Mechatronics 20 (2010) 876–886

Contents lists available at ScienceDirect

Mechatronics

journal homepage: www.elsevier.com/locate/mechatronics

Towards automation of control software: A review of challenges in mechatronic design

A.A. Alvarez Cabrera [a,*], M.J. Foeken [b], O.A. Tekin [a], K. Woestenenk [c], M.S. Erden [a], B. De Schutter [a], M.J.L. van Tooren [b], R. Babuška [a], F.J.A.M. van Houten [c], T. Tomiyama [a]

[a] Faculty of Mechanical, Maritime, and Materials Engineering, Delft University of Technology, Mekelweg 2, 2628 CD, Delft, The Netherlands
[b] Faculty of Aerospace Engineering, Delft University of Technology, PO Box 5058, 2600 GB, Delft, The Netherlands
[c] Faculty of Engineering Technology, University of Twente, PO Box 217, 7500 AE, Enschede, The Netherlands

ARTICLE INFO

Keywords:
Control software generation
Design methods
Design tools
Development challenges
Function modeling
Integration
Knowledge base
Mechatronics

ABSTRACT

Development of mechatronic systems requires collaboration among experts from different design domains. In this paper the authors identify a set of challenges related to the design of mechatronic systems. The challenges are mostly related to integration of design and analysis tools, and automation of current design practices. Addressing these challenges enables the adoption of a concurrent development approach in which the synergetic effects that characterize mechatronic systems are taken into account during design. The main argument is that in order to deal with software development problems for complex mechatronic systems, there is a need to look at system design practices beyond concurrency, i.e., there is a need to consider the complex interdependencies among subsystems and the designers that develop them. A review on current methods and tools is carried out to identify possible solutions proposed in previous works. The purpose is not to make an extensive review, but to show that integration, from different points of view, is a major issue and that increasing the level of abstraction in the description of systems can help to overcome the integration challenges. An increased level of abstraction also forms a basis for addressing other issues in mechatronic product development, which are presented in this work. With that in mind, concepts for an integration framework are proposed. The goal of the framework is to support a multi-disciplinary design team to (almost) automatically generate and verify control software. Based on high-level architectural descriptions, the software generation and verification process can be supported by knowledge-based methods and tools. Other goals are to support communication among engineers, improve reliability of designs, increase reuse of design knowledge, and reduce development time and development costs.

1. Introduction

Developing mechatronic products requires intensive collaboration between engineers of the mechanical, electronic, control, and software domains in a design team [1–3]. A central issue is that design decisions cannot be taken from the point of view of a single domain, as often they will have an impact in other domains. The procedure in traditional sequential design is that the mechanical design has to be "frozen" before proceeding to the design of control software. This responds to the required preparations for production of the hardware, while for the software still last-minute changes may occur, sometimes to fix problems from the rest of the design. This approach usually does not lead to optimal overall behavior, since it does not properly address the interaction among mechanical, electronic, and control behaviors. Furthermore, it does not reflect the importance of software design, which can have a major impact on overall system design and performance.

Therefore, there is a need for a concurrent engineering approach with a highly integrated development strategy [4,5], where design freeze is based on multi-disciplinary objective and constraint evaluation. Such an approach is referred to as a mechatronic design approach by multiple authors [3,6,7]. The idea that mechatronic design must be considered as a whole has been well known for some time now, as Ziegler and Nichols [8] commented in 1943: "In the application of automatic controllers, it is important to realize that controller and process form a unit; credit or discredit for results obtained are attributable to one as much as the other."

This paper describes the challenges related to the use of this mechatronic design approach and its relations to software generation. The authors revise current solutions and propose extensions and combinations of them. It is concluded that methods based on higher abstraction levels play an important role, but that their implementation is an issue. Furthermore, it is shown that

* Corresponding author. Tel.: +31 15 278 5608; fax: +31 15 278 4717.
E-mail address: a.a.alvarezcabrera@tudelft.nl (A.A. Alvarez Cabrera).

0957-4158/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.mechatronics.2010.05.003

multi-disciplinary design optimization and verification of both hardware and software require suited modeling paradigms and tool support. With these findings in mind, the authors propose an integrated design support framework for mechatronic systems, which also addresses control software development. The framework is supported by a common high-level system model, to provide both a clear system overview for the designers and to enable the transfer of critical product information between tools. So far, the proposed framework outline is intended to deal with the challenges mentioned above, mainly at the early stages of the design process. As such the proposal does not target a defined control paradigm, specific industry, or product type.

In Section 2 the identified challenges in mechatronic design are presented and discussed in more detail. Section 3 gives a review on existing methods and solutions to face those challenges. After that, Section 4 introduces the approach proposed by the authors, based on the discussions about the identified challenges. The conclusions are presented in Section 5.

2. Challenges in mechatronic design

Both academic and industrial sources have reported on challenges related to the design and development of mechatronic systems, such as:

- Exchange of design models and data [2,6].
- Cooperative work and communication among the design engineers [2,5,6,9,10].
- Multi-disciplinary modeling [4,7,9].
- Simultaneous consideration of designs from different disciplines [4,5,7,11].
- Early testing and verification [5,7,9].
- Persistence of a sequential design process [2,4,10].
- Lack of tools and methods supporting multi-disciplinary design [2,4,5,11].
- Support of the design of control software [3,5].

Examining these challenges, three core issues can be identified, which influence many of the problems in the development of mechatronic systems. These challenges relate to design integration, design verification, and generation of control software. In the next subsections, these will be discussed in more detail.

2.1. Design integration

Modern mechatronic systems provide an increasing number of functionalities [1,6], while available energy, space, weight, and time remain constant or even decrease. Specialists from various fields must combine their expertise to develop a single product. A need for tighter integration, which encompasses diverse factors related to design tools and practices, design methods, and members of the design team and their interactions, arises from such trends.

Fig. 1 shows a representation of the current mechatronic design process, where spaces represent common gaps between the different design phases and the tools used in the design. Design teams are often composed separately according to their area of expertise and often work at different locations. The integration phase is postponed until the moment when physical prototypes are available. Even in cases where a certain level of automation in the design has been reached, integration problems can still be found; e.g., to use linked domain-specific libraries of components, it is necessary to verify consistency and completeness of such libraries. These points are elaborated in Section 3.

Integration has been directly identified as an important research direction and a key element in the design of mechatronic systems by industry [9] and by authors like Craig [7], Schöner [6], and Wikander et al. [4]. Tomizuka [3] and Wang et al. [2] identify the importance of aspects closely related to integration, such as cooperative work of designers, data sharing, knowledge management, design project management, and simultaneous design in different domains (e.g., design of the control algorithm and of the system to be controlled). A recent report on industrial practices [5] shows that the leading mechatronic product manufacturers opt for integration oriented towards management of specialist designers and tools that support such an approach, rather than using tools that encompass all detailed design aspects. The desired tools, as identified by these manufacturers, should handle information at the system level and track requirements and design changes to efficiently support integration of design activities. Apart from the need for tools as identified by industry, it is also necessary to consider the design methods supporting these tools and their users to get close to an integrated design approach.

Appropriate methods and tools to support design integration are required, both in the conceptual phase as well as in the detailed design phase, as has been identified by academia [2] and by the engineering community [9,10,12]. The role of the human actors is also important, as communication of ideas and information between designers from different domains is necessary [5,7]. These three factors will be discussed in the next subsections.

2.1.1. Design methods

Despite many research contributions aimed at providing a theoretical framework for the design, this goal has not been achieved yet [13]. As depicted in Fig. 1, design activities might be separated in the sense that parts of the design might depend on data provided by other parts (e.g., the design of a controller may require knowledge of certain physical characteristics of the system).

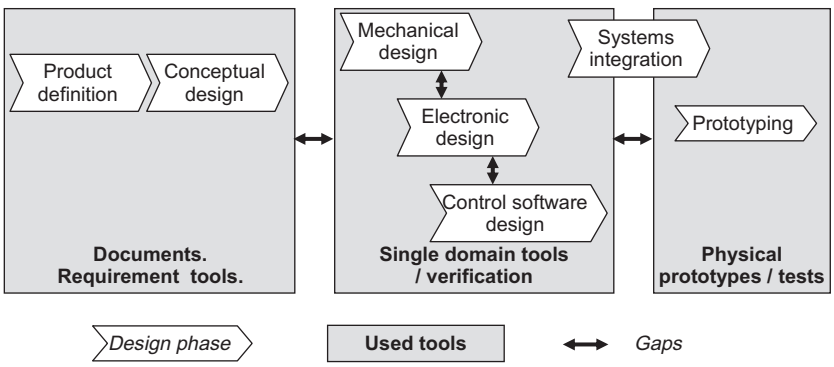

Fig. 1. Common current design practice phases and tools.

Traditional methods in engineering design broadly exhibit either a sequential or a concurrent flow of activities.

As reported by Wang et al. [2], sequential design has proven to be unsuitable because of its lack of flexibility, which increases design cost and development time. This perception is supported by engineers in industry [10]. Rzevski [14] recommends stepping out of the conventional end-to-end (i.e., sequential) design process in favor of a concurrent approach to deal with design of mechatronic systems.

The core of traditional concurrent engineering approaches (see e.g., [15]) is to consider all phases of the life-cycle of the product as early as possible in the design in order to deal with issues related to later life-cycle phases, such as production and disposal [16]. But even traditional concurrent approaches have proven to be limited when dealing with complex design situations, in the sense that strong interdependencies might have unpredicted effects on the overall performance [4]. As mentioned by Wikander et al. [4] and Rzevski [14], a typical approach for the design of mechatronic systems is to build the system by assembling single-domain subsystems and by paying special attention to the design of interfaces among them. Wikander et al. remark that such traditional methods can merely achieve a sound integration of the components (i.e., "something that works"), but not a synergetic integration. Therefore, research on mechatronics should also focus on the interactions of the different engineering disciplines [4] rather than only on the interactions between the subsystems that are being designed.

Dealing concurrently with the interactions of designers and of their designs is of paramount importance for the early detection of problems in product development.

2.1.2. Design tools

Recent reports on industrial practices confirm the use of different tools to manage design data, and state that the lack of tools that allow integration and shared use of such data is one of the main challenges in mechatronic product development [5,11]. As illustrated in Fig. 1, a current tendency is that designers from different design domains rely on specialized tools. Similar to Wang et al. [2], we denote a tool as domain-specific if it supports the design in a single domain, e.g., mechanical or electrical. Examples are tools like SolidWorks in the mechanical design domain, Synopsys and OrCad in the electrical domain, and Matlab/Simulink in the control domain. Furthermore, there are not many specialized tools that support the first stages of design and that also extend efficiently to the subsequent stages, although this limited reach is probably due to misuse of the existing tools. Examples of such tools are requirement management tools like Rational DOORS, and tools that support approaches to capture requirements like Quality Function Deployment (QFD) [17] or Integration DEFinition for function modeling (IDEF0) [35].

Mono-domain tools perform well within their own domains, but their specialization often makes it difficult to consider information from other domains. The tools used in the control design domain in general prove to be more flexible as they use mathematical models as modeling primitives, e.g., in the form of block diagrams or bond graphs [19]. Additional insights on tool integration can be found in the works of Cutkosky et al. [20] and Dolk et al. [21]. The varied nature of the different design tools interferes with a direct integration (i.e., direct mapping of the modeled objects) using a single tool or design environment. Examples that illustrate such variety are:

- In mechanical design, dimensions, shapes, and materials that correspond to the physical objects are the main interest. Thus, representing abstract concepts and grouping parts according to other criteria than physical proximity become problematic.
- In the design of controllers, the physical system, also referred as the plant, is often abstracted to a black box model. From such point of view it is difficult to find the explicit connection between the behavior and its physical causes.
- Electronics deals with the physical implementation of the control. The software packages for electronic design support predictions of behavior and execution time through logical and physical simulations.
- Electric engineering commonly designs "bridge" objects from electronic and mechanical domains, and tools related to it focus on the connectivity of components and the communication among them.
- Requirement management and capture tools focus on representing textual requirements information. The link to other design domains is mainly made through document referring, and it is the job of the user to (informally) connect such documents with the current design data.

2.1.3. Human factors

In part, the integration problem can be traced back to the early phases of design of a system in which its architecture is defined. In the conceptual design phase, the designers choose the solution principles, decomposition, interfaces, and design process planning that will guide the detailed design phases and the way in which designers will cooperate [14]. The selection of an architecture influences the choice of detailed solutions and the integration of those solutions in a rather straightforward manner; e.g., actuating an axis of a machine tool with a linear motor or with a precision ball screw completely changes the configuration of the machine at both the hardware and the software level, and therefore, different groups of specialists will need to interact in each case.

Human communication and cooperation are additional factors that affect design integration. One issue is to communicate the goals and requirements of the design and how they relate to the chosen solution, and to assign responsibilities for such requirements. In order to enable monitoring the requirements throughout the design process it must be possible to decompose and the requirements and to make budgets of resources for them, down to the interfaces of the individual designers. Another issue is to inform the designers on how their part of the solution in the design affects other parts. Individual designers make choices that can inadvertently affect the system as a whole. The design should therefore be tested for consistency and validity throughout the design process.

Both issues strongly relate to the fact that there are currently few methods and tools that support systems engineering and architecting activities and that capture the information produced in these activities in order to facilitate the exchange of information between designers.

2.2. Lack of interdisciplinary verification

The four classical verification methods are demonstration, test, inspection, and analysis [16]. Of these, the first three require physical prototypes to be developed, while the latter is based on a mathematical representation of the system, also known as a model. Developing appropriate models for analysis and a platform to verify various aspects of the system, including control software, represents a challenge. In practice, specific models are developed to perform tests at different stages of the design. Due to the use of domain-specific modeling tools, such models usually correspond to a specific point of view on the system, like either the electrical or mechanical aspects, or continuous dynamics and discrete, sequential behavior [22]. With the expected synergetic effects that characterize mechatronic systems, these separate views cannot capture the overall system behavior. Even more, the analysis of

changing operation modes, defined in terms of state machines, requires reconfigurable multi-domain models, which are often not supported.

Schemes of co-simulation and model sharing incorporate data generated in other domain-specific analysis tools into control design models, for example, as implemented in the de facto industry standard [23] Matlab/Simulink. However, often these dynamic models can be considered as an input/output box in the form of a transfer function, and the explicit relation with the original design input is lost. On the other hand, control and hardware co-simulations also require coordination among different specialists, and as discussed in Section 2.1.3, many challenges remain in that area.

For these reasons, verification and testing of control software still relies heavily on the use of hardware prototypes or breadboards, requiring considerable investment in terms of time and money. In a way, complete system prototypes allow a concurrent, multi-disciplinary verification that can reduce overall development time. On the other hand, besides their cost, the use of prototypes becomes less viable as the mechanical design has to be relatively well specified for their construction. An approach typically used in the aerospace industry is the 'Iron Bird' concept, in which a combination of part of the final hardware and software is used to test and verify the behavior of on-board systems, such as the electrical and hydraulic actuation devices. In this way, system verification does not require building a fully operational system, but it still requires significant investment and the detailing of portions of the design.

2.3. Lack of automation in control software design

In practice, the control system development effort is around 20–40% of the total software development effort [25]. Modern Computer Aided Control System Design (CACSD) tools such as Matlab/Simulink or dSPACE, and software development tools such as Rational Rose provide means to translate control algorithms, in the form of block diagrams and state transition diagrams, to machine-executable code. These code generators eradicate human coding errors, increase reliability and reusability, and reduce development time and effort. Nonetheless, a major part of the control system design is spent obtaining "working" formal models like block diagrams and the values for the parameters that configure each block. The aforementioned tools only help to transform those formal descriptions into control code.

Generating code from a model (e.g., a block diagram or a description in the Unified Modeling Language (UML) [24]) of the structure and logic of the software system is part of what is known as model-based software development. Only some of the top-level companies that design mechatronic systems take this approach and it is not a common practice [5]. In such cases, the primitives used for building such models usually represent objects clearly defined for certain specialists. To obtain a more transparent model that aids integration, it is desirable that the objects used in the model are familiar to the parties involved in the control design, which transcend the control engineers.

To arrive at a formal description that can be transformed into code, the designer must define a control structure and strategy, and think about the implementation of functions for the measurement and filtering of system signals and for the application of the control outputs to the system. Here, 'control structure' refers to the selection of groups of control inputs and outputs that will be handled by a software or hardware control unit, and the term 'control strategy' refers to the type of control algorithm to be used in a particular situation and the control sequences derived from the system's characteristics and the required behavior. Once the control structure and strategy are chosen, design rules and optimization routines can be applied to determine the controller parameters, provided that the requirements are given in a suitable form. Often, however, these requirements have to be derived by the experts first, as system requirements specifications are defined at a higher level of abstraction. There is still much to be gained by supporting and automating the control design tasks in the early stages of design mentioned earlier in this section.

3. Review of available approaches

Both academia and industry have come up with methods and tools to deal with the challenges identified above. In this section we discuss a selection of these methods and tools, grouping them in the same way as in the previous section.

3.1. Design integration

3.1.1. Design methods

Various methods consider the modeling of functions, requirements, and other information that is usually defined at the conceptual stage of the design. Documenting such information helps the designer to maintain an overview of the system and to keep track of the evolution of the design. Multiple authors have proposed models that contain functional descriptions of systems, like Function-Behavior-State (FBS) [26], Functional Representation [27], Schemebuilder [28], and MACE [29], to guide and improve choices made in the first phases of product design. These models represent knowledge about the functions of the system, complemented with information about how the function is accomplished and which objects, both hardware and software, are involved. For example, some functional modeling approaches complement this information with qualitative (e.g., Qualitative Process Theory and Qualitative SIMulator [30]) or quantitative (e.g., differential equations, bond graphs) data. Example applications are mentioned by Erden et al. [31]. The FBS methodology has been implemented in the software framework KIEF [32] to integrate tools from various domains and to facilitate the transfer of information, as discussed in Section 3.1.2. Other approaches use functional flow and block diagrams, and they model functions as transformation stages of matter, energy, or information [1,33–36]. So the functionalities of the system are then documented separately from other models. The IDEF0 method [35] offers a formalization for functional flow diagrams and various IDEF languages [18] model details of the system that could be connected more directly than the functions to other domain-specific models, but they do not provide a clear connection between the different IDEF models. The functions and key drivers method (FunKey) [37] proposes allocating budgets of resources to the functions of a system. In this way, FunKey pursues its goal of documenting the architecting process and of providing a means to compare product architectures.

The implementation of these methods is a challenge. As in the case of other theories related with design, either the approaches are not implemented in a tool, or the developed tools are not part of common industrial practice [13]. Furthermore, functional descriptions are mainly used to aid the designer in the identification of related information, but not to classify or identify such information with the help of an automated system. This stems from the fact that these abstract representations have proven to be hard to formalize. Another important factor is that there is not even a consensus for definitions and formalisms in the field of design research [13]. Additionally, requirements information is not included in most of these methods. An exception is FunKey, which mainly focuses on the system budgeting aspect. In particular, QFD specializes in capturing user requirements and connecting them to characteristics of the system that can be used to measure the fulfillment of those requirements.

Muller has proposed the Customer objectives, Application, Functional, Conceptual, Realization model (CAFCR) to decompose the product architecture into the five views its name indicates [38]. This allows for independently capturing the needs of the customer, the functions the product performs, and the design of the product from the conceptual and realization standpoints. Its main purpose is to provide mechanisms to keep track of stakeholder concerns, like safety, usability or performance, in order to maintain integrated goals throughout the whole design process. The work of Muller mentions what relevant information should be considered to obtain a proper description of the architecture of a product, and suggests methods to capture such information. However, these methods are not strongly linked to each other. The large variety and number of methods mentioned in CAFCR brings more flexibility, but leaves to the systems architect or designer the, sometimes difficult, task of choosing the most appropriate method out of all the presented methods.

The V-model [39] sets a general flow for the development process of a product. It indicates that each stage of the product definition should be used to systematically test the implementation as subsystems are integrated to arrive to the final product. Different stages of development and testing are defined depending on the source, but in general, requirements analysis, architecting, detailed design, and the corresponding verification/validation stages are defined. The model provides a structured base for the development process, but it is very general, and does not provide details for its implementation; there are no tools to fully support it, and companies have to carefully develop a framework of tools to model each definition phase and to put the test phases into practice. Though not explicitly specified in the V-model references, analysis verification methods (cf. Section 2.2) are crucial to support the definition stages and to obtain correct models that can be used for verification. At this point it is worth mentioning the spiral model [40], which has similar goals as the V-model, but which considers several iterations using prototypes to verify the design at one stage and to produce a base for the next one.

The axiomatic design method, presented by Suh [41] states that functional independence of the system constituents leads to an optimal design. To attain this, the method provides guidelines, namely, the axioms of independence and information, to compare and evaluate early design choices. Suh and other authors also report that the method has been applied successfully in multiple situations [41]. A crucial point from the axiomatic design method is the importance of linking high-level information (functional requirements) to implementation specific information (design parameters). On the other hand, modern mechatronic products implement an increasing number of functionalities while maintaining constraints on space and costs, and thus, a tight integration of the subsystems is desirable, which makes it harder to obtain functional independence.

Capturing and integration of information is important to deal with the challenges discussed here. The Knowledge and Information Management project [42] has proposed principles that describe the characteristics of engineering information that should be captured and kept for reuse.

In this section we have shown how several methods deal with one or more aspects related to integration, but gaps exist between early design phases and the detailed design phases.

3.1.2. Tool integration

According to Citherlet et al. [43], there are four different approaches to multi-disciplinary tool integration: stand-alone, interoperable, coupled or linked, and integrated programs. The first one is the least desirable, as the tools are unrelated and communication is not possible. Interoperable programs provide means to exchange or share models. Towards these goals, additional frameworks have been developed to streamline or automate the model exchange. This second approach will be treated in more detail later on in this section. Coupled or linked tools can communicate at run-time. Due to the flexibility of their modeling primitives (cf. Section 2.1.2) some tools used in the control design domain have taken the second or third approach. Finally, integrated programs facilitate work in different domains within a single tool. Vendors, especially those of mechanics CAD tools, have used this approach, integrating tools from other domains into their software suites. As an example, the latest version of CATIA also supports electronics, systems and control modeling, and incorporates embedded control code generation for the latter. Though the existing coupled and integrated programs provide a way of predicting the behavior of a system, they specialize in running models used in detail design and lack a direct connection with information from earlier phases of the design process (e.g., goals, functions).

Within the interoperable integration approaches we can mention the pluggable metamodel mechanism implemented in KIEF [45] and the framework of the Virtual Reality Ship (VRS) systems project [44]. The VRS project reference indicates that several tools used in the European ship building industry, including a physical testing platform, have been integrated, but unfortunately no details of how this is done could be extracted from the available material.

The core of KIEF is a knowledge base in which objects from different modeling tools are mapped to each other using "physical phenomena" as connecting points [32,46], in what is known as the process-centered approach [31]. This knowledge base also contains information about modeling tools to support their integration into the framework. A metamodel of the system is built according to the ontology underlying the knowledge base and KIEF manages the data transfer and consistency between the domain-specific modelers. An ontology can be defined as a formal representation of a set of concepts within a domain and the relationships between those concepts, and as such can define a language for communication between domains.

The software suite CORE [47] offers integration through a model-based systems engineering approach. The tool allows to make models to capture requirements, to model function decomposition and flows, and to map them to models of system components and their interfaces. It implements a concurrent design process called 'the onion model' [48] to validate the product definition stages subsequently within its models. Such a tool can support a good portion of the 'left arm' of product specification of the V-model (see Section 3.1.1), but lacks a direct link to the models and tools used in the detailed design and the subsequent testing phases (verification). Nonetheless, the models provided by this tool can be related manually by the designer, outside the CORE tool, at the level of components.

A component-oriented approach that also corresponds to interoperable integration is proposed by Peak et al. [49,50]. A framework based on the Systems Modeling Language (SysML) [51] is used to integrate information from different tools (e.g., CATIA, Ansys, Matlab/Simulink). Using a combination of SysML and the Composable Object (COB) [52] paradigm it is shown how to represent knowledge about a system and to link such knowledge to tools that can use it to build other models. COBs combine the structural and behavioral descriptions of a system. In this object-oriented approach the models can be built in such a way that they are both human- and machine-readable. COBs also form a basis for the integration of different views on a system, as shown by Peak et al. [49].

In support of multi-disciplinary design and optimization a framework called a Design and Engineering Engine (DEE) has been developed by La Rocca [53], see Fig. 2. Relying on a knowledge-based engineering platform, a DEE is a domain-independent tool

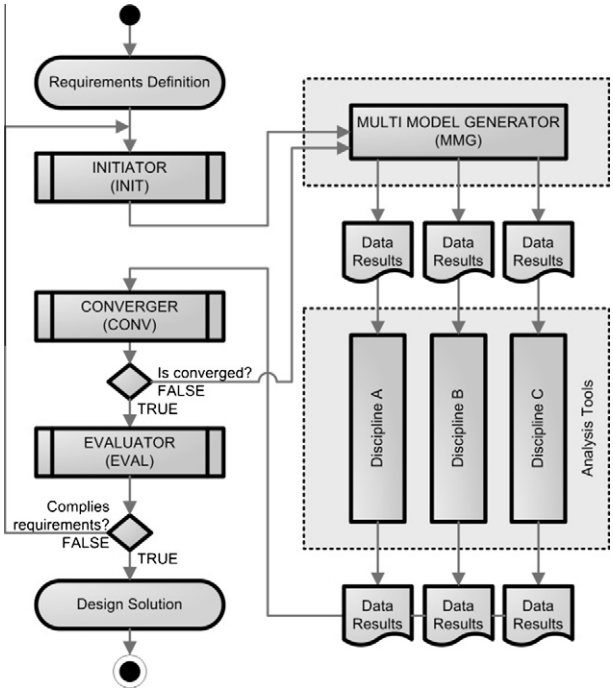

Fig. 2. Design and Engineering Engine [52].

suitable for the design of a variety of systems from multiple domains. The core of a DEE is the 'Multi-Model Generator' (MMG), which is responsible for the instantiation of a product model built from a set of parametric, object-oriented modeling primitives. Furthermore, the MMG processes the product model to generate input for domain-specific analysis tools, which are responsible for the evaluation of one or several aspects of the design. In this way, aspects such as aerodynamic performance and structural stiffness can be analyzed, all based on the same product model. Data sharing between the various tools is enabled by using an agent-based network [54].

Although the approach of the two methods discussed above is different, both rely on a product model-based on components to integrate multiple views on the system. The models from the CORE tool can also be manually integrated to other design information at the component level. This originates in the fact that most parameters and data are directly related to these components. The object-oriented properties ensure that components sharing parameters or data can be easily grouped into a new composite component. The component-oriented approach may be intuitive and fast at the moment of building models, but each modeling object can only be used in a specific situation. For example, a "gear pair" component used in a transmission must be defined in a completely different way than a gear pair used to grind material. A process-oriented approach can help to deal with these kinds of situations, by separating behavior and modeling primitives. The metamodel in KIEF uses such an approach. It relates all concepts of the system through their attributes to physical phenomena and laws, giving more applicability to each modeling object (cf. Fig. 3).

Recent interviews with mechatronic product development companies [11] reveal a problem with the fact that different disciplines use separated design tools and data, which hampers communication among them. The same interviews show that better results can be achieved when using specialist engineers working in well-coordinated groups rather than mixed groups with cross-disciplinary managers. Based on this, we conclude that a promising approach is to provide different modeling environments tailored to each domain, while integration is handled at the "back side" of the tools as a communication support mechanism. The next section treats efforts to overcome the communication issue in more detail.

3.1.3. Human factors

As argued in Section 2.1.3, it is important to consider human factors involved in the design if one wants to achieve an integrated design approach. The communication between the people involved in the design of a system, including stakeholders, is of special interest. Tomizuka [3] mentions that effective communication with others is a necessary requirement for the engineering practice, even more when considering that nowadays engineers must work in teams in design mechatronic systems. Industry also recognizes the importance of the communication among engineers [9,10].

Pahl et al. [1] identify communication and exchange of information between designers as one of the fundamental aspects of their systematic design approach that relates to division of work and collaboration. They mention methods like brainstorming and group evaluation to support the information exchange activities. As Pahl et al. comment, these methods are especially helpful for the search of solutions in the conceptual phase, and thus are focused towards that end in their work. Unfortunately, such methods seem less appropriate for being extended to later stages of design, because they have been conceived to deal with less detailed information than the one required for such design phases.

Although the importance of communication among engineers and information exchange has been widely recognized, to the best knowledge of the authors, there are no tools supporting the design activity while extensively considering these aspects, e.g., integrating the individual work of the engineers using their own tools together with an overview of the system and its goals.

3.2. Lack of interdisciplinary verification

As discussed in Section 2.2, in practice the use of domain-specific modeling tools limits the design and the verification to a specific point of view on the system. FEM models are used to verify strength and stiffness of the mechanical design, CACSD tools are used to develop and verify controllers, and data is transferred from one tool/domain to the other when required. Following an analysis method for verification plays an essential role in early multi-disciplinary verification of the design; the onion model discussed in Section 3.1.2 is an example of this. Often, real multi-disciplinary verification can only take place at late stages in the design process, when hardware prototypes are available. In relation to controller design, the use of hardware-in-the-loop and rapid control prototyping relies on these hardware prototypes. Though this is common practice, the reliance on prototypes makes this approach less suitable in a concurrent design environment. Our focus is to find alternatives to the use of physical prototypes, also to avoid the other disadvantages presented in Section 2.2.

The multi-domain dynamics models used in control design are often transfer functions, modeled with block diagrams in tools as Matlab/Simulink. Two other types of simulation models can be identified for this purpose: models of the first type are based on 'physical modeling' methods, which rely on differential equations and energy flows to describe the behavior of systems; models of the second type are based on geometric modeling, either in combination with finite-element meshes and solvers, or with multi-body dynamics solvers.

A drawback of the use of controller design tools to integrate multi-domain effects in system design is that the user often focuses on the design of the controller for the given model of the system. The 'black-box' nature of the plant models used supports that statement. In order to shift from controller design to system design, physical modeling languages like bond graphs [14], Modelica

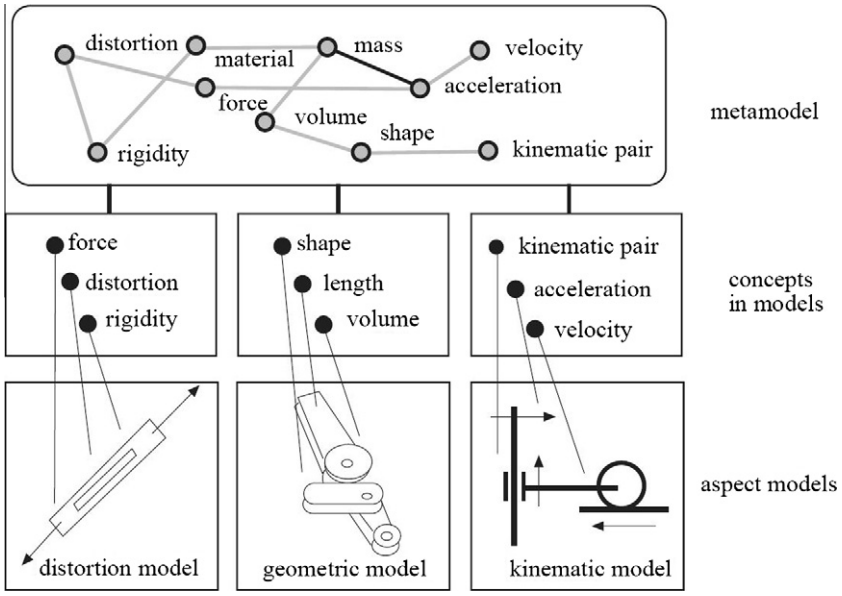

Fig. 3. Metamodel mechanism in KIEF [26].

[55], and SimScape [56] provide the user with graphical modeling elements representing physical components from various domains, such as electrical motors, resistors, and mechanical gears. The obtained system of differential equations is subsequently solved by the supporting tool. These tools often also allow for the modeling of signals and discrete events [57,58]. Due to the port-based approach, simplified models which are used early on in the design process can be replaced with more detailed models as the design matures.

The bond graph language from Karnopp et al. [19] has been promoted for the modeling of mechatronic systems by authors like van Amerongen [59,60]. The bond graph tool 20-sim consists of a block modeler, a set of control analysis methods, and a basic 3D modeler which can be used to link the block diagram representation to a mechanical model. Ferretti et al. [61] state that mutual interaction between domains, modular and object-oriented modeling, and reuse of modeling components using libraries and customization are required for a modeling and simulation tool for mechatronic systems. Their conclusion is that the combination of the Modelica language and the tool Dymola satisfies most of these requirements. There are various similar modeling and simulation tools available, both commercial and academic. These tools include gPROMS [62], SABER [63], HyBrSim [64], and Smile [65].

A disadvantage of these multi-physics modeling tools is that the model is based on assumptions about the expected behavior, such that a significant experience is required to know which assumptions are valid. For example, thermal effects can have a considerable influence on electronic components, but the designer needs to know the relative position of the heat source and the electronics to decide whether or not to take this into account. The use of first-principle based simulations, i.e., using finite-element analysis, is a way to (partially) circumvent this.

Simulation based on finite-element methods relies on 2D/3D CAD models. Various commercial CAD tools are available nowadays, and their use is a well-established industrial practice. Vendors of these tools often provide additional tool suites for finite-element analysis, covering domains such as mechanics and thermodynamics. Specialized multi-physics simulation tools, e.g., COMSOL, allow for simultaneous analysis of phenomena from different domains. To prevent consistency problems, often the geometry models developed in dedicated CAD tools are imported in the specialized tools, instead of being developed only for this purpose [22].

Results from these various analysis tools can subsequently be used in models that are used in the controller design, albeit via manual data transfer. The direct use of finite-elements tools in combination with controller design tools for verification purposes is computer-intensive and time-consuming, but might, however, in the long term be faster and cheaper than physical prototype-based testing.

To prevent the manual transfer of data, Voskuijl et al. [66] has used a combination of a Simulink-based aircraft dynamics model and computational fluid dynamics (CFD) analysis for the design and optimization of a blended-wing body aircraft. Albeit custom-developed, it shows that domain-specific analysis can be integrated in a multi-domain analysis and optimization tool. The DEE concept discussed in Section 3.1.2 applies a similar approach, in which multi-disciplinary analysis, optimization, and verification are supported by an integration framework.

With respect to the verification of discrete, event-driven control algorithms, there are various methods available, depending on the formalism in which the algorithm is defined. These methods are used for checking the existence of dead-lock situations, unreachable states and transitions that do not occur, among others. For realistic model-based verification, the model of the system should reflect the changes in operation mode, e.g., by reconfiguring the active actuators.

3.3. Lack of automation in control design

It must be stressed that in this work the automation of control software covers more than just the generation of control code out of a detailed control software model, and extends to obtaining such model (cf. Section 2.3). There are various commercial code generators available, both for Matlab/Simulink-like environments and UML-based modeling tools. The Gene-Auto project has developed methods for automatic model transformations, focusing on a "correct by construction" approach [67], such that the code can be implemented on critical embedded systems in the aerospace and automotive industry. By verifying the code generator itself, it can be used without the need to verify the generated code. To integrate design formalisms for continuous and discrete-event control, an

integrated design notation is used in both the PiCSi [22] and the Flexicon project [68]. UML is used as a common language, into which both Simulink models and Sequential Flow Charts are transformed. From the combined control system, platform-independent Java code can be generated. Again, the use of proven, domain-specific tools and methods in combination with a translation to an integrated model is preferred above a new and integrated "do it all" language. In contrast to this, the application of domain-specific modeling (DSM) languages to raise the level of abstraction of control software design relies on specific modeling elements. It removes the need to map elements to domain-independent languages as UML before code generation can be applied and as such decreases development time [69]. For DSM to work, however, the language and code generation tools have to be developed by one or more domain experts.

In terms of automation of the control design much can be gained in the early phase when requirements are translated into control structure and logic. Message Sequence Charts and UML sequence diagrams can be used to specify required behavior, but these specifications are considered to have a weak expressiveness [70]. Instead, Live Sequence Charts have more expressive power. By formalizing communication between actors over a timeline, Live Sequence Charts provide means to automatically derive control software logic and structure from them, e.g., in the form of UML. As discussed in Section 2.3, the generation of code from the latter description is possible, but not widely applied yet.

To get from requirements to control software, a method based on Requirements-Based Programming (RBP) is proposed by Rash et al. [71]. RBP should increase development productivity and the quality of the generated code by automatically performing verification of the software, which is supported by an approach that ensures that the application can be fully traced back to the initial requirements of the system. A more direct link between (functional) requirements and software has been achieved by the use of the Functional Block computer-aided design environment [72]. The prototype tool can be used to design and analyze reusable high-level control software components and to generate run-time code for distributed control systems. The applicability of such a direct approach, where functions and software code are directly linked, to continuous-feedback control software is however not straightforward, because of the strong dependency on the system properties.

Another approach that starts from high-level specifications is presented by Sakao et al. [73]. The input specifications are modeled in FBS [26] using qualitative descriptions. Qualitative reasoning techniques are used to derive a sequence of activations from the actuators, and quantitative information can be added to the resulting sequence. The method is only implemented for a specific case, but a patent [74] shows aspects of the control sequence derivation that could be used in generic cases.

Partial automation of the control development process can be obtained by instantiating pieces of pre-developed control code from databases linked to specific system components, e.g., sensors or actuators. For example, this approach has been implemented on a large scale by a company specialized in handling and transport systems of goods. In that company, around 80% of a the PLC controller code in a system can be generated from component descriptions and associated code elements. These code elements, stored in company-specific libraries, contain routines to execute most of the low-level tasks for each type of component; e.g., start up, shut down, and emergency handling sequences for an electric motor. Service functions and irregular situations have to be predicted by the engineers and programmed manually. Integrating generated code with manually written or existing library code removes part of the advantages of automatic code generation in this case.

4. An integrated approach for control software development

The need for a revised concurrent engineering design approach was identified in Section 2. The use of domain-specific design methods and tools to develop an integrated, multi-disciplinary system has inherent drawbacks, related to multi-domain modeling and the communication between designers and tools. The review in Section 3 has shown that methods based on higher abstraction levels play an important role, but that implementation is an issue, and that multi-disciplinary design, optimization, and verification of both hardware and software require suitable modeling paradigms and tool support. The framework proposed here is intended as a support for such a design approach. There are two important reasons not to develop a single tool for mechatronic system design to tackle the identified challenges. First, the design information of an entire system is too big and complex. On the one hand, creating a model that contains design information with the necessary detail would increase the model size, and create a bottleneck to access it [20]. On the other hand, providing the operations to model and handle the different kinds of design data in a single tool constitutes another barrier. Second, existing tools are designed and optimized for specific domains, and the designers are proficient with these tools. In practice, each designer is responsible for creating and maintaining the models related to her or his discipline [75].

As an alternative to the single tool approach, we propose an integration framework to support mechatronic system development, and in particular, the design of control software. Fig. 4 depicts an overview of the framework, with existing, domain-

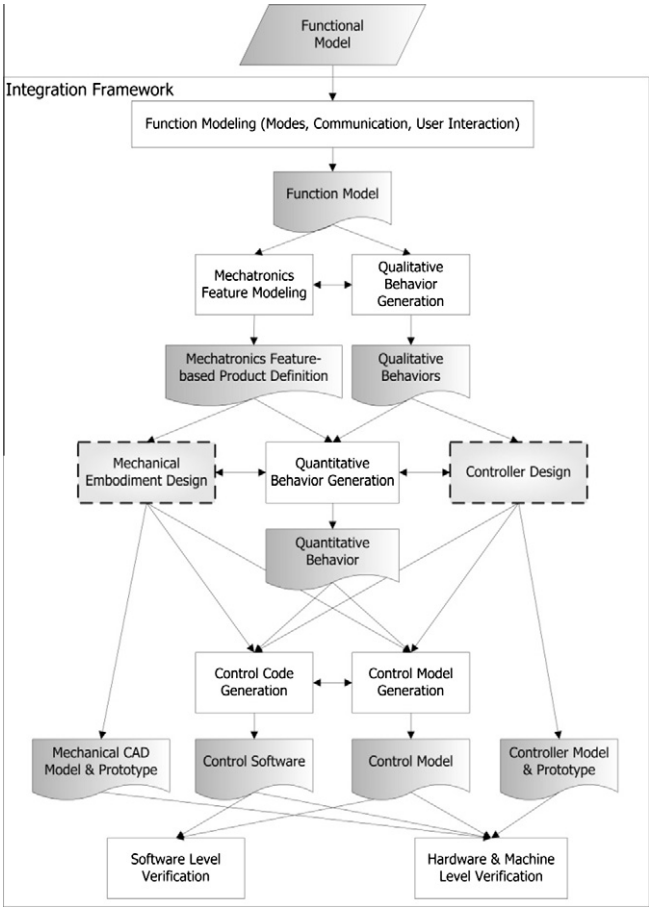

Fig. 4. Architecture of integration framework. White blocks represent tools to be further developed. Dashed-line blocks correspond to existing commercial software tools.

specific tools represented as dashed-line boxes. For clarity, design iteration loops are not included in this figure. This approach can be classified as an architecture framework, as defined by Browning in [75], and similarly, it aims at supporting the communication among developers and model transformations between tools. This addresses the challenges related to cooperation and communication discussed in Section 2.

A high-level or abstract functional information model appears in the upper part of Fig. 4. The intention is to obtain a backbone to navigate, give an overview, and classify detailed design information, by capturing functions, requirements, and the architecture of the system. In this aspect, the proposal follows the line of reasoning of the methods presented in Section 3.1.1, additionally aiming towards integration. The basic hypothesis for the use of functions as integration elements is that from the functional point of view it is possible to describe a system at different levels of detail, focusing on the points of interest to the user while maintaining coherence of the model. To that end, the initial design is specified using the FBS modeling scheme [76,77]. System-level choices related to control design are also included in this system model, at the level of strategies and goals. Quantitative requirements and constraints can also be incorporated at this stage, using a formal specification language. The architecture is initially specified in terms of objects, which can correspond to software, hardware, mechanical components, etc. Since this information is beyond the scope of the original FBS scheme, the proposed model will use an extended version of it.

Besides high-level information, a model of the system requires information of the analysis (e.g., controller simulations) and descriptive models (e.g., manufacturing blueprints) necessary for implementation, as indicated by the block of ‘‘mechatronic feature product definition” in Fig. 4. For this purpose we propose the use of the object- and component-oriented techniques discussed in Section 3.1.2. Such techniques allow for the building of models in an intuitive and fast manner once the modeling elements have been developed. The main difference between our proposal and the component-oriented tools presented in Section 3.1.2 is that we aim at building a hybrid representation in the knowledge base that merges the component- and process-oriented modeling paradigms.

Information needs to be exchanged between different domain-specific design tools, like mechanical CAD or CACSD software, in order to integrate the different design activities and to automate analysis, synthesis, and model transformation. An information manager should provide means to navigate, visualize, and ensure consistency of the system model and the associated modeling data. The integrated results obtained from the domain-specific tools are labeled as ‘‘quantitative behavior” in Fig. 4.

At the architectural level, multiple views on the overall system can be discerned. These views can be used to capture and trace the concerns and requirements of stakeholders [78,79]. System-level requirements must be decomposed or budgeted and tracked back to the various subsystems and the different domain-specific design processes.

Our proposal addresses the challenges from Section 2.3 and complements many of the approaches presented in Section 3.3 by obtaining the control structure and strategy, starting from the models presented before. For the controller design, the requirements defined at the architectural level are transformed into control design specifications, both for sequential-supervisory control as well as for continuous-feedback control. For the first, these specifications may be state machines defining operation modes; the second often contains requirements in terms of allowable overshoot or rise time, or the used power. From these, a cost function can be derived, and subsequently the controller (e.g., PID or Model Predictive Control) can be (semi) automatically designed and tuned using existing software tools, e.g., the control system toolbox in Matlab. Our proposed approach is similar to the Control Design Method (CDM) [72], which fills the gap between the functional design and the final implementation. In the framework proposed here, however, the control system and the plant model are developed in parallel, as appears in Fig. 4 labeled as ‘‘control code generation” and ‘‘control model generation”.

For developing the plant model, the concept of the ‘‘Multi-Model Generator” introduced in Section 3.2 is extended to include views containing simulation and verification models based on physical modeling principles, which can be used to interact with control algorithm design and analysis tools. Linearization and model reduction methods ensure that the order of the model is suitable for the control design algorithms. The use of a knowledge-based engineering platform allows the software to reason about the interaction of components from various domains.

The generated control software can be verified using non-linear, high-fidelity models incorporating finite-element analysis results. For this, the formal representation of the requirements specified at the start of the design process is used. Once the software code has been generated and tested on virtual models, it can be tested on real prototypes. Recent benchmarks show that this has been an advantageous practice for the top-level mechatronic product developing companies [5].

In order to support and eventually automate the aforementioned tools and processes, and to collect, structure and communicate the generated information and data, it is required to create a common understanding of related concepts in the associated design domains. For this, it is necessary to have a formal definition of the different types of models supported by the framework, the element primitives and domains that are used to build the model, and a vocabulary of the domain-specific languages to give meaning to the abstractions of the primitives in the model [80]. These three elements form the basis for an ontology: a formal representation of a set of concepts within a domain and the relationships between those concepts. The developed ontology will be the foundation of the knowledge bases on which the (custom) tools and processes are built.

5. Conclusions

The design of integrated mechatronic systems requires a paradigm shift towards cooperation between design teams/paradigms, paying special attention to the early phases of design. To obtain tighter integration, the design of mechatronic systems demands a holistic approach that considers interactions and interrelations among design domains. Tools to support such an approach are necessary and, at the moment, scarce.

The authors have identified a set of challenges related to the design of mechatronic systems. These challenges are related to the integration of tools, models, and human actors in the design process, the lack of multi-disciplinary verification, and the lack of automation in control software development. The review shows that current methods and tools attacking these challenges focus on specific points and that developed implementations are not available. Model and data sharing is a key issue to progress towards an overall solution. Furthermore, formalization of architecture, function descriptions, and requirements needs to be addressed.

Regarding the efforts to overcome the identified challenges, industry tends to focus on tool-level integration, while academia focuses on underlying integration methods. Methods proposed by academia are hard to implement due to the abstract system descriptions, but have a promising future.

To address the challenges, some basic concepts for an integration framework supporting mechatronics design are proposed.

The framework contains both existing and to be developed tools, with which a multi-disciplinary design team can automatically generate and verify control software. Based on the FBS modeling scheme, the designer can create a requirements-specifications model that gives a high-level overview of the system under development. By integrating existing domain-specific design and analysis tools, the model data can be used to build simulation and verification models. The application of high-level function models as a base to the framework to support communication through tool and model integration is a promising approach.

The proposed framework has many similarities with the CORE tool [47]. The fundamental difference is that the proposed framework aims at the integration of software tools used by the designers for detailed design. To support this, the main mechanism will be an information model that links the attributes present in the system components and in the phenomena (specified in the domain-specific models) that rule their behavior. Another fundamental difference is that the proposed framework leaves analysis, verification, and simulation to the specialized tools that can be integrated instead of dealing with such computations directly as in the case of CORE.

A high degree of automation can be obtained by applying knowledge-based techniques, to carry out non-creative and repetitive tasks, resulting in a decrease in development time and development costs and an increase in reliability. Various methods and tools are supported by an automated information manager that enables the integration of models using the central, high-level system model.

Acknowledgments

The authors gratefully acknowledge the support of the Dutch Innovation Oriented Research Program 'Integrated Product Creation and Realization (IOP-IPCR)' of the Dutch Ministry of Economic Affairs.

References

[1] Pahl G, Beitz W, Feldhusen J, Grote KH. Engineering design: a systematic approach, 3rd ed.. London (UK): Springer London Limited; 2007.
[2] Wang L, Shen W, Xie H, Neelamkavil J, Pardasani A. Collaborative conceptual design – state of the art and future trends. Comput-Aid Des 2002;34:981–96.
[3] Tomizuka M. Mechatronics: from the 20th to the 21st century. Control Eng Practice 2002;10:877–86.
[4] Wikander J, Törngren M, Hanson M. The science and education of mechatronics engineering. IEEE Robot Autom Mag 2001;8(2):20–6.
[5] Boucher M, Houlihan D. System design: new product development for mechatronics. Boston (MA, USA): Aberdeen Group; 2008.
[6] Schöner HP. Automotive mechatronics. Control Engineering Practice 2004;12:1343–51.
[7] Craig K. Mechatronic system design. ASME Newslett 2009. <http://files.asme.org/asmeorg/NewsPublicPolicy/Newsletters/METoday/Articles/17845.pdf>.
[8] Ziegler JG, Nichols NB. Process lags in automatic control circuits. Trans ASME 1943;65:433–44.
[9] Perrin K. Digital prototyping in mechatronic design. Proj Mech 2009. <http://www.projectmechatronics.com/2009/07/13/digital-prototyping-in-mechatronic-design/> [website].
[10] Mathur N. Mechatronics – five design challenges and solutions for machine builders. Instrum Newslett 2007;19(2):6–7. <http://zone.ni.com/devzone/cda/pub/p/id/145>.
[11] Jackson CK. The mechatronic system design benchmark report. Boston (MA, USA): Aberdeen Group; 2006.
[12] Shapiro J. Mechatronics design faces two challenges – and two solutions. Electron Des 2008. <http://electronicdesign.com/Articles/Index.cfm?AD=1&ArticleID=18068> [website].
[13] Blessing LTM, Chakrabarti A. DRM, a design research methodology. London (UK): Springer-Verlag; 2009.
[14] Rzevski G. On conceptual design of intelligent mechatronic systems. Mechatronics 2003;13:1029–44.
[15] Sohlenius G. Concurrent Engineering. CIRP Annals 1992;41(2):645–55.
[16] Martin JN. Systems engineering guidebook – a process for developing systems and products. Boca Raton (FL, USA): CRC Press; 1997.
[17] QFD Institute. QFD Institute home page. <http://www.qfdi.org/>.
[18] Knowledge Based Systems Inc. IDEF Family of Methods website. <http://www.idef.com/>.
[19] Karnopp DC, Margolis DL, Rosenberg RC. System dynamics: modeling and simulation of mechatronic systems, 4th ed.. New York (NY, USA): Wiley; 2006.
[20] Cutkosky MR, Engelmore RS, Fikes RE, Genereseth MR, Gruber TR, Mark WS, et al. PACT: an experiment in integrating concurrent engineering systems. Computer 1993;26(1):28–37.
[21] Dolk DR, Kotterman JE. Model integration and theory of models. Decis Support Syst 1993;9(1):51–63.
[22] Jackson CK. Simulation driven design benchmark report. Boston (MA, USA): Aberdeen Group; 2006.
[23] Ramos-Hernandez DN, Fleming PJ, Bass JM. A novel object-oriented environment for distributed process control systems. Control Eng Pract 2005;13:213–30.
[24] Object Management Group. Unified modeling language, V2.2; 2009. <http://www.omg.org/spec/UML/2.2/>.
[25] Heck B, Wills L, Vachtevanos G. Software technology for implementing reusable, distributed control systems. IEEE Control Sys Mag 2003;23(1):21–35.
[26] Umeda Y, Ishii M, Yoshioka M, Tomiyama T. Supporting conceptual design based on the function-behavior-state modeler. AI EDAM 1996;10(4):275–88.
[27] Chandrasekaran B. Functional representation: a brief historical perspective. Appl Artif Intel 1994;8:173–97.
[28] Bracewell R, Sharpe J. Functional descriptions used in computer support for qualitative scheme generation – "Schemebuilder". AI EDAM J – Special Issue: Represent Function Des 1996;10:333–46.
[29] Hunt J. MACE: a system for the construction of functional models using case-based reasoning. Expert Syst Appl 1995;9(3):347–60.
[30] Barr A, Cohen PR. The handbook of artificial intelligence. vol. 4. Los Altos, CA, USA: William Kaufmann, Inc.; 1989 [chapter 21].
[31] Erden MS, Komoto H, Van Beek TJ, D'amelio V, Echavarria E, Tomiyama T. A review of function modeling: approaches and applications. Artificial intelligence for engineering design. Anal Manuf 2008;22(2):147–69.
[32] Tomiyama T, Umeda Y, Ishii M, Yoshioka M, Kirayama T. Knowledge systematization for a knowledge intensive engineering framework. In: Tomiyama T, Mantyla M, Finger S, editors. Knowledge intensive CAD, vol. 1. Chapman & Hall; 1996. p. 55–80.
[33] European Cooperation for Space Standardization. Space engineering – functional analysis (E-10-05A); 1999. <http://esapub.esrin.esa.it/pss/ecss-ct05.htm>.
[34] Stone R, Wood K. Development of a functional basis for design. ASME J Mech Des 2000;122(4):359–70.
[35] National Institute of Standards and Technology. Integration definition for function modeling (IDEF0); 1993. <http://www.idef.com/pdf/idef0.pdf>.
[36] Wood W, Dong H, Dym C. Integrating functional synthesis. AI EDAM 2004;19(3):183–200.
[37] Bonnema GM. FunKey architecting – an integrated approach to system architecting using functions, key drivers and system budgets. PhD thesis. University of Twente. Enschede, The Netherlands; 2008.
[38] Muller GJ. CAFCR: a multi-view method for embedded systems architecting, PhD thesis. Delft University of Technology. Delft, The Netherlands; 2004.
[39] Stevens R, Brook P, Jackson. System engineering: coping with complexity. Europe: Prentice Hall; 1998.
[40] Boehm B. A spiral model of software development and enhancement. ACM SIGSOFT Software Eng Notes 1986;11(4):14–24.
[41] Suh NP. The principles of design. Oxford (UK): Oxford University Press; 1990.
[42] Mcmahon CA, Caldwell NHM, Darlington MJ, Culley SJ, Giess MD, Clarkson PJ. The development of a set of principles for the through-life management of engineering information; 2009. <http://www.bath.ac.uk/idmrc/themes/projects/kim/kim40rep007mjd10.doc>.
[43] Citherlet S, Clarke JA, Hand J. Integration in building physics simulations. Energy Build 2001;33:451–61.
[44] VRS ROPAX. Virtual reality ship systems project webpage. <http://www.vrs-project.com/index.phtml>.
[45] Yoshioka M, Sekiya T, Tomiyama T. An integrated design object modeling environment – pluggable metamodel mechanism –. Turk J Electr Eng Comput Sci 2001;9(1):43–62.
[46] Yoshioka M, Umeda Y, Takeda H, Shimomura Y, Nomaguchi Y, Tomiyama T. Physical concept ontology for the knowledge intensive engineering framework. Adv Eng Infor 2004;18(2):69–127.
[47] Vitech corporation. CORE software website. <http://www.vitechcorp.com/products/Index.html>.
[48] Childers SR, Long JE. A concurrent methodology for the system engineering design process. Unpublished green paper; 1994. <http://www.vitechcorp.com/support/papers.php>.
[49] Peak RS, Burkhart RM, Friedenthal SA, Wilson MW, Bajaj M, Kim I. Simulation-based design using SysML part 1: a parametrics primer. In: Proceedings of INCOSE international symposium, San Diego, CA, USA; 2007.
[50] Peak RS, Burkhart RM, Friedenthal SA, Wilson MW, Bajaj M, Kim I. Simulation-based design using SysML: celebrating diversity by example. In: Proceedings of INCOSE international symposium, San Diego, CA, USA; 2007.
[51] Object Management Group. OMG systems modeling language, V1.0; 2001. <http://www.omg.org/cgi-bin/apps/doc?formal/07-09-01.pdf>.
[52] Paredis C, Diaz-Calderon A, Sinha R, Khosla PK. Composable models for simulation-based design. Eng Comput 2001;17:112–28.
[53] La Rocca G, Van Tooren MJL. Enabling distributed multi-disciplinary design of complex products: a knowledge-based engineering approach. J Des Res 2007;5(3):333–52.

[54] Berends JPTJ, van Tooren MJL, Schut EJ. Design and implementation of a new generation multi-agent task environment framework. In: 49th AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics, and materials conference, 4th AIAA multidisciplinary design optimization specialist conference. Schaumburg, IL, USA; 2008.
[55] The Modelica Association. Modelica and the modelica association; 2008. <http://www.modelica.org>.
[56] The MathWorks. Simscape; 2009. <http://www.mathworks.com/products/simscape/?s_cid=HP_FP_SL_Simscape>.
[57] Dynasim AB. Dymola–dynamic modeling laboratory; 2008. <http://www.dynasim.se/index.htm>.
[58] Controllab Products B.V. 20-sim. <http://www.20sim.com>.
[59] van Amerongen J. Mechatronic design. Journal of Mechatronics 2003;13(10):1046–166.
[60] van Amerongen J, Breedveld P. Modeling of physical systems for the design and control of mechatronic systems. Ann Rev Control 2003;27:87–117.
[61] Ferretti G, Magnani GA, Rocco P. Virtual prototyping of mechatronic systems. Ann Rev Control 2004;24:192–206.
[62] Process Systems Enterprice Limited. gPROMS Advanced Process Modeling and Process Simulation. <http://www.psenterprise.com/gproms/index.html>.
[63] Synopsys. Saber mixed-signal, mixed-technology simulation. <http://www.synopsys.com/Tools/SLD/MECHATRONICS/Saber/Pages/default.aspx>.
[64] Mosterman PJ. HyBrSim – A modeling and simulation environment for hybrid bond graphs. <http://moncs.cs.mcgill.ca/people/mosterman/papers/jsce01/p.pdf>.
[65] Technical University of Berlin. SMILE – the simulation environment for scientific computing. <http://www.smilenet.de>.
[66] Voskuijl M, La Rocca G, Dircken F. Controllability of blended wing body aircraft. In: Proceedings ICAS of the international council of the aeronautic sciences including the 8th AIAA Aviation Technology, Integrated and Operations Conference. Edinburg, UK; 2008.
[67] Toom A, Naks T, Pantel M, Gandriau M, Indrawati. Gene-Auto: an automatic code generator for a safe subset of Simulink/Stateflow and Scicos. In: 4th European congress on embedded real time software. Toulouse, France; 2008.
[68] Thompson HA, Ramos-Hernandez DN, Fu J, Jiang L, Choi I, Cartledge K, et al. A flexible environment for rapid prototyping and analysis distributed real-time safety-critical systems. Control Eng Pract 2007;15:77–94.
[69] Kelly S, Tolvanen J-P. Domain-specific modeling: enabling full code generation. Hoboken (NJ, USA): Wiley-IEEE Computer Society Press; 2008.
[70] Harel D. From play-in scenarios to code: an achievable dream. IEEE Comput 2001;34(1):53–60.
[71] Rash JL, Hinchey MG, Rouff CA, Gracanin D, Erickson J. A requirements-based programming approach to developing a NASA autonomous ground control system. Artif Intel Rev 2006;25(4):285–97.
[72] Ferrarini L, Carpanzano E. A structured methodology for the design and implementation of control and supervision systems for robotic applications. IEEE J Control Syst Technol 2002;10(2):272–9.
[73] Sakao T, Umeda Y, Tomiyama T, Shimomura Y. Generation of sequence-control programs from design information. IEEE Expert 1997:12.
[74] Umeda Y, Tomiyama T, Yoshikawa H, Sakao T, Shimomura Y, Tanigawa S. Mita Industrial Co., Ltd., assignee. Method of automatically creating control sequence software and apparatus therefore. US patent 194,064, Feb 9; 1994.
[75] Browning TR. The many views of a process: toward a process architecture framework for product development processes. Syst Eng 2009;12(1):69–90.
[76] Umeda Y, Tomiyama T. FBS modeling: modeling scheme of function for conceptual design. In: Workshop on qualitative reasoning about physical systems. Amsterdam, The Netherlands; 1995, p. 271–8.
[77] Tomiyama T, Umeda Y. A CAD for functional design. Ann CIRP93 1993;42(1):143–6.
[78] Institute of Electrical and Electronics Engineers Standards Association. IEEE Std 1471–2000: recommended Practice for Architectural Description of Software-intensive Systems; 2000.
[79] Muller G. System architecting. Eindhoven (The Netherlands): Embedded Systems Institute; 2009.
[80] Guizzardi G. On ontology ontologies conceptualizations modeling languages and (meta) models. In: Vasilecas O, Edler J, Caplinskas A, editors. Frontiers in artificial intelligence and applications, databases and information systems IV. Amsterdam (The Netherlands): IOS Press; 2007.