify their required QoS characteristics from the resource layer infrastructure. A special kind of MTC, the integrating MTC, may be used for the integration of existing MTCs.

The application layer of MIM is used to model the controlling application software. The processing and communication resources required for the execution and collaboration of the controlling application’s software components constitute the resource layer of the MIM architecture. In this layer, the services provided by computing and communication resources are also modeled. Physical devices such as field devices, field buses, and interworking units, as well as logical devices such as virtual field buses, virtual field devices and logical paths for their interconnections, belong to the implementation space of the resource layer. Abstract software representations of these artifacts belong to the design space of this layer and play, in the developer’s workspace, the role of proxies of the actual real-world objects. They can be used during the deployment process to support the realization mapping of the application layer components to the resource layer ones.

The MTC appears in MIM as key construct from the early phases of the development process, which may be either domain analysis or the early design phase of the specific system. This MTC analysis construct is refined through the evolution of models up to the implementation one. Therefore, an MTC may be identified during product line analysis or domain analysis, but it can also be identified during the architectural design of a specific system.

As claimed in [4], at the lowest or device level of a MTS, the emphasis is implicitly on achieving specific levels of functionality and performance. However, as the author claims, progressing upwards to the aggregation hierarchy, the emphasis shifts from the performance of the individual devices, to the integration of a number of such devices as part of a larger system. As a result there is a move away from the detailed design and operation of these devices to the management of the information infrastructures required to achieve overall system functionality and performance. This is not valid for MIM, where at the lowest level, that of the primitive MTC, the emphasis is on concurrent engineering of the three disciplines to get the optimal solution that satisfies the required responsibilities and their QoSs. Functional and non-functional requirements are of primary concern to the developer. However, in contrast to [4] functional and non-functional requirements are of primary concern also for the developer of composite MTCs. This is also the case up to the MTS level where the focus is on system engineering and not only on management of the information infrastructures required to achieve overall system functionality and performance. The author in [4] brings up the following relevant question: “is it perhaps therefore the case that the lower levels of the hierarchy are those where the mechatronics approach has had a major impact, while at the higher levels it is systems integration, and hence systems engineering, that is more important?” Regarding this question, MIM defines the mechatronic layer to focus on system integration without worrying about the integration of constituent components or about their implementation. All the MTC details are handled in the vertical model integration dimension by the MTC builder.

IV. AN IMPLEMENTATION OF MIM: THE 3+1 SYSML-VIEW MODEL

A. The framework

For the development of a mechatronic system, engineers from different disciplines work on their view of the system. Fig. 3, represents the three views of a mechatronic system, using as example the feeder of the Festo MPS. Each view is used to describe the system from the perspective of the corresponding discipline. Mechanical engineers work on the mechanical view (m-view) of the system that corresponds to the Mechanical layer of the MIM architecture and captures all the mechanics, hydraulics and pneumatics of the system. Electronic engineers work on the e-view. The run time platform of the software part is also included in this view. Software engineers work on the software view (s-view). This view provides the models of the software part of the system and allows for software specific tools to be used to elaborate and further refine these models. However, there are several cross cutting aspects of the system that vertically cross the three disciplines. In Fig. 3, three of these cross cutting aspect are shown, i.e., functionality, safety and security.

![Fig. 3. The three discipline views of the smart Feeder of Festo MPS. Cross cutting aspects, such as functionality, safety and security, vertically cross the three views and increase the complexity of the MTS development process.](image-url)
the integration of the tools used in the development process is a very complicated task. The integration problem is becoming more complicated if we consider the cross cutting aspects of the system. Fig. 4 presents the Safety engineer that is using safety engineering tools to perform safety analysis. These tools should have access to the models of the different disciplines which makes the tool integration problem more complicated. Moreover, several questions arise on this model of tool integration. For example: “what about the format of the model repositories?”, “how to manage the need for a tool change in one discipline?”, and so on. It has been recognized that one of the most important features of mechatronics design tools will be the interchangeability of models between design tools from different domains [18].

The National Institute of Standards has estimated that data incompatibility is a 90 billion dollar problem for the manufacturing industry. This was the motivation for the Application Protocol 233 (AP233) [19]. AP233, formally known as ISO 10303-233:Systems engineering, is the STEP Systems Engineering Project, that is one of the several projects that focus on specific exchange and integration problems within various domains. It defines a data exchange format which provides a reliable way to move data between software tools as well as a system-independent format for archiving data. Its main goal is to address the interoperability problem that exists between the systems which are used to manage technical product data in the design and manufacturing of systems. However, if we look at the supporting tools, market support and the standardization process, the status of AP233 is discouraging. There is only one tool that supports it, the market support is limited and the standardization process is too slow.

![Fig. 4. The tool integration problem in the mechatronics system development process is further complicated when cross cutting aspects, such as Safety, are considered. Safety engineering tools should have access to the models of the different disciplines to perform safety analysis.](image)

The initial information model of the AP-233 was based on the core elements produced in a system engineering process based on structured analysis. However, it was soon recognized that object-oriented software engineering has emerged as the dominant software engineering method. This was the reason for proposing an integration of the concepts of the Unified Modeling Language (UML) into AP233. This initiative was probably the motivation for the SysML.

B. The SysML-view

SysML was developed to unify the diverse modeling languages currently used by system engineers. It support the specification, analysis, design, verification and validation of a broad range of complex systems, which may include hardware, software, information, processes, personnel, and facilities [20]. SysML reuses a subset of UML 2 and provides additional extensions to address system engineering aspects not covered by UML 2. It includes diagrams that can be used to specify system requirements, behavior, structure and parametric relationships. Requirements diagram and parametric diagrams are the new diagram types proposed by SysML.

The 3+1 SysML-view architectural model adopts SysML for the system’s modeling process in the mechatronic layer of the MIM architecture, as shown in Fig. 5. The MTS models constructed in SysML constitute the system models, which are available to the MTS developer through the MTS-view. The MTS developer has a black and white box view of composite MTCs, but it has only a black box view for primitive MTCs. On the other side, the MTC developer/integrator has a black and white box view of primitive MTCs.

The use of SysML in the MTS-view establishes a good understanding of the complete system between the engineers of the various disciplines who have different backgrounds and expertise. It provides a common understanding especially during the conceptual phase of the MTS development process, since it will permit the designers to effectively communicate with each other. SysML allows also the synergistic integration of mechatronic components, which are the constituent parts of mechatronic systems. The main view in the MTS development process is the SysML-view that corresponds to the mechatronic layer of the MIM Architecture. This view captures the system model that is the one constructed by the MTS developer. Each of the three views is used to describe the system from the perspective of the corresponding discipline. The view stereotype of SysML can be used to represent the different views involved in the mechatronic development process. A view according to SysML is a representation of a whole system.
or subsystem from the perspective of a single viewpoint. Views are allowed to import other elements including other packages and other views that conform to the specific viewpoint. This means that the MTS view will import all the three views, while the three views will import the SysML view.

C. The Mechatronic Component and the MTC package

We represent the construct of mechatronic component using the Block construct of SysML. During the MTS architecture definition process, the developer exploits already defined MTCs. These MTCs can be defined either by system product line analysis or by domain analysis. System specific MTCs are also defined during the architecture definition process. MTCs are discriminated to composites and primitives. MTCs that may be further decomposed into other MTCs are considered composites. MTCs that cannot be further decomposed into other MTCs or the developer decides to avoid further decomposition in terms of MTCs, are considered as primitive MTCs. Primitive MTCs are developed with generic functionality that enables their use in different systems of the same system product line or domain. These MTCs may be discovered through the web [21] and utilized for the development of systems that require the specific functionality in the mechatronic layer. For example, a Feeder MTC that appears in the architectural design of Festo MPS, is described by the required responsibilities and their QoSs. This MTC is later refined in subsequent steps up to the detailed design specification, which is next transformed to an implementation, that represents the same MTC. There is no need to use different names to refer to the different abstraction level representations of the Feeder. The Feeder MTC implementation refers to the real world Feeder. The terms Feeder MTC analysis construct and Feeder MTC design construct are used to refer to the more abstract representations of the Feeder MTC. This naming convention is compliant with software engineering rules in contrast to the use of the terms “mechatronic information object” [5] and “its describing information” [6].

For the development of the Feeder MTC, the MTC builder works vertically, either top-down or bottom-up, in the lower three layers of the MIM architecture in a concurrent way, to find the optimal solution for the assigned to the Feeder MTC responsibilities. The MTC builder works also horizontally in the model integration dimension and applies an information integration process that crosses the boundaries between mechanical, electronic, and computer science fields. The resulting Feeder MTC implementation construct, i.e., the real world MTC, is part of what is called Feeder MTC package. An MTC package (MTCPackage) except of the real world component (RealWorldMTC) includes also the MTC’s metadata and the simulation model of the MTC as shown in Fig. 6. The MTCsMetadata and the SimulationModel are stored in MTC repositories to be discovered and used by mechatronic system integrators. The Feeder RealWorldMTC is composed of the mechanical part, the device, or devices that host the controlling software, and the compiled code that constitutes the implementation of the software part of the MTC.

A special interface description language, which we call mechatronic component interface description language (MTC-IDL), is required for the description of MTC interfaces. This language should be defined on top of the interface description mechanisms of SysML by incorporating artifacts from mechanical and electronic description languages. MTC-IDL specifications should automatically be synthesized by proper model-to-model transformers from MTC-based design models.

MTCs may be directly interconnected assuming compatible interfaces. Moreover, a special kind of MTCs, the so called MTConnectors, can be used to capture extra coordination logic or handle incompatibilities between interfaces. MTConnectors can be used to define characteristics of material, energy or information communication channel, a synchronization model for communication, a communication protocol, etc. For example, a pipe can be modeled as an MTConnector that represents a material transfer channel. A belt is also a classical example of an MTConnector. If we model a pipe or a belt as an MTConnector, extra behavior and intelligence can be appended to the classical one, i.e., the poor material transfer, to get extra functionality. In this case, the pipe and the belt will have their own processing unit on which the basic software that monitors and/or controls the mechanical part, would be executed. This converts the traditional mechanic pipes and belts to smart ones.

D. The Composite MTC development process

The definition of system’s structure in terms of MTCs is a design process that results in the selection of system components and the definition of their collaboration. The activity diagram of Fig. 7 presents the development process for the composite MTC. Responsibilities which are assigned to each MTC are handled in the subsequent phases as its required responsibilities. If the MTC is primitive (primitive MTC in Fig. 7), an MTC synthesis process is applied. Otherwise, its architecture is defined and responsibilities are assigned to its constituent components, as shown in the activity diagram of Fig. 7. For the assignment of MTC responsibilities to the constituent MTCs (const. MTC’s, in Fig. 7), an iterative process is applied down to the primitive MTC level.
The 3+1 SysML-view model provides, as shown in Fig. 5, a framework for the synergistic integration of the various disciplines involved in the development process of mechatronic systems. It provides a proposal for an integration infrastructure for the three disciplines with the system level model of the mechatronic system, which is expressed in SysML. Specific model-to-model transformers will automate the model transformations between the models in the different domains.

A. Requirements handling and traceability

The 3+1 SysML-view model addresses the requirements handling and traceability challenge: a) through the adoption of SysML, and b) by capturing requirements in the mechatronics layer using the requirements diagram of SysML. To address the weakness of UML to handle requirements, SysML has introduced the requirements diagram and it has adopted a more effective use of use case diagrams. UML has adopted the use case diagrams to capture requirements. However, even though use cases are an effective tool for functional requirements, they are not suitable for capturing non-functional ones, which is a very important activity for mechatronic systems. We integrate use cases with the MTS requirement diagram by relating them to requirements of the requirement diagram using the refine relationship. In this way a use case is considered as a model element that refines a specific requirement. Moreover, the steps of a use case description may be captured as individual requirements in the MTS requirement diagram to get a more granular traceability between the use case and the remaining model elements of the MTS. This latter is related to the level of abstraction in use case definition. Based on the above, the functional requirements at the mechatronics layer are captured in the requirement diagram but they are refined using use cases. Safety requirements, that are not expressed in the form of required by the system behavior, have to be captured in requirements diagrams, as non-functional requirements. Later, during the development process, these requirements are handled as constraints.

The MTS requirement diagram is also used to represent many of the relationships that exist between requirements. It can also be used as a bridge between traditional requirements management tools, if such tools are used for handling the MTS requirements and the other SysML models. We use the concept of slave requirement to address the need for requirement reuse across product families and projects in the mechatronics domain. Typical scenarios are requirements that are applicable across products and projects, and requirements that are reused across families of MTS products.

Traceability is supported through the SysML requirement relationships which allow the MTS builder to relate requirements to other requirements as well as to other MTS models. These include relationships for defining a requirements hierarchy, deriving requirements, satisfying requirements, verifying requirements, and refining requirements. For example, we use the satisfy relationship to describe, during the MTS architectural design phase, how a design that is intended to satisfy the requirements, satisfies one or more requirements. We use the verify relationship to associate a test case, which verifies an MTS requirement, with the corresponding requirement.

B. Support for decision making

Decision making during the MTS development process is a
very complicated activity. In fact, the whole development process is a decision making process that exploits the analysis space, the experience, and the skills of the developer to define the optimal system that satisfies requirements. Decision making is in many cases a multidisciplinary process that imposes the designers of the various disciplines to effectively communicate with each other. Therefore, a medium that will be understood by all involved developers is required. The 3+1 SysML-view model approach addresses this challenge by the definition of the SysML-view and the use of SysML for constructing the MTS model. The SysML model of the MTS is used as the communication vehicle between the various disciplines.

Among the important decisions in the MTS development process we discriminate: a) the selection of the right components, and b) the definition of the collaboration scheme between constituent components. The selection of the right component is simplified by the infrastructure of the MTC package. Already existing components are stored in component repositories. Services of these repositories support a search based on the required responsibilities and their QoSs [21]. Requirements from other MTCs as well as QoSs for these requirements for the candidate components, which are provided by the search process, simplify the decision of the right component.

The next decision is on the definition of the collaboration scheme among the constituent components in order to get the requested system level behavior. The use of sequence diagrams to describe the behavior allows the estimation of the QoS of the specific collaboration scheme. The design is accepted, if the QoS characteristics of the specific collaboration meet the QoS requirements of the required, at the upper level of composition, responsibility. If the design is not accepted, corrective actions are proposed and analyzed. Alternative collaboration schemes may be analyzed to find the optimum one. Otherwise, components with better QoS characteristics may be used. Decisions such as design solutions are captured using the rationale construct of SysML, which can be attached to any model element or relationship. Moreover, we use the rationale stereotype to document the justification for several types of decisions such as design, verification and test case decisions.

The 3+1 SysML-view model assumes the execution of MTS models so as to be able to check a design decision at the early stages of development. However, the only alternative for model execution is to integrate existing simulation tools from other disciplines, since there is no engine today to execute SysML models. The integration of SysML with the Modelica language [23] is towards this direction.

C. Maintaining consistency

The models that constitute the MTS model change due to their refinement or evolution and impose the need to preserve their consistency. As claimed in [16] “Consistency can never be guaranteed […], however, a capability to avoid as many inconsistencies as possible aids in avoiding wrong decisions and potential losses at different stages of design.” Consistency of a model is referring to the coherence among its things or parts. Inconsistency is informally defined in [24] as “a state in which two or more overlapping elements of different software models make assertions about the aspects of the system they describe which are not jointly satisfiable.” In terms of the MTS model, we discriminate between consistency, i.e., absence of contradictions, among:

- the things that constitute a single model itself, i.e., the architectural model,
- the models that capture different aspects of the system in the same discipline and constitute the MTS model, e.g., structural and behavioral models in the s-view, and,
- the models of different disciplines that constitute the MTS model.

The third category is the most complex to manage since it crosses the discipline boundaries. The communication gap, which is introduced by this category, increases the level of complexity in handling inconsistencies. In the 3+1 SysML-view model, the MTS-view is used to bridge the gap between high-level systems aspects such as requirements, use cases, architecture defined in SysML, and the executable dynamic system model described and executed on the m-view specific tools. Consistency between the three views, i.e., the s-view, the e-view and the m-view is preserved through the SysML-view. The SysML-view supports traceability between different artefacts through the requirements and parametric diagrams and the various types of dependencies among which refine, satisfy, verify, deriveReqt, and allocation relationships.

In [25], authors describe an approach to manage dependencies between dynamic system model components and high-level system aspects that have been captured in SysML. A two steps approach is proposed to capture these dependencies. In the first step a high level definition of the system’s dynamics is given in the SysML model, which is then followed by a partial definition of detail system dynamics using the SysML4Modelica profile. Dependencies between the constructs of the dynamic model and other SysML models of the system are captured next. This is followed by a translation of the SysML4Modelica compliant model into a Modelica model, and the refinement and completion of the dynamic behavior. Simulation of the resulting model and analysis of the simulation results are the next steps. A model transformation from the Modelica model to a SysML model, compliant with the SysML4Modelica profile, follows. This transformation allows the developer to capture the model dependencies between constructs of the detailed dynamic model and the ones of the high level SysML model of the system, which are missing from the initial, i.e., the SysML model. Forward and backward traceability is therefore established in a tool level that allows to effectively manage the inconsistencies between the models of the various levels of abstraction of the system.

VI. CONCLUSION

Even though several new methodologies regarding the development of mechatronic systems have been already proposed, the challenges in the mechatronics domain are still open. As claimed in [4], questions are more than answers.
Moreover, the different terms used by researchers working in this domain, complicate even more the already existing communication gap between the disciplines that are involved in mechatronic systems. There is no commonly accepted terminology and even more, there are no basic principles on which a solid framework for the development of mechatronic systems will be based. In this paper, the 3+1 SysML-view model is refined and discussed regarding open challenges in an attempt to identify the key concepts of such a terminology and framework. Provisions of this approach for reuse, synergistic integration, size and complexity, requirements handling and traceability, support for decision making and handling of consistency have been discussed. The proper integration of MIM with the SysML, on which the 3+1 SysML-view model is based, is a promising platform for a solid framework for mechatronic systems development.

ACKNOWLEDGMENT
The author would like to thank Dr. Aart-Jan deGraaf for his comments and discussions on a very early version of this work. He is also grateful to the anonymous referees for their remarks and suggestions that have lead to significant improvement of this paper.

REFERENCES