Published in *Computer Aided Conceptual Design*, Proc. 1994 Lancaster Int. Workshop on Engineering Design CACD 94, J. Sharpe and V. Oh (eds.), ISBN 0–901800–37–6, 1994, pp. 299–318

MAX: a mechatronic model building environment

Theo J.A. de Vries, Arno P.J. Breunese and Peter C. Breedveld

University of Twente, Department of Electrical Engineering and Mechatronics Research Centre Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands
Phone: +31 53 892817, Fax: +31 53 340045, E-mail: T.J.A.deVries@el.utwente.nl

Abstract: A description is given of the state of the art and the functionality of MAX, an expert system for supporting conceptual design of mechatronic systems. Three model building principles are combined in MAX:

– embedding equations in networks: a tight coupling between the graphical model formulation and the underlying equations assists the user in model building and evaluation.
– multiple model formulations: one system can be manipulated and inspected simultaneously in multiple formulations (languages).
– polymorphic modelling: a submodel definition is divided into a type that defines essential properties, and a specification that defines incidental properties. One type generally has multiple specifications, and types are organised hierarchically inside the library.

By means of a simple case study, the utility of these principles is demonstrated. It is shown that MAX is a powerful model building environment that is well adapted to usage by designers.

1 Introduction

We will consider the design of *controlled electro–mechanical systems*. Examples of such systems are assembly machines, automatic guided vehicles, consumer products like CD–players and video cameras, etc. The number of systems belonging to this class is rapidly growing. Recent advances both in electronics and computer technology, in software engineering and in control engineering enable the addition of ‘intelligence’ to traditionally purely mechanical products. This may increase their functionality, performance and reliability simultaneously, without necessarily increasing their cost.

Controlled electro–mechanical systems are generally made up of a large number of parts, which interact in many ways. Such systems are *complex*, because the whole is more than the sum of the parts in the sense that given the properties of the parts and the laws of their interaction, it is not a trivial matter to infer the properties of the whole (Simon, 1981). Their complexity is essential; with sufficient effort we may master it and even engineer an illusion of simplicity in their interfaces, but we can never remove the internal complexity (Booch, 1991).

The inherent complexity of a controlled electro–mechanical system will have to be tackled during its design in order that the outcome will be a reliable, high performance product with the proper functionality. The technique to do this has been known since ancient times: *divide et impera* (divide and rule). A design problem is usually decomposed into subproblems *in specific disciplines*. As a result, design is generally done in a team composed of different specialists. The resulting product typically consists of a collection of subsystems which provide functionality in a single domain and have relatively little mutual interaction. In other words, each function of the product is largely realized by one specific subsystem or module. Inter–module linkages have been avoided as much as possible. Also, decomposition over disciplines influences the phasing of the design process, as generally one subsystem will be designed after the other. The standard pattern is to design first the mechanical modules (which might be viewed as the skeletal and muscular subsystems), then the electronic parts (the sensory– and nervous subsystems), and finally the control modules (the brains).

Due to the nature of evolution, designers are continually faced with increasing demands both on the products they are designing and on their own productivity. They are required to come up with new designs which have more functionality, higher performance and better reliability (i.e. better *quality*) for less cost and in less time. These demands have become so high that the traditional approach to design outlined above is no longer sufficient. The sequence of design phases causes the innovation time to be too long and the quality to be hard to guarantee. The specialized subsystems are a barrier to the improvement of quality if costs are not to be increased. A new design approach has been proposed to change this: *mechatronics*.

A commonly used definition of mechatronics is “a synergetic combination of precision mechanical engineering, electronic control and systems thinking in the design of products and manufacturing processes” (IRDAC, 1986). Buur (1990) proposed a more concise definition: mechatronics is “a technology which combines mechanics with electronics and information technology to form both functional interaction and spatial integration in components, modules, products and systems”. Mechatronics can thus be viewed as an approach which recognizes the usefulness of decomposition of design problems over disciplines, but which emphasizes that this should not lead to domain–specific subsystems with minimal functional interaction and spatial integration. Benefits claimed to result from the use of a mechatronic design approach are mainly the increase of functionality (more intelligence and flexibility), of performance and of reliability without increasing costs.

The mechatronic approach specifically influences the *conceptual design* task, as it is during this task that most of the decisions are taken with respect to functional interaction and spatial integration of the system under design. Hence, if we are to enhance mechatronic design, we should offer sophisticated support for conceptual design of controlled electro–mechanical systems. In this paper, we describe the state of the art and the functionality of MAX, an expert system for supporting conceptual design of mechatronic systems.

In section 2, we shortly analyse conceptual design of mechatronic systems, and clarify that better support requires improved modelling capabilities for automated support systems. Three

proposals to improve support are discussed in section 3. Subsequently, the system in which these proposals have been incorporated is outlined in section 4. The utility of the system is shown in section 5 by means of a case study. Finally, future work is outlined in section 6 and conclusions are summarised in section 7.

2 Conceptual design of mechatronic systems

Designing in the mechatronic way requires better cooperation and coordination among the members of the design team. Their collaboration is to be realized through communication. However, members are generally individuals separated by discipline and/or functional responsibility. Their communication is difficult due to a *lack of* "*shared meaning*" (Konda et al., 1992). Simply stated, people just do not speak the same language or do not attribute the same value to a common word (figure 1). These communicative problems were also present in the traditional way of designing, but they are much more dominant in the new situation.

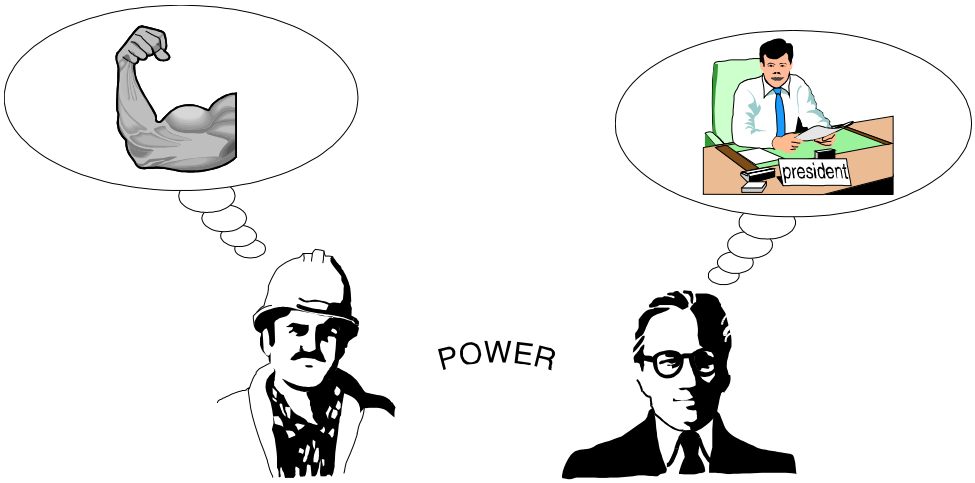

FIGURE 1 *Lack of shared meaning*

A second barrier to the use of a mechatronic design approach relates to what has been called the "*design process paradox*" by Ullman (1992). During design a circular dependency is present. Finding a good solution requires an understanding of the problem; conversely, problem understanding can almost only be gained throughout the solution process (figure 2). This can only be solved by trying to reduce the lack of knowledge as much as possible early in the process. In other words, one needs to learn about the design problem and the proposed solution as soon as possible. Compared to the traditional approach, the need for learning is both more serious and more difficult to satisfy when applying a mechatronic design approach.
From the above we conclude that conceptual design of mechatronic system can be significantly enhanced by improving the communication and learning processes that take place during this task. Our aim is to develop technological means to realize this, i.e., automated support.

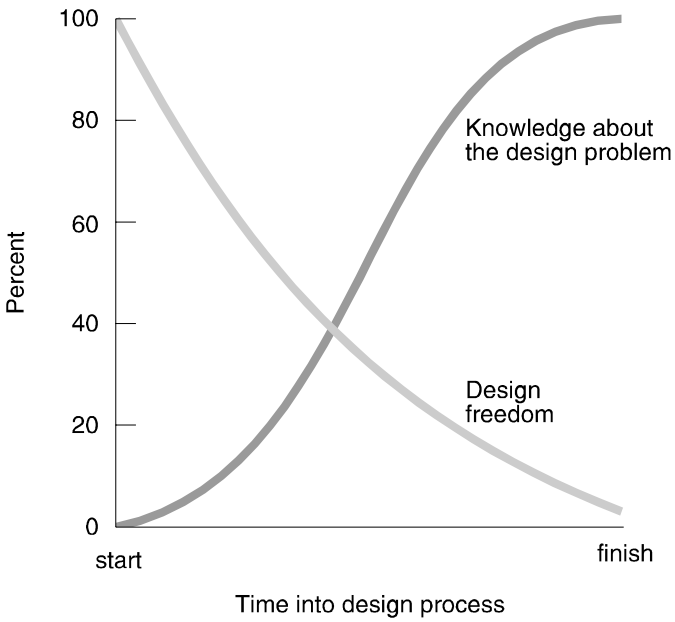

FIGURE 2 *The design process paradox (adapted from Ullman (1992) , figure 1.8)*

Both in learning and communication, and therefore in design, the use of *abstractions* plays a crucial role. Abstractions are descriptions of a system in terms of a formal language that are competent for answering specific questions. Abstractions provide the framework within which the evolution of the design object takes place. As Hoover et al. (1991) put it, 'the quality of the completed design depends on the ability of the designer to select useful abstractions, to use them to model the performance of the design, and to use the results of the evaluation to guide further design refinements'. Hence, we may expect that communication and learning problems are caused by an improper or ineffective use of abstractions.

In recent years, substantial progress has been made in the area of supporting the 'downstream' tasks of the design process, especially detailed design. This progress indeed has been accompanied by the possibility to make more advanced abstractions in CAE systems, such as geometric modelling (Finger and Dixon, 1989) parametric modelling and feature–based modelling (Salomons et al., 1993). Apparently, for these tasks support is being provided that properly uses abstractions. However, for the 'upstream' tasks such as conceptual design support is largely not available (Finger and Dixon, 1989).

The basic reasons why computer based design support systems are *not* suitable for conceptual design are according to Buur and Andreasen (1989):

1 Abstractions are *not formulated properly*. For example, they are not specified in terms of relevant concepts (i.e. concepts from the problem domain of the designer), or cannot be represented in a relevant perspective. Also, the descriptions are often 'stand alone', and not embedded in a structure of declarative knowledge that specifies the context of different abstractions and in what ways they are interrelated. We might say that the maintenance of abstractions in CAE systems gives problems, because the descriptions of the design object are *underformalized*.

2 Designers *cannot modify* the design object in the way they would like. Examples of this are the restrictive, rigid sequence of operations that systems allow, and the lack of support for converting design descriptions to a higher level of abstraction. If CAE systems lack flexibility, they limit possibilities for making design refinements, thereby missing the opportunity to enhance designing. This is caused by an *overformalization* of the design actions.

It should be noted that the second problem is mostly a consequence of the first one; because abstractions are not described in the proper way, it becomes possible to operate on them in ways that do not make sense. One way of preventing that is to severely restrict the manipulations that can be done on the abstractions. Of course, a better way would be to improve the maintenance of abstractions. This clarifies that in fact the dominant problem is one in the area of *modelling*: what is the exact meaning of abstractions, and how are they treated. Therefore, we studied the development of design support from this perspective (De Vries, 1994). We propose three new concepts to improve the maintenance of abstractions in the next chapters: two stemming from an analysis of linguistic aspects (embedding equations in networks and multiple model formulations), and one from an analysis of implementation aspects (polymorphic modelling). These concepts will be discussed in the next section.

3 Three proposals to improve support

3.1 Embedding equations in networks

One could argue that equations should not play a role in an environment for conceptual design, the reason being that conceptual design has a qualitative nature rather than a quantitative. To our opinion, this is a too restrictive view of the (conceptual) design process.

Ultimately, any design will be evaluated. The design must meet certain requirements. Merely using the term ‘good performance’ is not enough: we have to specify what is considered sufficient and what is not.

Because the goal parameters are in general not the degrees of freedom in our design (design parameters), we have to convert the design parameters into equivalent goal parameters. Simulation is a commonly used technique for this purpose. Although qualitative simulation is a known technique (Kuipers, 1986; Top et al., 1991), quantitative simulation is more appropriate for giving definite answers as to whether quantitatively formulated specifications are met.

Quantitative simulation cannot be performed without quantitative relations between variables in the system. These quantitative relations are formulated in a model. Although many model formulations may seem to work without equations, there is always a level in which equations are added. The link between equations and other formulations may be explicitly specified or left implicit. To deal with the model’s equations in a well–organised manner, a modelling system should provide an explicit link between graphical formulations and equations. In MAX, the property that models are connected by means of connections between (power or signal)

ports is used. A port can be a connection to a graphical submodel, but also a link to variables in an equation submodel. Figure 3 gives an example of a model which is specified using this approach.

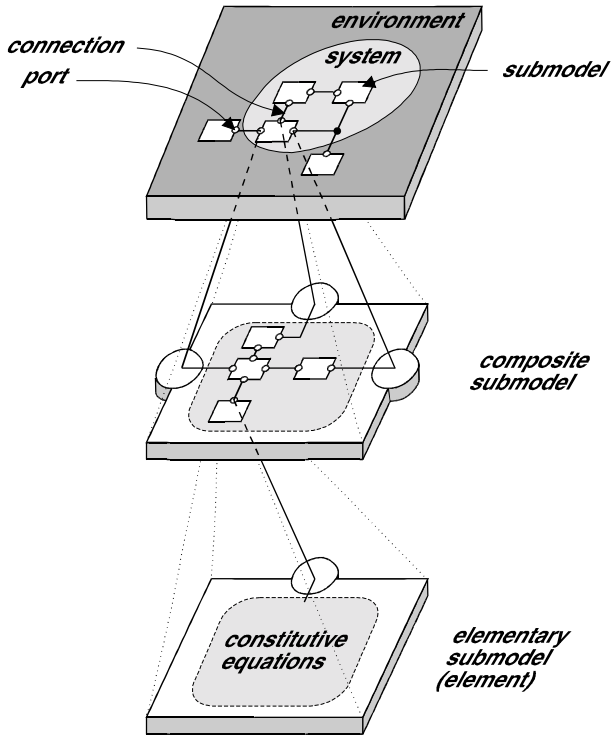

FIGURE 3 *Equation models as the leaves of a hierarchical model tree*

The system should offer the same level of user assistance and model verification in the equations as is offered in the graphical formulation. If we require that a modelling tool reports unconnected elements in a model graph to the user, then we may also require that the tool reports parameters which are not used, outputs to which no value is assigned etc. In analogy with the property that a good graphical editor refuses erroneous connections between submodels, we may also expect that an equation editor does not allow variables of different types to be assigned to each other. If models are incorrect, but to a limited extent, the tool should give a reasonable solution to fix the erroneous situation.

An aspect in which an equation editor differs from a graphical editor is the time at which feedback can be given to the user. In a graphical tool, the graph construction process is completely under control of the program. Erroneous, but well–defined situations (e.g. new node inserted, not yet connected to anything) may exist during the construction of a graph. These may be accepted during the editing process, but upon completion of the graph they should be reported as errors. In an equation editor, the user will be allowed to type a certain amount of text, and only after finishing the model entry, a check is started.

3.2 Multiple model formulations

The language in which a model is formulated determines what information can be expressed in a model, and whether it is easy or not to observe and interpret the information that is incorporated in the model. When using an integrated design approach, 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'. The use of multiple model formulations enhances communication between people from different disciplines, because they can choose a convenient formulation individually while still addressing the same issues. Also learning will improve by this concept, as one can select a formulation that is suitable for the problem at hand.

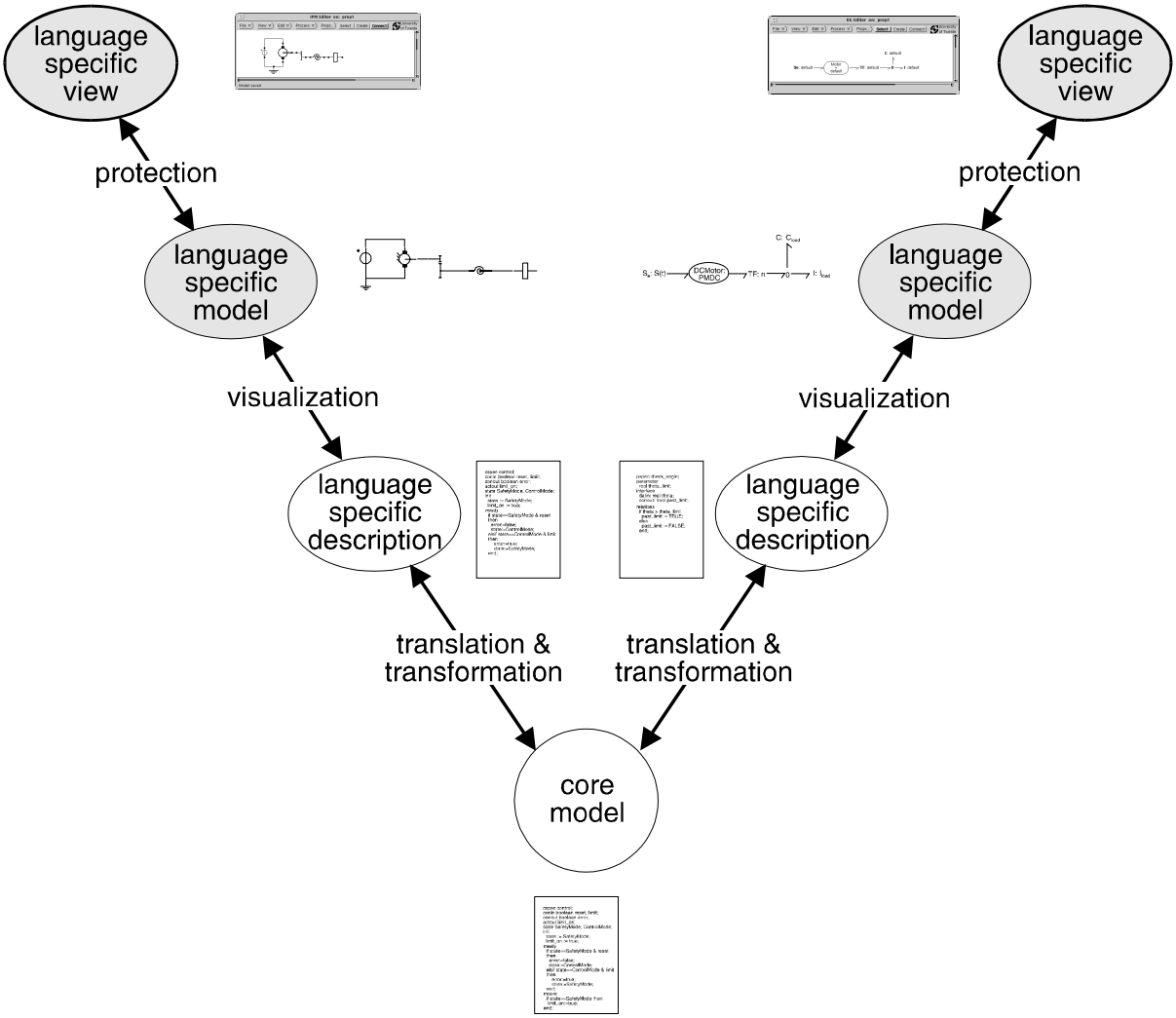

FIGURE 4 *Set–up for a system featuring multiple model formulations*

In order to keep multiple model formulations consistent and tractable, a system should be set up in the following way (figure 4). The user has access to a formulation through a language specific view of the design object in an editor. This view is created by adding protection to a

language specific model such that the user can only issue commands that have observable effects in the view itself. The (graphical) language specific model results from a visualization of a (textual) language specific description that only stores intrinsic model properties. The language specific description finally is linked with a central core model by means of bidirectional transformations. The core model integrally stores the information needed for all different model formulations, and coordinates the language specific descriptions.
For conceptual design, a set of graphical languages is required that at least enables description of function, behaviour and initial form. In the domain of controlled electro–mechanical systems, the set consisting of iconic diagrams (e.g. electric circuit diagrams or mechanical system diagrams), bond graphs (Paynter, 1961; Breedveld et al., 1991) and a variant of the Modern Structured Analysis notation (Yourdon, 1989; Wijbrans, 1993) seems appropriate. Feasibility of the system set–up outlined above has been shown for iconic diagrams and bond graphs (Breunese, 1992; De Vries, 1994)

3.3 Polymorphic modelling

Like Ward (1989), we believe that the design object is not a collection of descriptions of single artefacts, but rather defines *sets of equivalent artefacts*. Reasoning in terms of sets is the major way designers deal with the large solution space faced during designing. Using equivalence sets, designers can reason about many individual artefacts in an economical way. They only have to consider the features that define an equivalence set, i.e. that all artefacts that are a member of the set have in common. Because they use set–based descriptions, they do not have to think of other features, which would distract them from the problem. Set–based reasoning is their method to prevent over–constraining of the solution space, while still having reasonably small amounts of information to deal with. The evolution of the design object then is the continuous modification of the sets that are contained in the design object. This evolution takes place by changing the features that define these sets.
To support this, computer–based systems should provide means to describe a part of the design object as (a specific instance of) a *set of artefacts*. This means that it must be possible to separate essential properties of a subsystem from incidental properties. The essential properties are those properties of a subsystem that are *typical*, i.e. that are essential to properly classify the subsystem. For example, an essential property of a DC–motor is that it converts electrical energy into mechanical energy. The specific parametric values of the motor are incidental properties; they need not be given in order to classify the subsystem.
The computer science term for the above outlined separation is *modularisation*, a structuring principle that was most explicitly applied in the programming language MODULA (Wirth, 1982). Modularization in this context means that a subsystem definition is divided into two parts: a *type* that defines essential properties, and a *specification* that defines incidental properties.

An interesting consequence of modularising subsystem definitions is that subsystem types can be usefully organised in a kind–of hierarchy using *subtyping*. That is, a type is expressed as a special kind of its supertype, with some additional essential properties. This is beneficial for at least two reasons. Firstly, it implies that types are defined incrementally, which is a very natural and economic way of working. Secondly, a type can be generalized to a supertype or specialized to a subtype, thus allowing for stepwise refinements.
A second consequence of modularisation is that it becomes possible for one type to have more than one specification, i.e. subsystem types become polymorphic (figure 5). This provides a second way to stepwise refine a model in an organised manner; one can vary the specification of a type.

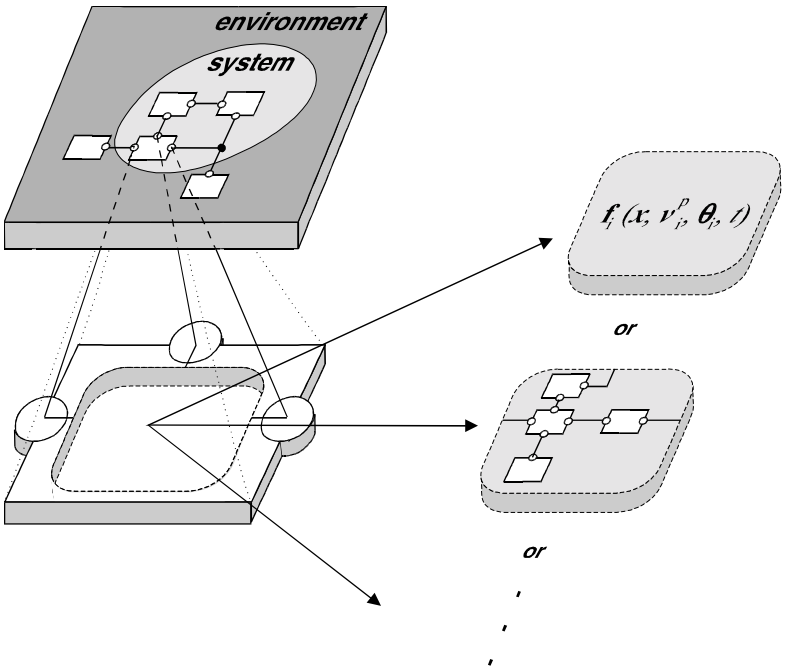

FIGURE 5 *Polymorphic modelling: model in terms of modular, classified components*

The combination of subtyping and modularization in a model building system has been called polymorphic modelling (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 and coherent structure. Furthermore, it facilitates the structured manipulation of a model, like the creation of alternative design solutions and the variation of detail incorporated in models.

4 Outline of the system

This section describes the MAX system (Van Dijk et al., 1992). It should be kept in mind that this is a momentary impression; MAX is still being modified and expanded. The main tools of the MAX system are the library browser, the graphical editors and the equation editor.

4.1 Library browser

The ‘Library browser’ enables the user to browse through the libraries of subsystems and components. Figure 6 depicts the tool. The library of design solutions plays a major role in supporting conceptual design.

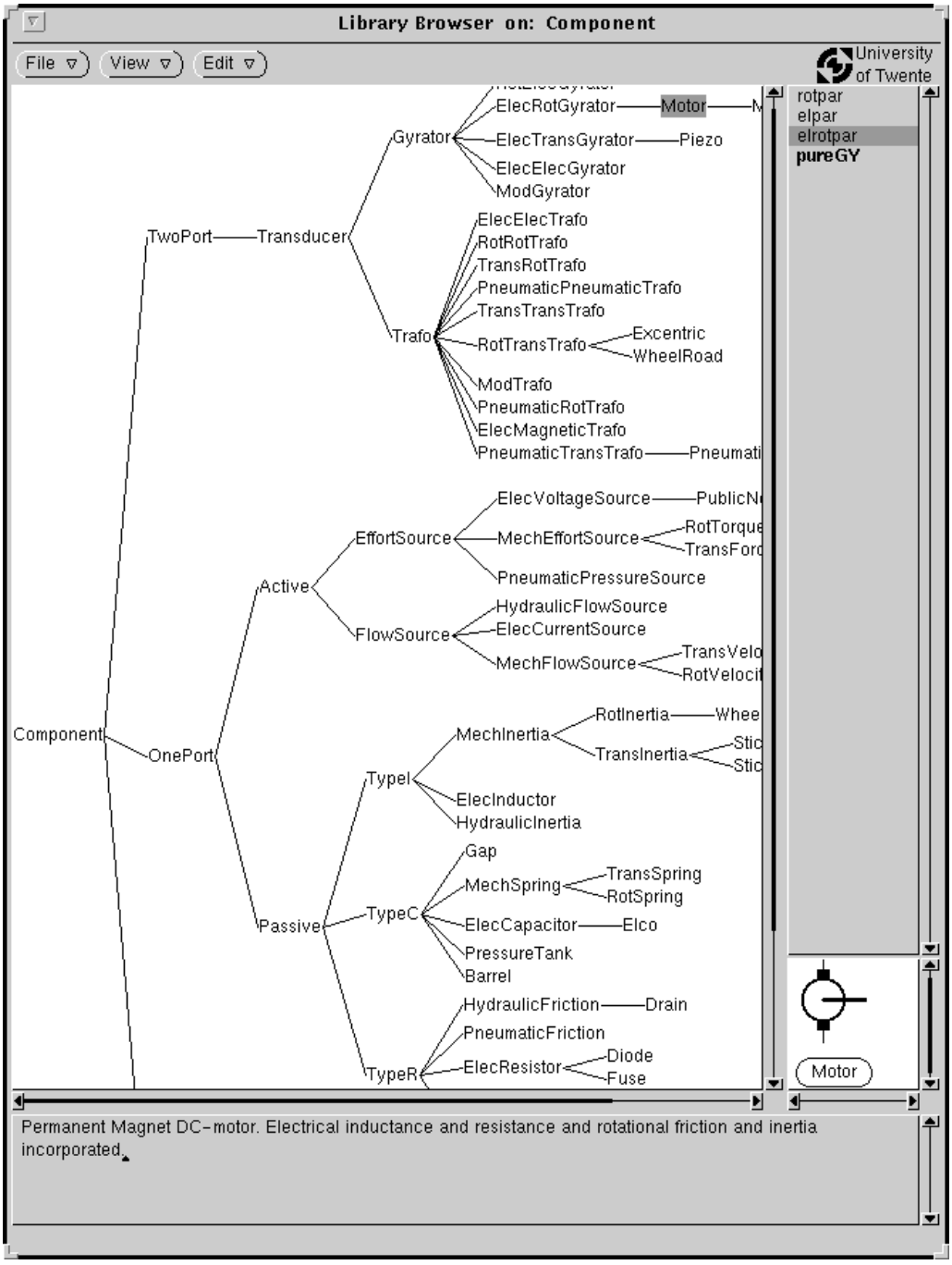

FIGURE 6 *MAX library browser*
Top left: hierarchically organized subsystem types
Top right: list of specifications of the selected type
Middle right: available representations of the selected type
Bottom: comment for the selected type and specification

The hierarchy of subsystem types in MAX is such that creation of useful alternative models by means of sequences of the operations ‘generalize’ and ‘specialize’ upon a type is supported. Furthermore, types were given names from a design perspective, i.e. functional and not physical. This means that a component is expected to *mainly* behave conform the stated *function*, although other physical effects may be incorporated in the specification. Finally, emphasis was put on types for (controlled) electro–mechanical subsystems.
Three hierarchies are currently available:

1 *signal processors* hierarchy: this hierarchy contains subsystems of which the functionality mainly is to process signals in some way. This also implies that most of these subsystems only have signal ports.
2 *components* hierarchy: in here, models of physical components are contained. This is the most extensive hierarchy. Figure 6 depicts an illustrative part of this hierarchy.
3 *special* hierarchy: in this hierarchy, model parts that are specific for a certain formulation are contained. Also, ports are incorporated here. This hierarchy is rather static; it only needs adaptation when new formulations or new physical domains are considered.

4.2 Graphical image editors

MAX features two graphical image editors, namely the ‘IPM editor’ and the ‘bond graph editor’. These editors allow entry, analysis and modification of main models and specifications of subsystems in the iconic diagram and bond graph formulation respectively.
Besides the obvious drawing capabilities required for these editors, they also provide interactive checks of the correctness of the model. The editors will refuse to make connections between incompatible ports, and will report ‘short-circuited’ connections.
The bond graph editor features powerful algorithms for causality assignment. Causal analysis is a non–trivial issue that needs attention, also in a design context (Rosenberg, 1987; Van Dijk, 1994). Also, bond graph models can be exported to and imported from the CAMAS modelling and simulation environment (Broenink et al., 1992).

4.3 Equation editor

In section 3.1 it was pointed out that equations should play a role in a system like MAX. The obvious moment for attaching equations to a model is during the specification of a type. Besides graphical specifications (which add a layer to the model hierarchy), equation specifications are also possible. These specifications terminate the bond graph and/or iconic diagram hierarchy. Ultimately, all leaves of the model tree will be filled in with equation specifications. To enter the equation specifications into the system, the MAX equation editor is used. The equation editor is constructed considering the principles presented in section 3.1 and corresponding to the overall modelling approach of the MAX system.
The equation editor is presented to the user in a window with two areas of text. The first area shows information relevant for creating equations which can be derived from the type for

which the specification will be used. The text in this area cannot be edited by the user. The second area is the area in which the equations for the specification are actually entered. Initially this area contains a template showing how a valid model text is constructed.
The equations are formulated in SIDOPS++, a language specially designed for the description of physical systems (Breunese, 1993). Although the notation of expressions in SIDOPS++ resembles that in popular general–purpose programming languages like C or PASCAL, there are important differences.
In SIDOPS++, a model consists of declarations and equations. In the declarations part local variables, parameters, constants, and tables are defined. Using these entities and the declarations which are made in the type (e.g. the port information), equations are specified. Each equation consists of two expressions which are presumed to be equal at all moments of (simulated) time. The equations part is not a computer program: it is not required that the left hand side can be calculated from the right hand side of the equation. Furthermore the order of the equations is not of importance. It is not possible to use flow control structures as known from imperative programming languages, because all equations should be valid for each moment in time. SIDOPS++ has no `for`, `while`, `repeat` or `if` statements. The `if then else endif` construct does exist in SIDOPS++, but it is an expression rather than a statement, so that the declarative nature of SIDOPS++ is not violated.
When the user has finished typing the equations, the MAX equation editor will perform an analysis of the model. The first step in the analysis of the equations is a syntactical analysis. The text entered by the user is first converted into a data structure in the computer's memory which is suitable for further processing. A parser generated with the T–gen translator generator (Graver, 1992) is used for this purpose.
Once the equations are parsed correctly, it is by no means certain that the equations are in fact valid and meaningful. Though computers can not identify all errors which humans make, a considerable share of the errors can be detected automatically. The MAX equation editor performs the following checks:

– All identifiers (variables, parameters, constants) should be declared before use.
– Functions and table references should have the correct number of arguments. The types of the arguments must match the expected types.
– Operators and operands must match in infix and prefix expressions.
– The types of the left hand side and the right hand side of an equation should be the same.
– The equations should be meaningful for the type. An expression must be specified for all outputs. For each port, there must be an expression for the effort variable or the flow variable (or both).

After all checks are completed successfully, the model is assumed to be correct. It may then be used for analysis or simulation. To support the reuse of models, the MAX equation editor stores correct specifications in the library.

5 Case study

Hereafter, a relatively small design problem is presented that shows how MAX can be applied in a design context. This is sufficient to demonstrate the functionality and utility of the system. More realistic test cases, i.e. for larger problems in an industrial environment, are currently being studied. The first results are successful, but proprietary.
The problem to be discussed is the design of a mechatronic drive system to be used by third year master students for gaining some experience in the design and implementation of a practical controller. The main requirements were:

– low cost
– clearly observable difference between controlled and non–controlled performance
– robust to misuse
– variable stiffness and load

Figure 7 depicts the initial design proposal (an existing set–up), entered as an iconic diagram.

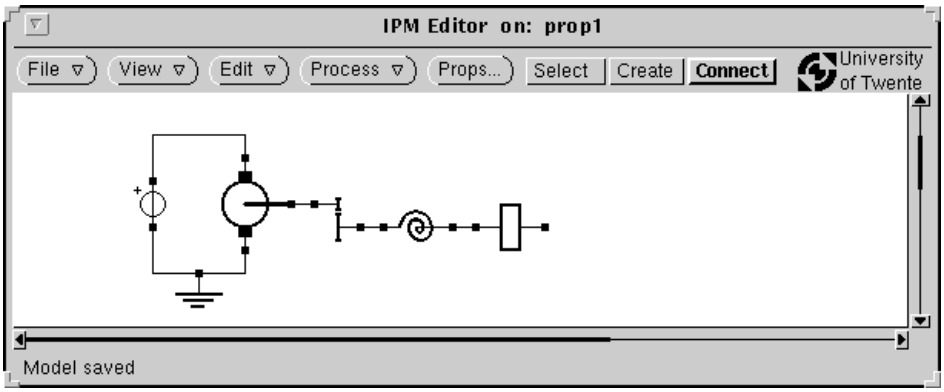

FIGURE 7 *Initial design proposal*

Subsequently, the model is transformed to a bond graph formulation, see figure 8.

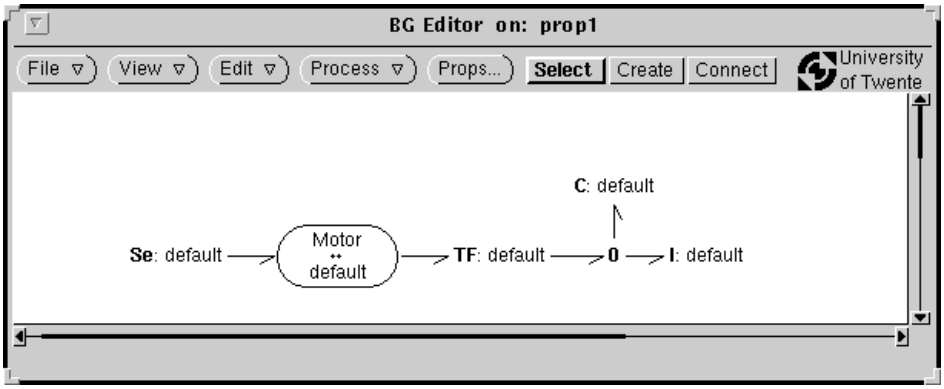

FIGURE 8 *Initial proposal in bond graph formulation*

Alternative solutions, whereby parts of the system are realized in other domains, were systematically found by subsequently generalizing and specializing parts of the bond graph. A

good candidate for this operation is the flexibility in the transmission (the C that represents torsional stiffness): it might be easier to vary the flexibility if it is realized in the translation mechanical domain. This solution is created from the available one by introducing a transformer between the torsional stiffness and the load, and after that changing the domain of the bond graph fragment between the two transformers to the mechanical translation domain. Figure 9 depicts the result.

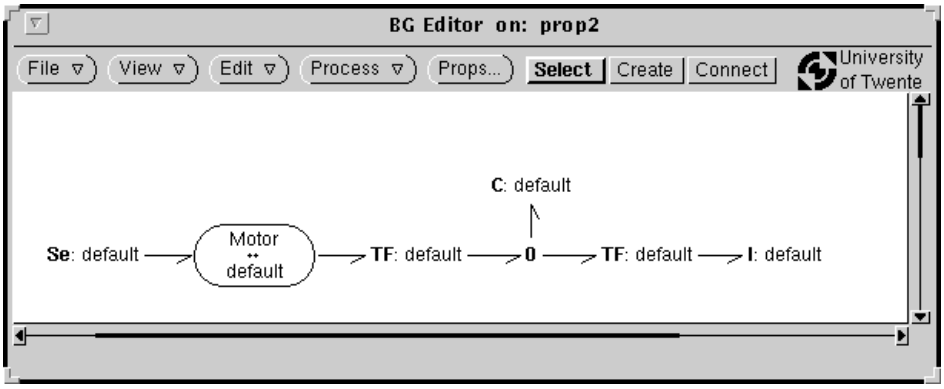

FIGURE 9 *Alternative solution, systematically created by means of subsequently generalizing and specializing model parts*

The newly created proposal is transformed back to the iconic diagram formulation, (figure 10). It shows that the solution can be realized as two pulleys coupled by means of an elastic belt. This solution has certain advantages compared to the initial proposal; variable stiffness and load can be obtained easily by incorporating two pairs of pulleys with differing transmission ratios, and/or by varying the number of elastic belts that couple the pulleys. This appears an elegant way to satisfy the robustness and cost requirements too. Therefore, this solution was selected to work out further.

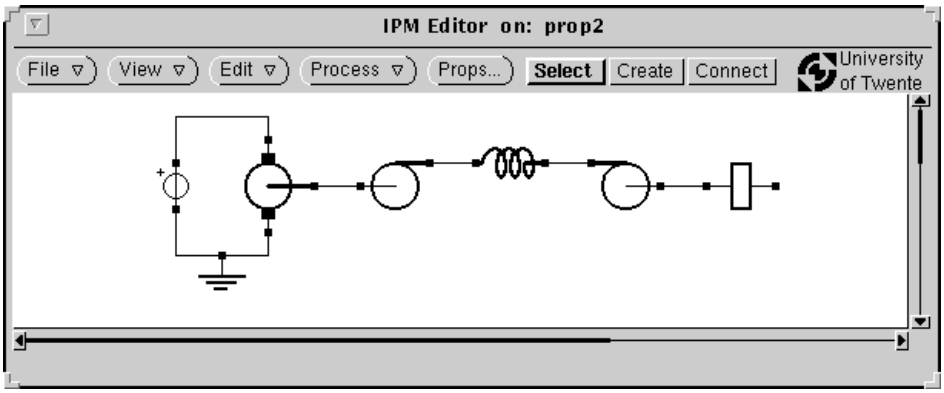

FIGURE 10 *Alternative solution, iconic diagram formulation*

Before entering the stage of simulation to assess the performance of the proposed design, three rather obvious refinements to this model are made:

– the specification of the motor is changed from a simple ideal transducer (i.e. neglecting storage and dissipation phenomena) to a more realistic model including electrical inductance and resistance and mechanical friction and inertia. This can be accomplished easily by selecting the specification ‘elrotpar’ for the motor submodel (cf. figure 6).
– the submodel representing the elasticity of the belt is refined by including some damping.
– a new specification is made for the spring used to model the elasticity of the belt to take into account that the spring is less compliant when pulled beyond its nominal length than when contracted. The equation editor is used to make the new specification.

The equation editor is started from the library browser. The type which is selected in the library browser is automatically loaded into the equation editor. We use the translation spring as the type for the new specification. We only have to declare local variables and parameters and to specify the equations related to the submodel, because the equation editor automatically generates those parts of the SIDOPS++ text which can be derived from the type information.
We use the symbol x for the elongation of the spring relative to its nominal value, F as the force acting on the spring, and C_1 and C_2 for the compliance values in the two regions of operation. The spring characteristic with the equations for both regions is shown in figure **11**.

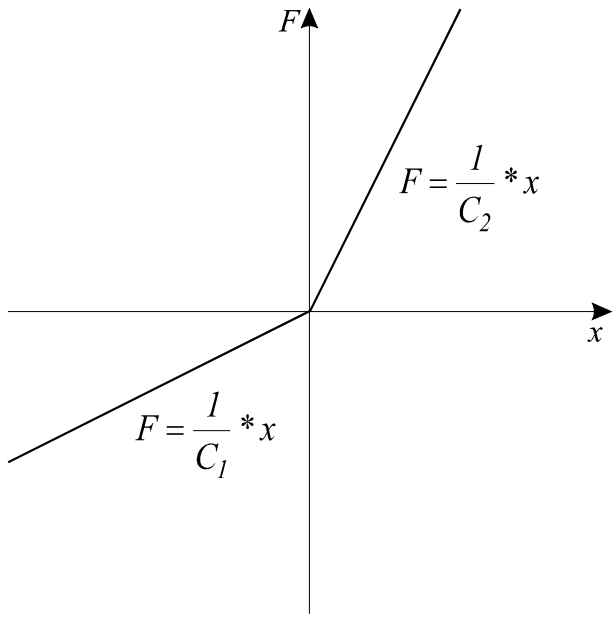

FIGURE 11 *Spring characteristic*

In SIDOPS++ these equations are formulated as:

*F = if x<=0 then (1/C_1)*x else (1/C_2)*x endif;*

After simply entering this equation into the equation editor, we are notified that the identifiers *F*, *x*, *C_1* and *C_2* are not declared. The equation editor will assist in declaring these identifiers by opening a dialog in which we can specify the type, and – if relevant – upper and

lower bound and default value of the identifiers. Valid SIDOPS++ text is automatically generated and the analysis of the specification is resumed.
With this additional declaration, the equations are correct with respect to the declarations. However, during further analysis of the equation submodel, another flaw is discovered. This is reported in an error window (figure **12**).

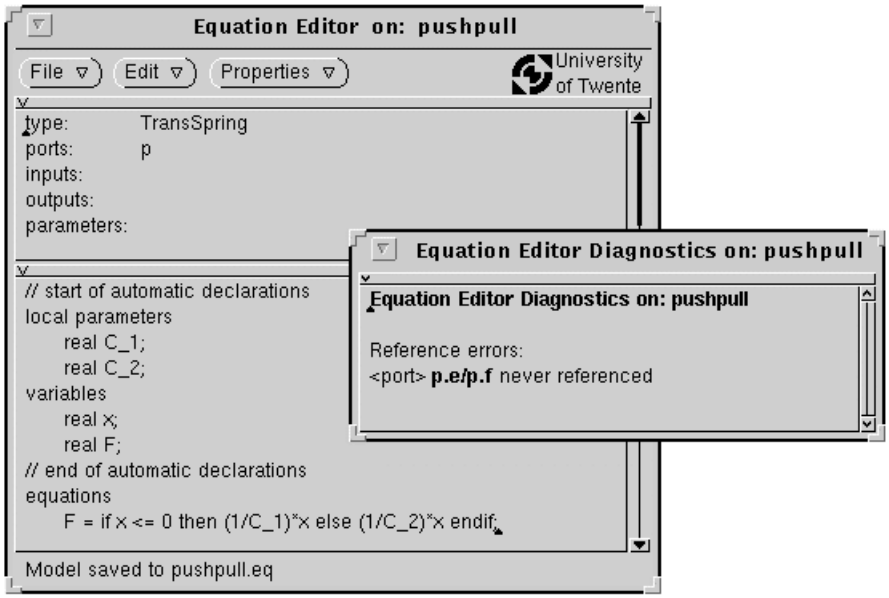

FIGURE 12 *Equation editor with error message window*

It appears that we have correctly coded the constitutive equation depicted in figure 11 into SIDOPS++ code, but that we have forgotten to make the link to the port variables of the spring submodel. We should relate variables *x* and *F* to the port variables *p.e* and *p.f* (the effort and flow variable respectively). We know that the displacement of the spring (*x*) is the integral of the flow variable, and that the force (*F*) is the effort variable. We can thus add two extra lines of SIDOPS++ code:

x = int(p.f);
F = p.e;

Once this is done, the model can be recompiled. The equation editor will now accept the model. The model is also stored in the MAX library to allow future reuse.
Upon completion of the equation model for the spring we can now fill in all submodels used in our system. The hierarchy can be inspected in the hierarchy browser supplied with the MAX system (figure 13).

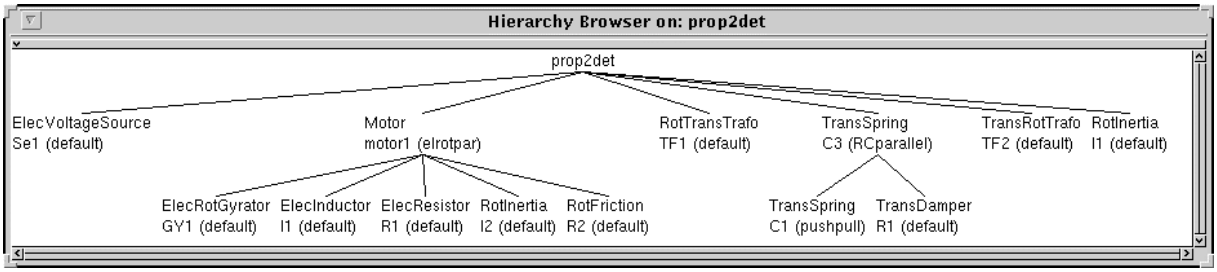

FIGURE 13 *Detailed model of the selected alternative (motor specified, and belt detailed to incorporate damping and special spring characteristic)*

To make a quantitative analysis of the model, we export it to the CAMAS environment and perform a simulation. The result of this simulation is shown in figure 14.

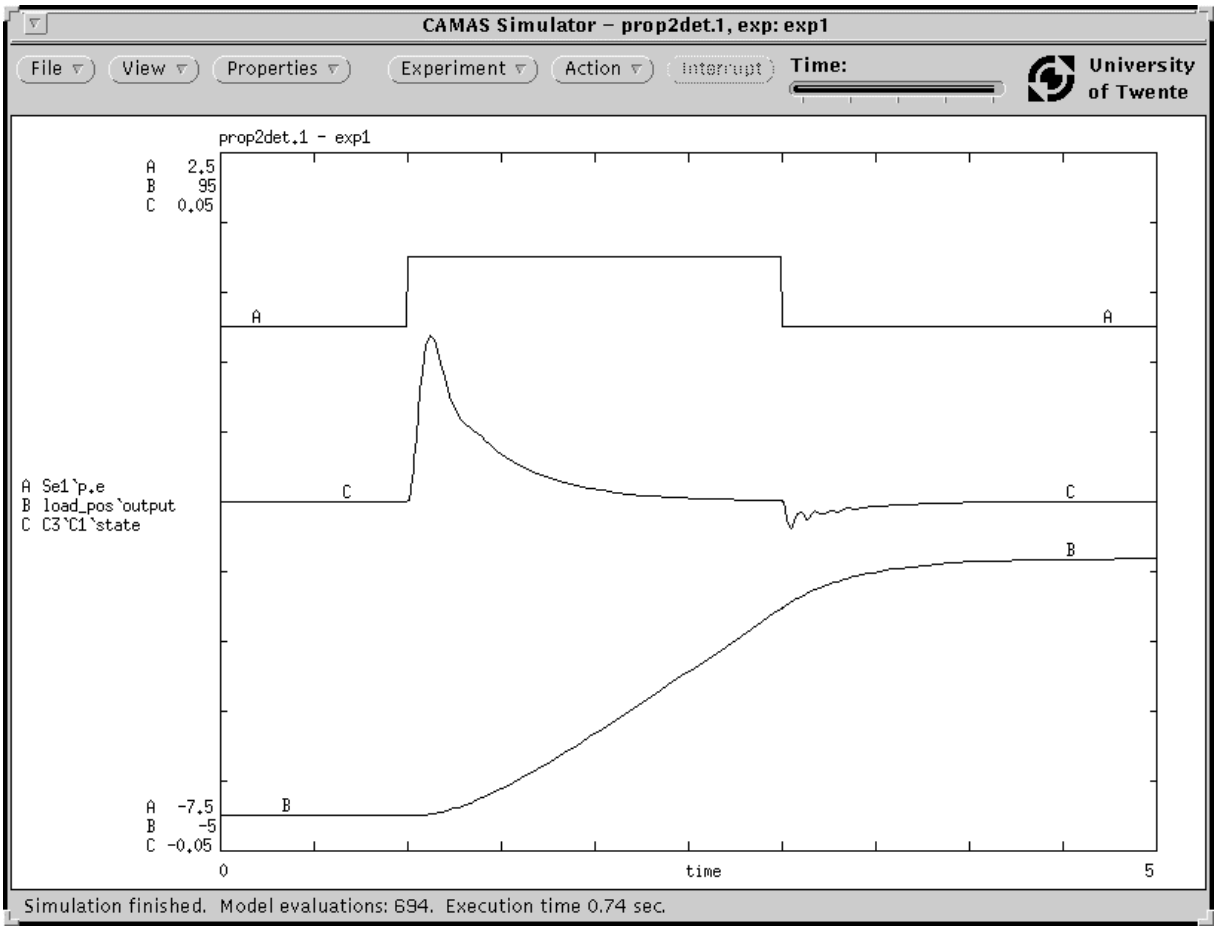

FIGURE 14 *Result of the simulation of the model constructed in the case study.*

6 Future work

MAX is still subject to further development. In this section we discuss two areas in which we plan to extend MAX in the near future.

Classification. Polymorphic modelling classifies models as specifications of certain types. Ideally, the modelling system should be able to determine whether a given model is classified correctly or not. This implies that submodels should be checked to see whether they are compatible with the assumptions made by the type. For example, if a specification for a C–type storage element indeed integrates a flow–variable and that the result of the integration is used to compute an effort–variable.
Parameter relations. In a hierarchical system with fine–grained decomposition, one parameter is often contained in multiple submodels. A rigid body in a gravity field is often modelled as a combination of an inertia and a force source. The mass of the body is a parameter in both submodels. This dependency should be expressed explicitly in the model. This requires the possibility to define parameter relations between submodels.

7 Conclusions

Enhancing conceptual design of mechatronic systems requires automated systems that support designers in their communication and learning processes. In communication and learning, the use of abstractions plays a major role. Therefore, modelling capabilities of design support systems need to be improved. However, design support systems that properly support maintenance and manipulation of models are lacking. Therefore, we have firstly addressed the question in what terms models should be made, i.e. in what way we should formulate models in a computer–based system. Secondly, we have investigated how computer–based systems should support the creation and modification of models. Three proposals have been formulated:

– To deal with the model's equations in a well–organised manner, a modelling system should provide an explicit link between graphical formulations and equations. An equation editor should enable entry of equations of elementary models in a user–friendly way. The form of the equations should be relatively close to the natural form, but also formal enough to allow thorough checks of the consistency of the equations.
– When using a mechatronic design approach, 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 a system set–up that enables multiple model formulations and yet will keep different formulations of a model consistent and tractable.
– Computer–based support systems should provide means to describe a part of the design object as (a specific instance of) a set of artefacts. This requires the modularisation 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 modularisation and subtyping in model building is called 'polymorphic modelling'.

The case study has illustrated how these three model–building principles have been implemented. It demonstrated that equations can be easily embedded into hierarchical models with the high level of support offered by the system. The case study further clarified that intermediate results of transformations from one model formulation to another and that polymorphic refinements are useful in order to learn about the actual design problem. That is, they help to check validity and completeness of the problem statement and to verify correctness, consistency and suitability of the proposed solution. As a result, more informed decisions are made earlier on in the design process. It is essential that little effort and time is required for realizing transformations and polymorphic refinements, in order to communicate about the actual design problem, that is, to describe and represent the proposed solution. As a result, more alternative solutions can be considered and proposed alternatives can be better judged on their merits than before. Overall conclusion is that MAX is a powerful model building environment that is well adapted to usage by designers.

References

Booch (1991), *Object Oriented Design with applications*, Benjamin/Cummings, Redwood City, CA, U.S.A.

Breedveld, P.C., R.C. Rosenberg and T. Zhou (1991), Bibliography of bond graph theory and application, *J. Franklin Institute* **328** (5/6), 1067–1109.

Breunese, A.P.J. (1992), *Design and implementation of a mechatronic modelling environment using object oriented principles*, MSc thesis no. 92R071, Control Laboratory, University of Twente, Enschede, Netherlands

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

Broenink, J.F., J. Bekkink and P.C. Breedveld (1992), Multibond–graph version of the CAMAS modeling and simulation environment, *Bond graphs for engineers*, P.C. Breedveld and G. Dauphin–Tanguy (eds.), Elsevier, Amsterdam, Netherlands, 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 M.M. Andreasen (1989), Design models in mechatronic product development, *Design Studies* **10**, 19–34

Dijk, J. van, T.J.A. de Vries, A.P.J. Breunese and P.C. Breedveld (1992), Automated mechatronic systems modelling using MAX, *Bond graphs for engineers*, Elsevier, Amsterdam, Netherlands, 269–280

Dijk, J. van (1994), *On the role of bond graph causality in modelling mechatronic systems*, PhD thesis, University of Twente, Enschede, Netherlands

Finger, S., and J.R. Dixon (1989), A review of research in Mechanical Engineering Design. Part I: descriptive, prescriptive, and computer–based models of design processes, *Research in Engineering Design* **1**, 51–67.

Graver, J.O. (1992), *T–gen User's Guide*, University of Florida, Gainesville, FL

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

IRDAC (1986), *Opinion on R&D needs in the field of mechatronics*, Industry R&D Advisory Comittee of the Comm. of the EC, Brussels, Belgium

Konda, S., I. Monarch, P. Sargent and E. Subrahmaniam (1992), Shared Memory in Design: A Unifying Theme for Research and Practice, *Research in Engineering Design* **4**, 23–42

Kuipers, B.J. (1986), Qualitative simulation, *Artificial Intelligence* **29**, 289–338

Paynter, H.M. (1961), *Analysis and design of engineering systems*, MIT–press, Cambridge, Mass., U.S.A.

Rosenberg, R.C. (1987), Exploiting bond graph causality in physical system models, *Trans. ASME J. Dyn. Sys. Meas. Control* **109**, 378–383

Salomons, O.W., F.J.A.M. van Houten, and H.J.J. Kals (1993), Review of Research in Feature–Based Design, *J. Manufacturing Systems* **12** (3), 113–132

Simon, H.A. (1981), *The Sciences of the Artificial*, MIT Press, Cambridge, Mass., U.S.A.

Top, J.L., J.M. Akkermans, and P.C. Breedveld (1991), Qualitative Reasoning about Physical Systems: an Artificial Intelligence Perspective, *J. Franklin Inst.* **328** (5/6), 1047–1065

Ullman, D.G. (1992), *The mechanical design process*, McGraw–Hill, New York, U.S.A.

Vries, T.J.A. de, P.C. Breedveld, and P. Meindertsma (1993), *Polymorphic modelling of engineering systems*, Proc. Int. Conf on Bond Graph Modeling and Simulation, Western Simulation MultiConference, SCS, San Diego, California, U.S.A., 17–22

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

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, Mass., U.S.A.

Wijbrans, K.C.J. (1993), *Twente Hierarchical Embedded Systems Implementation by Simulation*, PhD thesis, University of Twente, Enschede, Netherlands

Wirth, N. (1982), *Programming in Modula–2*, Springer–Verlag, Berlin, Germany

Yourdon, E.N. (1989), *Modern Structured Analysis*, Prentice–Hall, Englewood Cliffs, NJ, U.S.A.