Proc. ASME Dyn. Sys. & Control Div. '95 (San Francisco, USA),
DSC-57-1, T.E. Alberts (ed.), ASME, New York, pp. 421-430, 1995.

MAXIMIZING IMPACT OF AUTOMATION ON MODELING AND DESIGN

Arno P.J. Breunese
Theo J.A. de Vries
Job van Amerongen
Peter C. Breedveld
Department of Electrical Engineering
University of Twente
Enschede
Netherlands

ABSTRACT

Modeling engineering systems requires creativity and ingenuity, and therefore involves steps that are hardly suitable for automation. However, these steps are complemented by a multitude of routine tasks. Automation of these routine tasks, if tailored to complement non-automated tasks, allows the modeler to focus on the creative and innovative aspects of modeling and design.

To this end, three important principles for automated modeling have been formulated and incorporated in the MAX (Modeling and Analysis eXpert) system:

- The *polymorphic modeling* concept provides hierarchical models, in which each subsystem consists of a *type*, defining essential properties, and a *specification*, defining incidental properties. Subsystem types can be organized hierarchically in a library, and each type can have multiple specifications.
- Information about a model is processed using *multiple formulations*. Different formulations of one model can be used simultaneously. The task at hand determines which formulation is appropriate for inspection and manipulation.
- By *embedding equations in networks* in an intuitively appealing way, MAX provides the same kind of support for network models and equation models.

The utility of these concepts is demonstrated in a case study involving a fourth order servo system.

This work was supported in part by Unilever Research Laboratory, Vlaardingen, Netherlands, and by the CEC ESPRIT–III project P6521 'OLMECO' (Open Library for Models of MEchatronic COmponents). In the latter project, the Control Laboratory of the Electrical Engineering Department of the University of Twente collaborates with PSA Peugeot Citroën (France), BIM (Belgium), Fagor (Spain), Ikerlan (Spain), Imagine (France) and ECN (Netherlands).

1 INTRODUCTION

Products that were traditionally purely mechanical are becoming more and more 'intelligent', and yet their cost does not increase as much as their performance. The recent development of a design attitude known as *mechatronics* is an important factor in this trend. The combined application of new developments in control engineering, electronics, computer science and other disciplines leads to improved performance for a wide spectrum of products ranging from durable equipment like assembly machines and automatic guided vehicles to consumer products like CD-players and video cameras.

The 'added value' does not originate in the mere fact that the systems become multidisciplinary. The power of mechatronics is not so much in the decomposition of a system into single-discipline subsystems, but rather in the emphasis on the interdependence between the subsystems. Functional interaction and spatial integration are exploited to obtain systems that are 'more' (in terms of performance and quality) than the sum of their subsystems (Buur, 1990).

Communication about the system under design, or more precisely, about models thereof, is used to realize the cooperation between designers, and to coordinate their work (Buur and Andreasen, 1989). An obstacle in the communication is that the members of a mechatronic design team do not share a common 'language', because they have different backgrounds and responsibilities. Misunderstandings due to the mismatch in concepts and terminology are likely to affect the result of the design effort in a negative way. Consider for instance the design of an autofocus lens. A different choice for the material of a lens to obtain better optical properties implies a different mass for a mechanical engineer, and a different time constant for a control engineer. A message from the optical specialist that the lens material has changed (if given at all) will probably not trigger the control engineer. These interdependencies and their

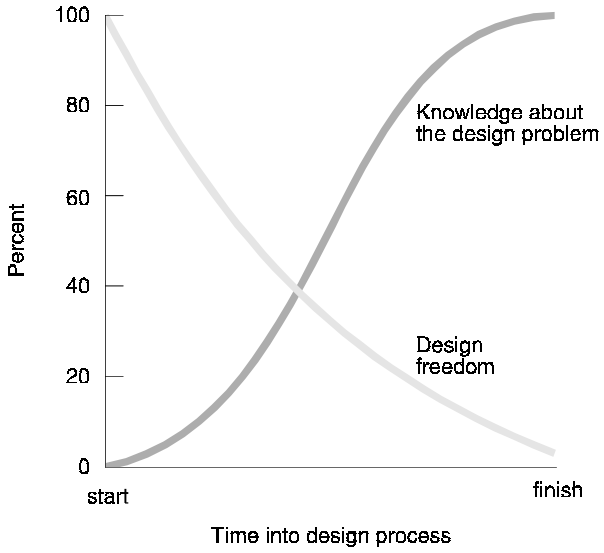

Figure 1: The design process paradox (adapted from Ullman (1992), figure 1.8)

relevance for the design are not taken into account properly if the design object is modeled by means of independent, single-discipline models. Tools are required in order to maintain an integrated and consistent set of models of the design object. Modeling systems have been described for this purpose that formulate models of the design object in terms of a single multi-domain modeling language, often bond graphs (Rosencode, 1989; Broenink et al., 1992; Bidard and Favret, 1993; Sharpe and Bracewell, 1993). These systems typically also support simulation, and have shown to be useful for analysis purposes. However, the use of such systems requires the designer to describe the design object in unnatural terms, in a language that he would otherwise probably not have used during synthesis. A more effective approach is to allow each member of the design team to work on the model of the design object in terms that are relevant and familiar to him, and to provide translation and coordination of the different models. This significantly enhances communication and mutual understanding, as no one is forced to sacrifice expressiveness. This type of modeling support is hardly possible without automated tools. The amount of information processed, and the knowledge required to do the information processing, are too involved to be carried out manually by designers.

As a second consequence of the focus on interaction and integration between subsystems, the mechatronic design approach requires parallel activity for the design of the subsystems. The traditional, sequential approach (which typically designed the mechanical structure first, then added the sensors and actuators, and finally constructed a control system) is unable to provide the interaction and integration which is typical for mechatronic systems. The challenge that results from a parallel approach is that the design of a subsystem is no longer based on complete knowledge of other subsystems and on well-defined requirements. Ullman (1992) describes this '*design process paradox*'; the understanding required to find a good solution can almost only be gained in the course of the design process. However, the knowledge may be acquired at a stage when the design is already too constrained to be modified according to the latest insights. This is illustrated in figure 1.

Models can be exploited to shift the knowledge curve in figure 1 to the left, by enabling rapid exchange of information about the problem and the proposed solutions in an early stage, i.e. when it still matters. For optimal results, structured modeling should be part of the design process from the very beginning. In the conceptual design phase, important decisions are taken that determine a large number of properties of the final product. The availability of information in that stage is important to avoid that side effects of apparently minor decisions lead to failure of the final product. Automated support in the form of knowledge bases storing models that capture experiences from similar projects is of great help (Konda et al., 1992). A major problem in this is that knowledge bases of a size needed in practice become hard to use and unmanageable due to a lack of structure.

When models are introduced early in the design process, they are subject to considerable evolution. They form the framework in which design refinements take place (Hoover et al., 1991). To take full advantage of this, automated modeling tools should allow the models to grow along with the design activities. Automated modeling tools should support the life cycle of the model, and provide meaningful operations on the model for each stage in the design. In this context, it is most important to be able to start with fairly simple models, and add details as they become available. In the automated modeling and AI community, considerable attention has been given to the 'hard' modeling problems of automatic abstraction (from detailed models to simpler ones; e.g. Rinderle and Subramaniam, 1991; Wilson and Stein, 1992; Weld, 1992; from concrete models to abstract ones; e.g. Amsterdam, 1992), of automatic model synthesis (find a simple model with a given behavior; Ulrich and Seering, 1989; Redfield and Krishnan, 1992; Malmqvist, 1993) and of automatic generation of behavior (equations generation and simulation). However, automated support for refinement of models, i.e., gradual addition and alteration of model properties (e.g. shifting the system boundary, considering parasitic behavior, etc.) has not been given attention, whereas it is the most common step during design.

From the remarks above, we may conclude that models in a wide sense are important in the context of mechatronic design. Automated support for modeling can provide a major contribution, but it can not completely replace human skills and creativity. Previous generations of modeling tools are not always suitable for application in mechatronic design, due to the special characteristics of design in general and mechatronics in particular. In the remainder of this paper, we describe three extensions to automated modeling that help to overcome the problems we have identified in this section:

– To enhance communication between disciplines, an automated modeling tool should provide means to work with *multiple model formulations*, while maintaining consistency and coherence between formulations. Section 2 presents an architecture for support of multiple formulations.

– Section 3 deals with *polymorphic modeling*, which is our answer to the requirement that tools for automated modeling should allow to update models as we learn about a design problem. Furthermore, polymorphic modeling provides an elegant way to integrate model libraries into automated modeling tools.
– To extend the life cycle of models into the stage where quantitative analyses are performed, we suggest to *embed equations in network models*. Section 4 discusses how proper integration and support can be achieved.

The proposed concepts are illustrated by the modeling of a small, yet typical mechatronic system. This modeling is performed using the MAX system, an automated modeling tool for mechatronic systems. Conclusions are summarized in section 5.

2 MULTIPLE MODEL FORMULATIONS

The representation of a model determines the information that can be expressed, and how accessible this information is with respect to inspection and interpretation. The question which representation to use does not have a unique answer. An iconic diagram is suitable for giving an overview of the decomposition into subsystems, whereas a block diagram is a more useful representation when selecting a control scheme. In a context with different people, with different backgrounds, and with different interests, it should be possible to simultaneously formulate one model in multiple 'languages'. It should also be possible to manipulate the model in any of the formulations. This concept is called *multiple model formulations*.

Automation is crucial for the application of multiple model formulations, because the concept can only then be fruitful if the formulations are consistent. This involves a considerable maintenance effort, which cannot be performed by hand for non-trivial cases. In figure 2, we present an architecture that structures the operation of a system with multiple model formulations. The lowest layer in the model is the *core model*. The core model is a repository of all information required by the formulations supported by the system, and it acts as a coordinating agent between the various formulations. The core model is linked to a number of formulations. Each formulation consists of two layers. The layer closest to the core model is the language specific *description*. The descriptions are coupled to the core model by means of bi-directional transformations. A language specific description is a formal description (typically in textual form) of those parts of the core model that are of importance for the formulation at hand. The description is linked to a *representation* by a visualization process. The representation can be textual (e.g. equation models) or network-based (e.g. block diagrams or bond graphs). The editors of the modeling tool allow the user to perform manipulations that have a directly observable effect in the representation used by the editor.

When a model is manipulated, each layer in the formulation propagates the changes from the next higher layer. Upon reception of a change, the core model will notify all other formulations of the change, and the propagation takes place in the reverse direction.

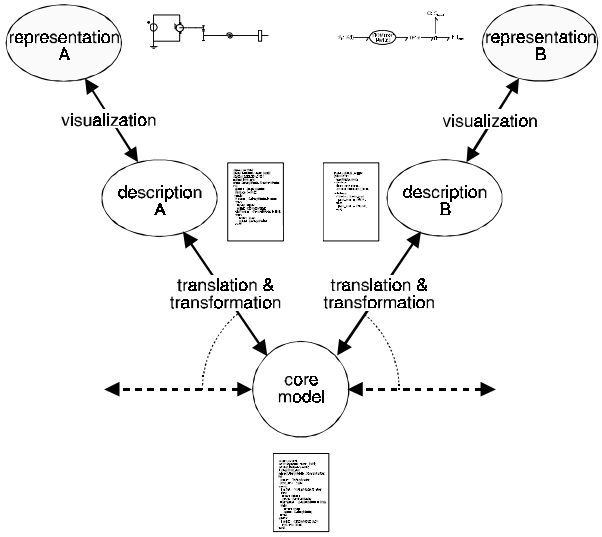

Figure 2: Architecture of a system featuring multiple model formulations

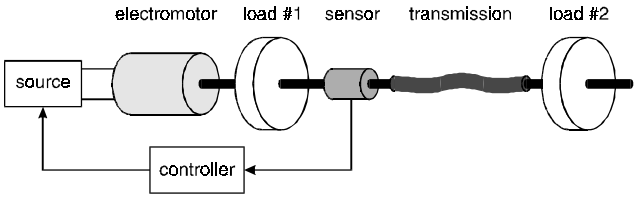

Figure 3: Initial proposal for the practical setup

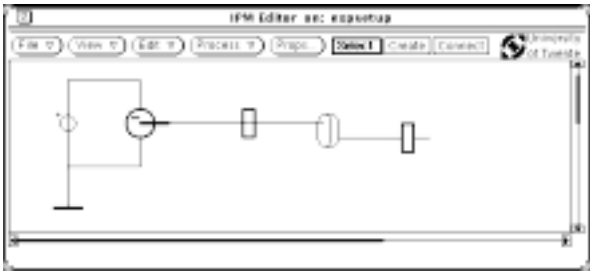

Figure 4: Initial model in iconic diagram formulation

The feasibility of this architecture has been shown for iconic diagrams and bond graphs (De Vries, 1994). The architecture was specifically designed to be scaleable, i.e. to allow the addition of extra formulations without disproportional effort.

We illustrate the use of multiple model formulations by showing the entry of the initial model of our case study into the MAX modeling system. The system to be modeled is a setup for gaining practical experience in the design and implementation of controllers for mechatronic systems. This electro-mechanical device is to be used in the education of third year master students at the Control Laboratory of the University of Twente. The initial proposal for the system is sketched in figure 3.

Expected dynamic features that make this setup attractive are that it may have variable stiffness and load, that the system

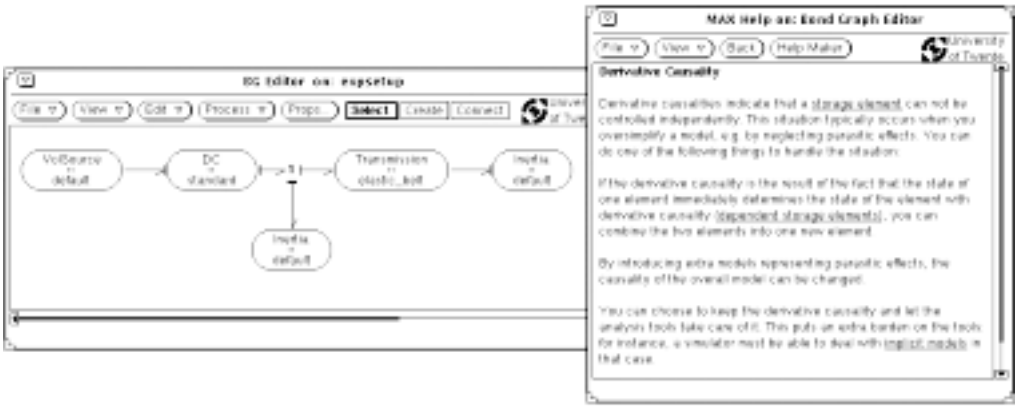

Figure 5: Bond graph of the initial proposal

exhibits non-linear behavior and that there will be a clearly observable difference between controlled and non-controlled performance for various types of controllers. These aspects will be investigated in the case study.

As shown in figure 4, the model is entered into the IPM Editor. The iconic diagram formulation is preferred for initial model entry, because it closely resembles the sketch drawn in figure 3. The control loop is omitted in the first version, because we first want to obtain an impression of the open loop behavior of the system. The editor used for the entry of the model is more than a simple drawing tool. The iconic diagram editor for instance refuses to make connections between incompatible terminals: it is impossible to connect the (electrical) terminals of the source to the (rotation) terminals of a load. If a connection causes a 'short circuit' this is also reported.

Furthermore, the editors provide formulation-specific functions. For instance, the Bond Graph Editor provided in the MAX system can export models to the bond graph-oriented CAMAS modeling and simulation environment (Broenink et al., 1992), and it can also import models from the CAMAS environment. The MAX Bond Graph Editor also features powerful algorithms for causality assignment. This is shown in figure 5, where causality is assigned to the bond graph formulation of the initial proposal.

The structure of the bond graph is automatically generated from the original IPM model. The causality assignment shows that the left load does not represent an independent state in the model. The causal stroke is emphasized by the editor to indicate that a situation requiring further attention has been detected. The help function of the MAX system explains that a derivative causality is a typical symptom of an oversimplified model. This message is not completely unexpected.

3 POLYMORPHIC MODELING

Ward (1989) states that a design object is not a collection of descriptions of single artifacts, but a collection of *sets of equivalent artifacts* instead. The same is true for the model used to describe the design object. Reasoning in terms of sets is an excellent way to handle the large 'solution space' that occurs in the early stages of modeling or design. Classes of similar solutions can be treated as if they were a single solution, and evaluation of such classes is a powerful instrument to select or reject solution classes. Specific features of individual elements in the classes are left unspecified. At the time these features become relevant, the sets can subdivided on the basis of additional features. This process continues until a small number of fully detailed models are obtained.

In order to have automated support for this approach, a modeling tool should provide means to describe model fragments as specific instances of a class of models (i.e. as elements of a set of artifacts). To do so, it must be possible to distinguish essential properties of a subsystem from incidental properties. The essential properties are those properties of a subsystem that are required to properly classify the subsystem. The incidental properties are those properties that can be used to discriminate between elements within a set, but that are not relevant for the classification. If we consider the example of a DC motor, the fact that it converts DC electrical energy into mechanical energy is an essential property, whereas the specific parameters of the motor are incidental properties; they are not needed for a proper classification.

In computer science the term *modularization* is used for a comparable separation. A good example of modularization can be found in the Modula programming language (Wirth, 1982). In a modeling context, modularization means that a model is divided into two parts: a *type* that defines the essential properties, and a *specification* that defines the incidental properties. The modularization of subsystem definitions is used to allow gradual refinement of the model in two ways:

– In a model library, types can be organized in a kind-of hierarchy using *subtyping*. Each type is specified as a special kind of its supertype, inheriting all the essential properties of the supertype and adding additional properties. By changing the type of a submodel to the supertype (generalization) or to one of the subtypes (specialization), the model can be refined step by step. Subtyping is also an important means to obtain a well-organized library; the possibility for incremental definition of the type properties leads to a coherent, practical structure.

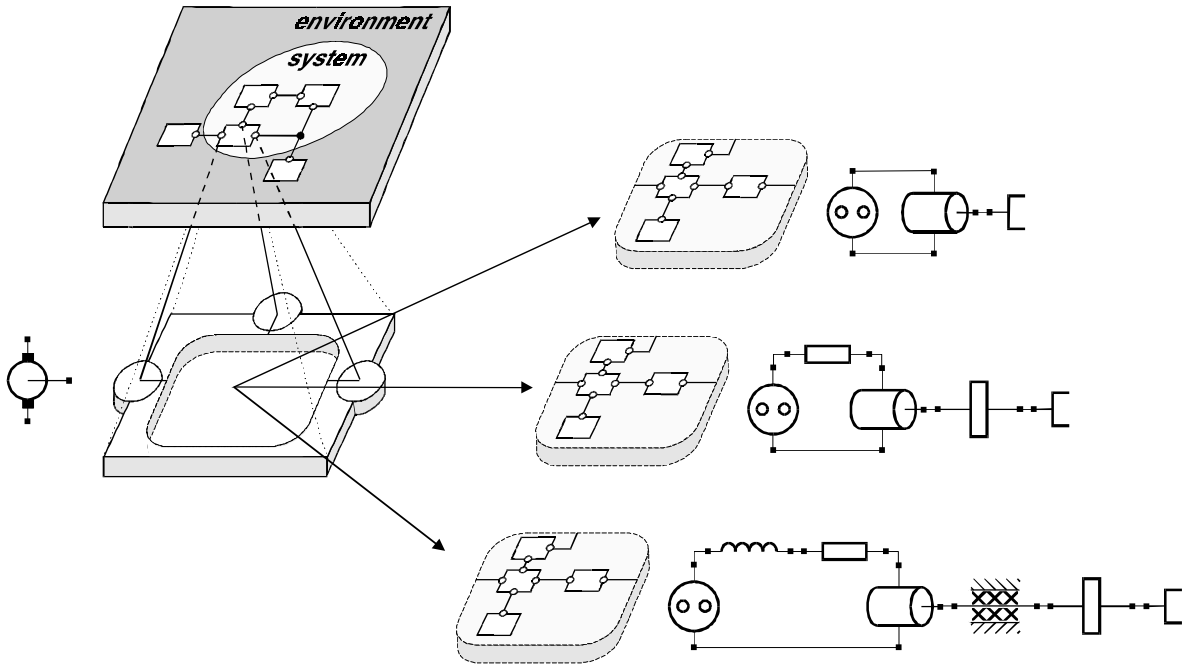

Figure 6: Polymorphic modeling: model in terms of modular, classified components

– One type may have multiple specifications. By choosing another specification for a submodel of a particular type, incidental properties of the model can be easily modified, while keeping intact the information defined by the type. In fact, the subsystem type becomes *polymorphic*. This is illustrated in figure 6. Model polymorphism has been exploited by other modeling and design support systems as well (Malmqvist, 1993; Sharpe and Bracewell, 1993; Stein and Louca, 1995).

The combination of subtyping and modularization in a model building system is called *polymorphic modeling* (De Vries et al., 1993; De Vries, 1994). The use of these principles results in a hierarchical subsystem library that has a well defined, conceptually clear structure. Polymorphic modeling facilitates the structured manipulation of a model, for instance creation of alternative solutions and variations in the level of detail provided in models. Polymorphic modeling also supports hierarchical models. The specification of a type can be built from submodels of various types.

Recent projects like the OLMECO project (Top et al., 1995) show industrial and academic interest for libraries for mechatronic models. In order to quickly build complete models from predefined components, the concept of *realizations* has been introduced. Realizations are specifications for which a (part of the) model hierarchy has been specified. We foresee that component suppliers will more and more provide models with their components, as is already common in the electronics industry. Realizations are ideally suited for this purpose.

In MAX, modularization and subtyping are used to set up a hierarchical library of types, with realizations for most types. The library is accessible through the Library Browser. This browser shows a taxonomy of model types. Four taxonomies are currently available:

– *Signal blocks*: Models of information processing elements, including controllers. These models are typically encountered in block diagrams, and block diagram-like parts of models in other formulations.
– *Elements*: Models of basic physical phenomena, as recognized in physical systems theory. The taxonomy is based on the classification of models used in the bond graph community. These models are typically used in bond graph models, and (given a particular domain) as idealized components in iconic diagrams.
– *Components*: Models of physical devices, using the main function of the component as a criterion for classification. A large share of the library content can be found in this taxonomy.
– *Special*: Model parts that are specific for a certain formulation, including models for ports. This hierarchy is rather static; it only changes when new formulations or physical domains are added.

The MAX Library Browser is shown in figure 7. The top left pane in the window shows the taxonomy of electrical motors, indicating how the taxonomy in the MAX system resembles the component taxonomy that is formulated by domain experts.

In our case study, we exploit the Library Browser and the information contained therein for the selection of an alternative for the flexible shaft, because it appears that a simple flexible shaft can not easily satisfy the requirement of varying stiffness. A promising alternative is the specification of the transmission that uses a translation spring and two transformers to realize the

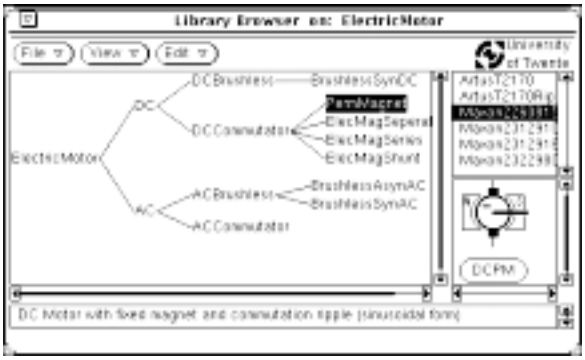

Figure 7: MAX Library Browser
Top left: hierarchically organized subsystem types
Top right: list of realizations of the selected type
Bottom right: available representations of the selected type
Bottom: comment for the selected type and realization

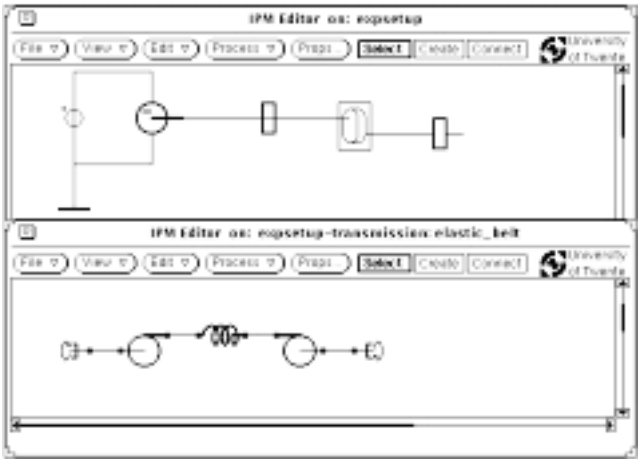

Figure 8: Alternative solution for the flexible transmission, obtained by selecting another specification

Figure 9: Iconic diagram representation of the electromotor: (a) generic DC motor (b) commutated DC motor (c) permanent-magnet DC motor

desired function. This specification is already available in the MAX library, and can be installed in the model by selecting the 'specify' option for the spring component in one of the graph editors. Subsequently, the 'explode' function is selected, and the editor now shows the newly installed specification (figure 8). The iconic diagram formulation is a good choice for the representation, because it hints at a possible implementation.

The new proposal can be realized as two pulleys coupled by means of an elastic belt. By providing pulleys with different radii or by varying the number of elastic belts, this solution satisfies the requirements for which the initial proposal failed. To make the model more realistic, the specification of the spring in this transmission model is changed to include damping. The default (linear) damping model can be used for a preliminary analysis. The model of the motor needs some refinement before proceeding with the analysis, as it has a large influence on the behavior of the system. The type of the initial model of the motor is that of a generic DC motor. This is reflected by the icon used for the representation of this type in the iconic diagram editor (figure 9a). The model of the motor is refined by repeatedly *specializing* the model type into one of the subtypes of the current type, thereby suggesting sensible options for the choice of the actual motor. This operation is supported by the MAX graphical editors. The first specialization leads us to the commutated DC motor. In the iconic diagram editor, this change is visualized by the change of the icon for the model. The icon is now the one shown in figure 9b. A further specialization leads to the permanent-magnet DC motor. Once more, the icon changes, this time to the one shown in figure 9c.

At this point, we find that the realization 'Maxon2260815' (as shown in figure 7) provides an adequate model of the device we want to use in the setup.

So far, the model of the setup has been an open loop model only. It is prepared to become a controlled process by adding sensors, and by changing the source into an amplifier. The initial model of the sensor is an ideal velocity transducer. If desired, a more detailed model can be specified later by refining the type and/or the realization of the sensor model.

The complete model now has signal ports: an input for the steering, and outputs for the measurements. When storing this model, MAX prompts the user to select an appropriate type for the model, thereby suggesting 'Process' as one of the options. This suggestion is made on the basis of the ports and attributes of the model.

Next the model 'ControlledSystem' (provided by the library as a template for rapid model construction) is opened, and the subsystem 'Process1' is specified as the realization we just made for our setup. The structure of the resulting model is shown in figure 10.

The above sequence of operations demonstrates that a lot of information can be added to the model with minimal effort. Because the manipulations are highly structured, and checked by the system, there is a relatively high assurance that we have obtained a high quality model.

4 EMBEDDING EQUATIONS IN NETWORKS

Equations are needed to describe knowledge of quantitative behavior of the smallest subsystems, which is in turn used to determine the performance of the system. The performance is typically expressed as a performance index, or set of goal parameters. In general, these goal parameters are not directly available from the model (because they can not be influenced directly by design decisions). Settling time and overshoot are typical goal parameters for a controlled system. These parameters are in general determined by physical parameters (masses, stiffnesses) and controller parameters (gain factors and integration constants). For simple cases, an analytical expression may be available to derive the goal parameters from the

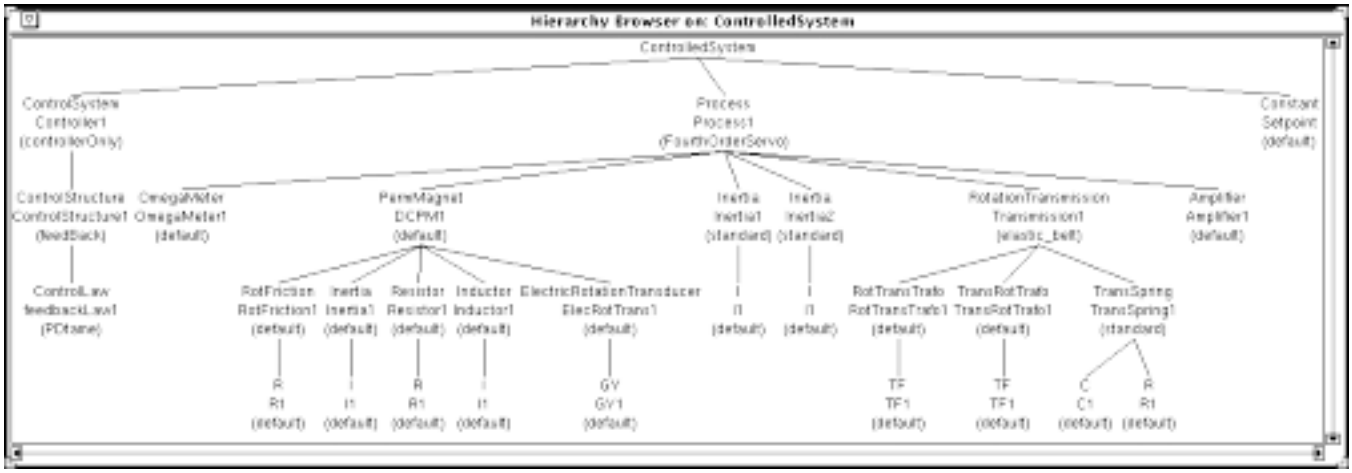

Figure 10: Model hierarchy of the controlled system

parameters available in the model. For more complex models, other means are required to obtain the goal parameters for a given set of model parameters. Simulation is a commonly used technique for this purpose.

Quantitative simulation cannot be performed without quantitative relations (equations) between variables in the model. Even if a tool seems to work without equations, there will always be a level in which equations are added. We advocate the use of an explicit link between equations and non-equation formulations. An explicit link emphasizes the role of equations, and makes the tool more open towards the addition of user-specified equations. In MAX, we use the property that network-type models are interconnected by means of ports (power ports and/or signal ports). This port can be used to connect to another network-type model, but it can also be used to connect to a set of variables in an equation model. This is illustrated in figure 11.

By decomposing the network structure to a sufficient depth, the individual equation sets can be small. The typical use of embedded equations is to specify a single (physical) process in less than five equations. The connection with the network provides a well-defined context for the equations. Together, this makes that the modeler can easily formulate correct equations, and quickly understand the meaning of equations made by others (e.g. those provided by the library).

For proper integration of network-type models and equations, the automated system must offer a comparable level of support and model analysis and verification for both types of models. For instance, if the tool reports unconnected elements in a network, it must also report variables and parameters that are not used in a set of equations.

There is an important difference between graph editors and equation editors: in graph editors, the presentation of the model to the user is relatively close to the representation in the computer's memory. Therefore, a graph editor can give immediate feedback during model editing. In an equation editor, the presentation of the model to the user is done in textual form. The transformation of the text to the representation in computer memory requires compilation. Hence, it is not possible to provide immediate feedback. Typically, feedback is provided after the user indicates that he is ready. This implies that the reports generated by an equation editor should be accurate in reporting the location and the nature of the errors.

The MAX Equation Editor follows these guidelines. The upper half of the Equation Editor window (see figure 13) contains information that can be derived from the type for which the equations are being specified. This information can not be edited immediately by the user (the Type Editor should be used instead for that purpose). The actual equations are specified in the lower half of the window. The equations are formulated in SIDOPS++, a language especially designed for the description of physical systems (Breunese, 1993). Models are specified in a-causal form in this language. The equations represent equality of two expressions, not a computer program or algorithm. Based on our view of automated modeling, the conversion from equations to simulation code is a task that should be performed by the computer.

When the user has finished typing the equations and gives the command to save the equations, the first step in the analysis is a compilation of the equations from the textual format into a data structure in the computer's memory that is more suitable for the actual analysis. In this phase, syntax errors are detected and reported. If this compilation is successful, it is by no means certain that the equations are valid and meaningful. For instance, the equation '1 = 2' is syntactically correct, but it is not valid. The MAX system checks a wide range of aspects of the equations. The simpler checks include a check that all identifiers (names of variables, etc.) are declared before use, and a check that the arithmetic type matches for operators and functions, whereas more advanced checks determine whether it is possible to compute all unknowns on the basis of the given equations, and what the causality restrictions of the equation set are. These causality restrictions are matched with the restrictions defined for the type selected for the equation set, and incompatibility is reported. A more detailed overview of the analysis of equation models can be found in (Breunese and Breedveld, 1995). If an equation model passes all checks, it is assumed to be correct and suitable for simulation or analysis.

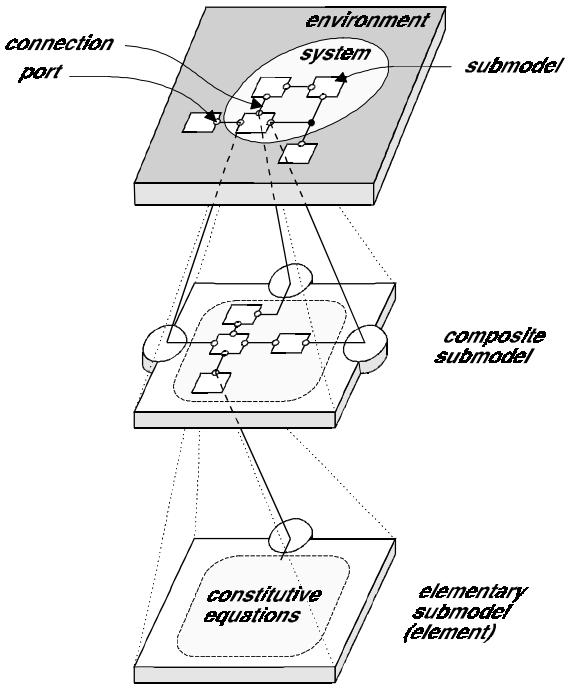

Figure 11: Equation models as the leaves of a hierarchical model tree

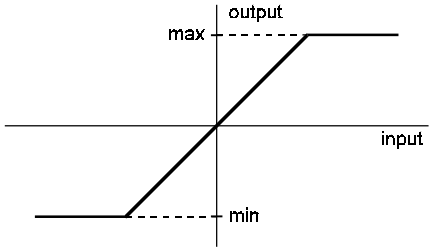

Figure 12: Characteristic for the motor amplifier

Figure 13: Equation Editor with the model for the amplifier

The Equation Editor is also used in our case study. We have observed that the motor amplifier is unable to provide the desired output voltage if the controller requests a large value. The characteristic of the amplifier is shown in figure 12.

In figure 13, we see how the initial model is formulated in the Equation Editor. Since we do not want to fix the bounds for the amplifier to a numerical value, we use the symbols *min* and *max*. The symbol *p.e* identifies the effort variable of port *p* defined by the type of the amplifier. If we attempt to store these equations in the library, the Equation Editor reports that the symbols *min* and *max* are not declared. After querying the user for the desired type (local variable, parameter, etc.) for the symbols, the Equation Editor gives the opportunity to specify additional information for the parameters. After the user has entered the information he chooses to provide, the Equation Editor will automatically generate correct code for the declaration of *min* and *max*. After the system has checked that the equations can indeed be used to describe an amplifier, the adjusted model is stored in the library. To install the newly created model into the overall model of our system, we can simply change the realization of the amplifier into 'clipping'.

To perform a simulation of the complete, controlled model, we load the model into the Bond Graph Editor, and export it to the CAMAS environment. The exported model consists of the model structure and the model parameters defined in MAX. CAMAS is then used to perform the actual simulation. The results of this simulation are shown in figure 14.

5 CONCLUSIONS

Modeling engineering systems involves tasks that are best performed by humans, and tasks that are suitable for automated support. Ideally, these tasks should complement each other. Presently, most modeling tools do not yet provide the level of support that is desirable, and worse, the support they offer is not tailored to practical use for multidisciplinary design.

In this paper, we describe the role of modeling in the context of mechatronic design. This leads to three proposals for automated modeling tools that do support the mechatronic design process:

– It should be possible to simultaneously formulate one model in multiple languages, in such a way that the model can be manipulated in any of the formulations. This concept is called *multiple model formulations*. We have devised an architecture that enables multiple model formulations and yet keeps different formulations of the model consistent and tractable.
– Automated modeling tools should support the evolution of a model over time. It is important that sets of alternatives can be considered effectively, and that increasing levels of detail of the model can be evaluated. As a solution, we propose modularization of submodel descriptions into a type (defining essential properties) and a specification (defining incidental properties), and the subtyping of submodel types, i.e. expressing a type as a specialization of a more general type. The combination of modularization and subtyping in model building is called *polymorphic modeling*.

Figure 14: Results of the simulation for the controlled model constructed in the case study

– Equations are needed for quantitative analysis of a model. By providing a tight link with the graphical formulations, consistency is improved, and the explicit context makes understanding easier. Equation editors should allow model entry in a user-friendly and intuitive fashion. However, the format should be formal enough to allow thorough checks.

The implementation of these three concepts is introduced by means of an example session with MAX. The example demonstrates the utility of the concepts in a mechatronic context. Multiple model formulations enable a convenient way to inspect the model, and the transformation between formulations provides additional insight in the properties of the model. The polymorphic modeling concept enables the evolution of the model when knowledge about the details of the design increases, and also allows for simple evaluation of alternative solutions. Finally, embedding equations in network descriptions helps to describe detailed (physical) behavior of model parts in a consistent way. The construction of simulation models is made easy by automated procedures for the derivations of the aggregated set of equations for a complete model.

REFERENCES

Amsterdam, J., 1992, "Automated modeling of physical systems", in (Falkenhainer and Stein, 1992), pp. 31–36.

Bidard, C., and Favret, F., 1993, "SCRIBT.MODELISATION, bond graph modelling core for technical systems CAD", *Proc. Int. Conf. Bond Graph Modeling and Simulation '93*, Granda, J.J., and Cellier, F.E., eds., SCS, San Diego, CA, pp. 179-184.

Breunese, A.P.J., 1993, "Preliminary design of SIDOPS++", Internal report 93R199, Control Laboratory, University of Twente, Enschede, Netherlands.

Breunese, A.P.J., and Breedveld, P.C., 1995, "Automatic derivation of causality restrictions and verification of bond graph submodels", accepted for publication in *J. Mathematical Modelling of Systems.*

Broenink, J.F., Bekkink, J.W., and Breedveld, P.C., 1992, "Multibond-graph version of the CAMAS modeling and simulation environment", *Bond graphs for engineers*, Breedveld, P.C., and Dauphin-Tanguy, G., eds., Elsevier, Amsterdam, Netherlands, pp. 253–262.

Buur, J., 1990, "A theoretical approach to mechatronics design", PhD thesis, Institute for Engineering Design, Technical University of Denmark, Lyngby, Denmark.

Buur, J., and Andreasen, M.M., 1989, "Design models in mechatronic product development", *Design Studies*, Vol. 10, pp. 19–34.

Falkenhainer, B., and Stein, J.L., eds., 1992, *Automated Modeling*, DSC-41, ASME, New York, NY.

Hoover, S.P., Rinderle, J.R., and Finger, S., 1991, "Models and abstractions in design", *Proc. Int. Conf. on Engineering Design ICED '91*, Heurista, Zurich, Switzerland.

Konda, S., Monarch, I., Sargent, P., and Subrahmaniam, E., 1992, "Shared memory in design: a unifying theme for research and practice", *Research in Engineering Design*, Vol. 4, pp. 23–42.

Malmqvist, J., 1993, "Computer aided conceptual design of energy transforming technical systems", *Proc. Int. Conf. Engineering Design '93*, Roozenburg, N.F.M., ed., Heurista, Zurich, Switzerland, pp. 1541–1550.

Rosencode Associates, 1989, "The ENPORT reference manual", Lansing, MI.

Sharpe, J.E.E., and Bracewell, R.H., 1993, "Application of bond graph methodology to concurrent conceptual design of interdisciplinary systems", *Proc. Int. Conf. on Systems, Man and Cybernetics*, P. Borne et al., eds., IEEE, Piscataway, NJ, Vol. 1, pp. 7–13.

Stein, J.L., and Louca, L.S., 1995, “A component-based modeling approach for system design: theory and implementation”, *Proc. Int. Conf. Bond Graph Modeling and Simulation ’95*, Granda, J.J., and Cellier, F.E., eds., SCS, San Diego, CA, pp. 109–115.

Redfield, R.C., and Krishnan, S., 1992, “Towards automated conceptual design of physical dynamic systems”, *J. Engineering Design*, Vol. 3, No. 3, pp. 187–204.

Rinderle, J.R., and Subramaniam, B. L., 1991, “Bond graph modeling and simplification to support design”, *Automated Modeling*, DSC-34, Stein, J.L., ed., ASME, New York, NY, pp. 45–68.

Top, J.L, Breunese, A.P.J., Broenink, J.F., and Akkermans, J.M., 1995, “Structure and use of a library for physical systems models”, *Proc. Int. Conf. Bond Graph Modeling and Simulation ’95*, Granda, J.J., and Cellier, F.E., eds., SCS, San Diego, CA, pp. 97–102.

Ullman, D.G., 1992, *The mechanical design process*, McGraw-Hill, New York.

Ulrich, K.T., and Seering, W.P., 1989, “Synthesis of schematic descriptions in mechanical design”, *Research in Engineering Design*, Vol. 1, pp. 3–18.

Vries, T.J.A. de, 1994, “Conceptual design of controlled electro-mechanical systems”, PhD thesis, University of Twente, Enschede, Netherlands.

Vries, T.J.A. de, Breedveld, P.C., and Meindertsma, P., 1993, “Polymorphic modelling of engineering systems”, *Proc. Int. Conf. Bond Graph Modeling and Simulation ’93*, Granda, J.J., and Cellier, F.E., eds., SCS, San Diego, CA, pp. 17–22.

Ward, A.C., 1989, “A theory of quantitative inference for artifact sets, applied to a mechanical design compiler”, PhD thesis, Artificial Intelligence Laboratory, MIT, Cambridge, MA.

Weld, D.S., 1992, “Generating simplified models with confidence”, in (Falkenhainer and Stein, 1992), pp. 21–30.

Wilson, B.H., and Stein, J.L., 1992, “An algorithm for obtaining minimum-order models of distributed and discrete systems”, in (Falkenhainer and Stein, 1992), pp. 47–58.

Wirth, N., 1982, *Programming in Modula-2*, Springer Verlag, Berlin, Germany.