INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN

ICED '95

PRAHA, AUGUST 22–24, 1995

STRUCTURING PRODUCT MODELS TO FACILITATE DESIGN MANIPULATIONS

Theo J.A. de Vries and Arno P.J. Breunese

ABSTRACT

In this paper, we analyse the problems associated with product models in computer–based support systems for conceptual design of mechatronic systems. Basically, current product models fail in the sense that they seriously limit the designer in possibilities for making and modifying a description of the system under design. Based on the analysis, we propose a product model structure that is more suitable in this respect.

1. INTRODUCTION

We consider here the design of *mechatronic systems*. Examples of such systems are assembly machines, automatic guided vehicles, consumer products like CD–players and video cameras, etc. Buur [1] proposed a concise definition of mechatronics: it 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'. 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 cost. 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 multi–disciplinary systems.

Engineering design in general, and thus also conceptual design of mechatronic systems, essentially addresses two issues:

1. Determine the *requirements* for the product to be designed.
2. Formulate a model (the *proposed design solution*) that satisfies these requirements and that can be realised.

As conceptual design is an ill–structured process, these issues are not addressed sequentially, but rather *iteratively*. That is, conceptual design is a process of continually refining the proposed design solution on the basis of an evolving set of requirements. Consequently, the list of requirements will (partly) be dependent of the proposed design solution and vice versa [2]. Therefore, it is fruitful to incorporate both requirements and proposed design solution integrally in one design *product model*.

In many computer–based design systems, product models consist of either a set of constraints (the requirements-oriented form) or an assembly of components or features (the solution-oriented form). To properly capture dependencies of requirements and proposed solution, an integration of both forms is required. Recently, several propositions have been made to obtain this [3]. Characteristic for such product models is that they are 'knowledgeable'; quite some data about properties of the proposed design solution and relationships between parts of it are contained a product model. Therefore, a library of reusable product models is indispensable in computer–based support systems featuring integrated product models [4].

With respect to integrated, knowledgeable product models for conceptual design of mechatronic systems, we have identified the following problems:

1. Proper integration of requirements and proposed solutions in a product model involving assemblies of components is difficult. This is worked out in section 2, where a product model structure is proposed that solves this problem.
2. Gradual refinement of the proposed solution is generally not supported; fully specified assemblies or components need to be selected during synthesis and structured search for alternatives is hard. In section 3, this aspect is analysed and an addition to the above proposition is given to deal with this.
3. Libraries of product models are unstructured and, when of adequate size, hard to maintain. This obstructs reuse of design knowledge. Section 4 addresses this issue, and it is shown how this can be improved by further extending the proposed product model structure.

Conclusions are listed in section 5.

2. INTEGRATING REQUIREMENTS AND PROPOSED SOLUTIONS

CADET [4, 5] is a system supporting embodiment design in which requirements and proposed solution are effectively combined in a product model. In this system, the proposed solution is described as a set of interconnected components (tangible parts) and interfaces (interactions between components). Both components and interfaces are characterised by parameters, specifying for example material and material properties, geometry and relative positioning. Requirements are incorporated in the product model as inequality constraints upon (combinations of) parameters of components and interfaces. This product model structure is effective, because requirements and proposed solution are clearly distinguishable, and at the same time their dependencies are explicit and they can be treated integrally as constraints to be satisfied in the remainder of the design process.

However, to apply the same ideas in our situation, a few difficulties with respect to the product model structure in CADET must be resolved:

1. It is tailored towards mechanical design, in the sense that interconnections imply force exchange and relative positions. In mechatronics, also electric and electro–mechanical (and other) interconnections are present. Hence, this requires generalisation of the product model structure.
2. It is focused on description of static behaviour, with extensions foreseen to include kinematics. For mechatronic design, however, dynamic behaviour is the major concern.
3. As reported so far [5], it is not yet fit for describing an assembly of components and interfaces as one 'composite subsystem'. Mechatronic systems typically involve a large number of components. Composite subsystems are indispensable then, as they introduce a 'part–of hierarchy' in the product model, thus enabling the designer to keep track of the system under design. This principle is intuitively applied by many designers to keep the design problem and its proposed solution tractable.

In order to structure the model of the interaction between systems or processes, the interface between components is specified in terms of *ports* and *parameters*. A port represents a variable (or set of variables) that links the 'outer world' (the related subsystems) and the 'inner world' (the detailed description of the behaviour) of the subsystem. A parameter represents a characteristic property that is available to the outer world for use in requirements, and to the inner world for use in the composition of parameters of lower level subsystems.
The use of ports allows for the description of interconnections beyond the mechanical domain. It also provides an extension towards dynamic models. The values of the port variables can vary over time. By specifying constitutive equations that define relations between the port variables and involve the parameters of the subsystem, the dynamic behaviour of a system can be expressed.
Using a uniform interface in terms of ports and parameters, it becomes possible to treat a composite subsystem in the same way as a component. The interface of a composite subsystem consists of those ports and parameters of all constituent parts that are not used exclusively for internal connections. This technique is not limited to a single level: it is possible to have composites within composites.

The propositions made above lead to a powerful product model structure in the form of *hierarchical networks*. This structure indeed supports mechatronic design by addressing the relevant issues. The generic structure is depicted in figure 1a. A concrete example of a simple electro–mechanical system (figure 1b) shows how the structure is used in practice.

3. REFINABLE PRODUCT MODELS

A difficulty with the product model structure proposed in the previous section, and in fact of most product models in design support systems, is that when incorporating a subsystem in the design, the design freedom is limited to parameter modifications; structural modifications are not possible. That is, we have to select a full model for a subsystem as soon as we want to include it in our design. For example, if we want to describe the outline of the simple electro–mechanical system, we have to be explicit about the composition of the motor or its equations. In conceptual design, however, we typically do not want to be bothered by such details rightaway. When including a motor in a design, we merely mean to say that we need a transducer from the electric to the rotation-mechanical domain; in due time, we will refine the model and be explicit about the exact kind of motor, and thus its internal structure.

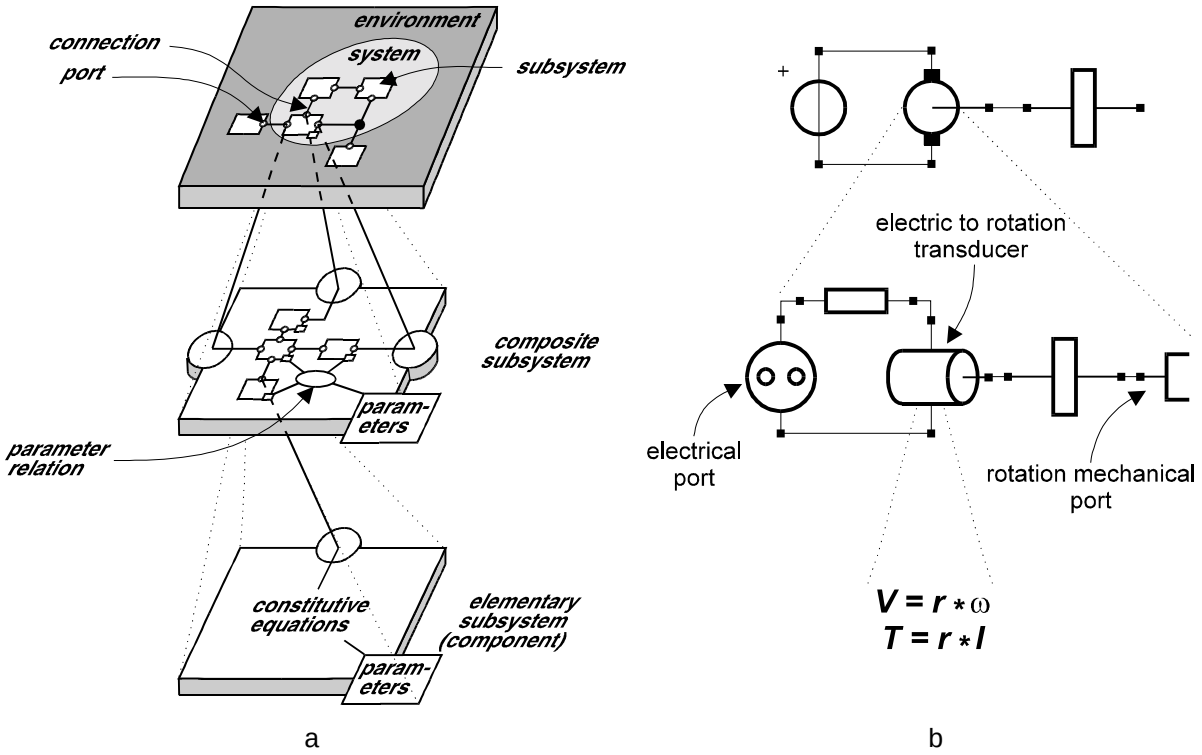

Figure 1 Product model structure in the form of hierarchical networks
a) General structure
b) Example of a simple electro–mechanical system

The proposed product model structure can be extended to support this in the following way. We should distinguish essential characteristics and incidental characteristics in a subsystem description. *Essential characteristics* are those characteristics on basis of which a subsystem is classified (that 'make it what it is'), and *incidental characteristics* are the remaining data. In other words, we modularise a subsystem description into a *type* (defining the essential part) and a *specification* (defining the incidental part). For an electric motor, the electric port, where power enters the subsystem, the rotation-mechanical port, where power flows out of the system, and the transduction ratio (the motor constant) are essential characteristics; the parts of which it its composed or the equations that describe its behaviour may take varying forms, and thus are incidental characteristics. This is mostly true: ports of a subsystem and some of its parameters are essential attributes, and hence should be defined in the type; its structure (the 'inner world') and remaining parameters are incidental characteristics, and hence part of the specification. As incidental characteristics may take varying forms, we can think of multiple specifications for one type; i.e. types become *polymorphic*. This polymorphism can be exploited in two important ways [6]: to describe a subsystem in varying resolution or in alternative forms. Figure 2 depicts the adapted structure for the product model.

Note that a type in fact determines a set of equivalent artefacts [7], and that a specification may be either a set of equations (i.e. a particular parametrisation) or a structure of lower level subsystem types (i.e. a particular combination of smaller subsets of artefacts). That is, only the combination of a type and an elementary specification make up a fully specified submodel. The combination of a type and a composite specification just determine one level in the model hierarchy; lower level layers are left undetermined.

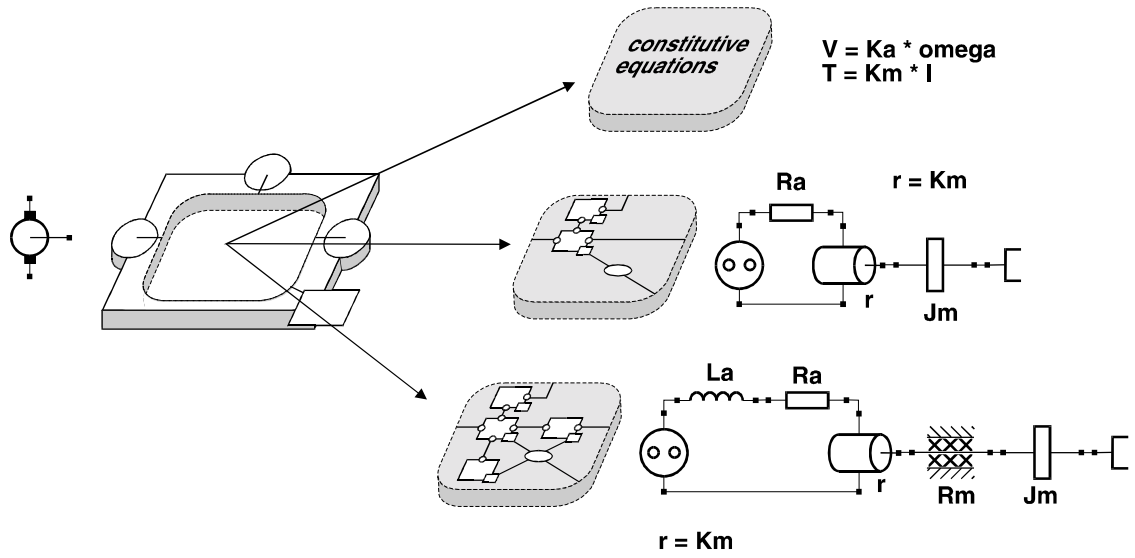

Figure 2: Modularisation of a subsystem description into a type and a specification

4. MODEL LIBRARIES

Use of integrated product models of the kind discussed here is only effective if the building of these models does not require the designer to perform many extra actions that he would otherwise not do. In effect this implies that models should preferably be built during conceptual design as if the designer was making a sketch or scheme. This is feasible by providing to the designer a (large) library of connectable 'icons', where each icon represents a model of a subsystems.

There is at least one condition for this approach to work: the library must cover models ranging from quite abstract, functional subsystems ('a transducer') to commercially available, concrete and completely known subsystems ('a permanent magnet DC servo motor of vendor A, type X'). In terms of the previous sections, an abstract subsystem is described by a subsystem type with unknown parameters and a default or no specification, and a commercially available, concrete subsystem is described by a type with particular parameter values and with a particular specification, of which the types have particular parameter values and a particular specification, etc. In other words, an abstract subsystem is a model hierarchy of minimum depth (= 1), with all parameters unknown, and a concrete subsystem is a fully determined model hierarchy, where all parameters have an exact value.

To deal with this, we introduce the concept of a realisation. A *realisation* is a description of a piece of model hierarchy and of the associated parameters. Therewith, it is the proper instrument to store in a library anything between an abstract subsystem and a commercially available, completely known subsystem.

In order to obtain an extendible library that can be well maintained and contains reusable design knowledge, these models must not only be available, but also be systematised in a knowledge structure tailored to support design manipulations. The product model structure proposed so far is well suited for this for the following reasons:

1. Subsystem types can be meaningfully organised 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. The importance of this is that by changing the type of a subsystem to the supertype (generalisation) or to one of the subtypes (specialisation), it can be made to describe a more or a less abstract subsystem [6]. Also, the possibility for incremental definition of the type properties leads to a coherent, practical library structure that is well understandable, extendible and maintainable.
2. Realisations can be categorised under the type that is at the top of their hierarchy. Within one type, realisations can be listed according to the amount of detail incorporated, and realisations (partly) incorporating the same lower level subsystems and/or parameters can be grouped. In this way, related variants of subsystem types are readily available.

The basis for the product model structure as proposed here is the concept of polymorphic modelling [8]: modularise a subsystem into a type and a specification, and relate types in a kind–of hierarchy. This proposition has been implemented in a system for conceptual design of mechatronic systems, called MAX [9, 10]. Space limitations prevent us from discussing the functionality of this system; interested readers are encouraged to refer to the references or to contact the authors. Also the SCHEMEBUILDER system [11] features a product model comparable to the one proposed here.

5. CONCLUSIONS

Proper integration of requirements and proposed solution in a product model can be obtained if each subsystem therein is characterised by two kinds of information: its ports and its parameters. A *port* is a set of variables that is available to the 'outer world' of a subsystem to make connections to, and that is available to the 'inner world' of a subsystem for expression of its behaviour. A *parameter* is a characteristic property of the subsystem that is available to the outer world to be used in e.g. requirements, and that is available to the inner world as e.g. the composition of parameters of lower level subsystems or equations. In this way, the system under design is modelled in the form of a *hierarchy of networks*.
In conceptual design, it must be possible to include in the proposed design solution a subsystem that is only partly characterised. To support this, we propose to modularise a subsystem description into a *type*, which defines the essential characteristics that hold for all realisations of the subsystem, and a *specification*, which defines the remaining, incidental characteristics that may take varying forms. This 'polymorphism' can be exploited in two important ways: to describe a subsystem with varying detail or in an alternative form.
In order to have an extendible subsystem library that can be well maintained and contains reusable design knowledge, the models in it must range from abstract and generic to concrete and fully parametrised. Also, they must be systematised in a knowledge structure tailored to support design manipulations. This can be obtained by organising subsystem *types* in a *kind–of hierarchy*, and by providing for each subsystem type an appropriate set of *realisations*, which describe a less or more complete subsystem hierarchy. Such a structure enables guided modification of a subsystem in a proposed design solution to less or more abstract versions.

REFERENCES

[1] **Buur, J. (1990),** *A theoretical approach to mechatronics design*, PhD thesis, Institute for Engineering Design, Technical University of Denmark, Lyngby, Denmark

[2] **Hoover, S.P., J.R. Rinderle and S. Finger (1991),** Models and abstractions in design, Proc. of ICED '91 (Zurich, Switzerland), WDK-20, Heurista, Zurich, Switzerland

[3] **Roozenburg, N.F.M. (1993),** *Proceedings of the ICED '93* (The Hague, Netherlands), WDK-22, Heurista, Zurich, Switzerland

[4] **Thornton, A.C. and A. Johnson (1993),** Constraint specification and Satisfaction in Embodiment Design, in [3], 1319–1326

[5] **Johnson, A.L., A.C. Thornton and C.F. Fong (1993)**, Modelling functionality in CAD: implications for product representation, in [3], 1610–1617

[6] **Vries, T.J.A. de, J. van Amerongen, and P.C. Breedveld (1993),** A model of the design process for development of CAE systems, in [3], 1409–1417

[7] **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

[8] **Vries, T.J.A. de, P.C. Breedveld and P. Meindertsma (1993)**, Polymorphic modelling of engineering systems, *Proc. Int. Conf. Bond Graph Modeling and Simulation '93*, San Diego, CA, 17–20 January 1993, 17–22

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

[10] **Breunese, A.P.J., T.J.A. de Vries, J. van Amerongen and P.C. Breedveld (1995)**, MAXimizing impact of automation on modeling and design, internal report no. 010R95, Control Laboratory, University of Twente, Enschede, Netherlands; submitted to J. Dynamic Systems, Measurement and Control

[11] **Bracewell, R.H., D.A. Bradley, R.V. Chaplin, P.M. Langdon and J.E.E. Sharpe (1993)**, Schemebuilder, a design aid for the conceptual stages of product design, in [3], 1311–1318

CORRESPONDENCE

Theo J.A. de Vries
EL-BSC-RT
Mechatronics Research Centre Twente
University of Twente
P.O.Box 217
7500 AE Enschede
Netherlands

Phone: +31 53 892817
Fax: +31 53 340045
E–mail: T.J.A.deVries@el.utwente.nl