POLYMORPHIC MODELLING OF ENGINEERING SYSTEMS

Theo J.A. de Vries, Peter C. Breedveld and Piet Meindersma
Control Laboratory, EE Department and Mechatronics Research Centre Twente
University of Twente, P.O. Box 217, 7500 AE Enschede, Netherlands
phone +31-53-892788; fax +31-53-340045; e-mail vri@rt.el.utwente.nl

Abstract
The heart of modelling is formed by three aspects: the decomposition of a system into interrelated subsystems, called components here, the classification of these components and relations, and the representation of the resulting model. Modelling support (i.e. computer tools) should provide means for utilising powerful decomposition, classification and representation concepts, such as those incorporated in bond graphs for example. In this paper, an analysis is presented of techniques which (partially) enable the implementation of these concepts in computer-based modelling tools.

The first and oldest technique applied as such is typing, which enabled the encapsulation and parametrisation of components and hierarchical modelling. This improved decomposition and classification possibilities of modelling support. Secondly port based interfaces have been incorporated in modelling tools. Such interfaces made it possible to model with an object oriented approach instead of a process oriented one, and improved the representation of models by enabling a component to be depicted as a network of lower level components. It is shown that typing and port based interfaces together allow the decomposition and its representation as done in bond graph modelling, but they do not support classification properly. The use of inheritance by means of subtyping improves this, but leads to components that can not be instantiated. To solve this, it is proposed to use the technique of modularisation.

Modelling tools, modularisation involves the division of component definitions into two parts: a type, defining essential properties of a component, and a specification, defining the incidental ones. One type may have more than one specification, i.e. component types become polymorphic. Modularisation can, in combination with subtyping, support classification, and therefore it is said to extend hierarchical modelling into polymorphic modelling.

An implementation in a modelling system is presented and examples are shown which demonstrate features of a polymorphic modelling system.

1. INTRODUCTION
Modelling of continuous time systems involves the characterisation of systems by a set of state variables \( \mathbf{x} \) and by a set of (possibly time-dependent) relations \( f(\mathbf{x}, \mathbf{u}, t) \) on these state variables and environmental variables \( \mathbf{u} \). These relations are supposed to be satisfied at all time instances. Models are created in order to get insight into a system and to understand its behaviour. The difficulty in this is that the system generally is complex, and therefore the heart of modelling is formed by three aspects, as can be shown in a theoretic model of the modelling process (De Vries and Breedveld 1992): the decomposition of the system into interrelated subsystems, the classification of these subsystems and relations, and the representation of the resulting model. The term components will be used here as a synonym for subsystems. In this context, a component is not necessarily a physical, concrete part, but can also be of a conceptual nature.

The system characterisation \( f(\mathbf{x}, \mathbf{u}, t) \) can be regarded as the most straightforward and plain form of a model. A model with such an internal structure does not explicitly incorporate the decomposition into components and their classification, and it is represented by means of mathematical equations. This is depicted schematically in figure 1a. In a system like TK Solver (Universal Technical Systems, Inc. 1988) models are specified in this form. While such systems may be powerful for solving problems with their models, they do not offer support for the step ahead of that, namely for obtaining good models of complex systems, as their models will not provide much insight and will not be easily understood. Bond graphs (Breedveld, Rosenberg and Zhou 1990) by contrast are a good modelling environment in this respect: the reticulation as introduced by (Paynter 1961) allows for explicit and meaningful decomposition and classification, and at the same time the notation he and others (Karnopp and Rosenberg 1968; Breedveld 1982) provided allows a graphical representation that enables direct reasoning and is unambiguous.

Models play a crucial role in the analysis and design of engineering systems (Rosenblueth and Wiener 1945), and therefore the availability of powerful support for the engineer in his modelling task is of key importance. Modelling support should provide means for utilising powerful decomposition, classification and representation concepts, such as those incorporated in bond graphs for example. In this paper, an analysis is presented of techniques which enable the implementation of these concepts in computer-based modelling tools.

Available techniques that have been exploited in this sense are typing and port based interfacing, and they will be discussed in section 2.1 and 2.2 respectively. These two techniques together extended modelling capabilities of computer tools so that the representation and decomposition as done in bond graph modelling could be supported. However, it will be explained in section 3 that presently available modelling systems still do not capture classification properly. Therefore, a new concept, called polymorphic modelling, will be introduced in section 4, and it will be shown that this concept will resolve for the shortcomings. Section 5 will present some examples of the features that a polymorphic modelling system enables. Conclusions are listed in section 6 finally.
2. EXISTING TECHNIQUES

2.1. Typing

Early systems used to support modelling, of which ACSL (Mitchell and Gauthier 1976) is a well known example, were mainly based on a technique called macro-modelling. A macro is a definition of a component, and consists of two parts: the heading specifies the macro name and its formal parameters, and the body describes the internal structure of the component. A component defined in this way can be incorporated in a model by invoking the macro name with the actual parameters only. The top level model might be viewed as a special kind of component, namely one where the parameters are equal to environmental variables.

Macro-modelling can be regarded as a specific form of what is generally known as typing, i.e. the categorisation of objects according to their usage and behaviour. The key characteristic of typing in computer science is that the definition of an object (i.e. a component here) is separated from its usage. This is done in order to enforce correctness of object usage and to encapsulate local information of an object (Cardelli and Wegner 1985). To enable this, the type definition of the object consists of two parts: the interface part (the heading in case of macros) specifies the information that has to be provided at object usage time, and the implementation part (the body) contains the encapsulated local information of the object.

When applying typing in modelling tools, the internal structure of a component can be expressed, besides in terms of mathematical equations as before, in terms of lower level components as well. In other words, a component can be specified as an aggregation of parametrised, encapsulated lower level components. This holds for the top level model, but also for lower level components. As a result, the complete model takes the form of a part-of hierarchy, and the tools feature the concept of hierarchical modelling. These effects of typing on the form of a model are depicted in figure 1 b. In this figure and later on, the rectangles around components symbolise typing.

With the advent of Simula (Dahl and Nygaard 1966), the technique of typing was given an important extension: Simula allowed a type definition (named a class here) to be expressed as a specialisation (a subclass) of an other type definition. This introduced the notion of inheritance into computer science: a type automatically inherits the properties of its supertype. While this appeared very powerful in a programming context, it had only moderate influence on modelling tools. The reasons for this will be examined in section 3.

Application of typing in computer based modelling tools is useful, because:

- It improves decomposition by enabling the model to be organised in a part-of hierarchy of submodels
- It enables classification by giving a type name to components
- It improves representation of models by encapsulating the internal structure of components
- It facilitates reuse of components by means of parametrisation
- It enables the component library to be organised around (related) types.

2.2. Port based interfacing

The interface part of macros very much resembled those of "conventional" program part interfaces, like in Pascal. The heading specifies two kinds of parameters: value-parameters, which are constant (i.e. the "true parameters"), and var-parameters, also known as reference parameters, which form the
input and/or output of the component. When using a component in an aggregation, the actual variables of the aggregation are passed to the component and processed locally.

With port based interfacing, the var-parameters (the input and output of a component) are defined as a special kind of variable, namely as ports (or terminals). What is special about ports is that they can be incorporated in relations of the "superpart" of the component, i.e. in the aggregation in which the component is embedded. So instead of passing the variables of the superpart down to the component (as in macro-modelling), the variables of the component are "popped" up to the superpart. Due to this, an aggregation of components can be simply described in terms of connections, as a connection is a relation stating that the variables of the two involved ports are equal. It is useful then to restrict the internal structure of components to be either a pure aggregation of lower level components or a pure set of equations (Broenink 1990). In that case, a component that is an aggregation can be represented as a graph or network of interconnected components. In case the component is not an aggregation, its internal structure will define the constitutive equations. This is depicted in figure 1c, where the ports are reflected by "bulbs".

When using non-port based interfaces, components are incorporated in models in the form of procedure calls which process the input and deliver output. This reflects a process-oriented approach towards modelling: a model is built by stating the processes that take place. This approach might to some extent be appropriate for modelling components that themselves contain no other components, as these indeed can be characterised by process. However, modelling generally is concerned with the decomposition of a component into smaller subsystems that are interconnected in some way, until some set of basic elements is found. This typically reflects an object-oriented approach, and port based interfaces enabled modelling tools which supported this. The first modelling tool that incorporated port based interfaces was Dymola (Elmqvist 1979). Most of the currently available systems (for example the bond graph oriented systems CAMAS (Broenink, 1990) and Enport (Rosenberg 1974)) have included a variant of this technique, which shows its importance.

Port based interfaces improved computer based modelling support, because

- They improve decomposition by enabling an object-oriented modelling approach,
- They allow representations of a model in the form of a network, which can be depicted graphically
- They enable the component library to be further organised according to port attributes

3. WHAT STILL IS LACKING: PROPER CLASSIFICATION

Application of typing and port based interfacing enables computer based modelling tools to fully support the decomposition and its representation as done in bond graph modelling. Systems such as CAMAS and Enport demonstrate this. CAMAS also supports classification to some extent: each component is declared as an instance of a certain class. However, in these (and comparable) systems, there is a one-to-one correspondence between a component's type and its internal structure. This means that for each single component type there is one unique form of instances. Stated more abstractly, types are monomorphic (of one form) in these systems. This characteristic is the cause for the fact that it is not possible to define generic component types in available modelling systems. For example, it is impossible to define a general component type "friction" that can be incorporated in a network and that covers both viscous (linear) and Coulomb (non-linear) friction. This is due to the difference in their internal structure. In (bond graph) modelling however, the invocation of generic components that are specialised later on is a common procedure. Above the impossibility to define generic components, it is mostly also impossible to represent a component as a specific instance of a more or less general class. Being able to do this is however desirable, as this property is utilised often when reasoning about a model, for example during causality assignment in a bond graph. These two limitations show that available modelling systems do not support classification properly.

One might guess that the use of type inheritance as introduced in Simula would solve for this problem. Applying inheritance allows the creation of generic classes that capture common properties of multiple component types. However, two component types with a common superclass hardly ever share (a part of) their internal structures. In the case of the frictions for example, the constitutive equations are very different. As a result, generic classes do not define component types, because they do not define an internal structure. Instead, they only specify abstract component types, i.e. component types which can not be instantiated and therefore can not be used to type a component in a model. Abstract component types allow to gradually define common properties of components, and thus enable component libraries to be organised in a kind-of hierarchy. However, only the leaves of this hierarchy are types of actual components which can be instantiated. In computer science terms: type inheritance provides for abstraction of component types.

As a result, the hierarchy incorporated in the component library of systems featuring abstraction is not a kind-of hierarchy of components itself, but rather a hierarchy of component attributes. While this is very useful for development purposes (Rosenberg, Whitesell and Reid 1992), it still does not give computer-based support the expressive power to capture classification properly: generic components still can not be created. In the next section, a solution for this problem is proposed.

4. POLYMORPHIC MODELLING

Above, it has been shown that the basic problem why classification is not supported properly is, that the inheritance mechanism enables abstraction, but leads to types that can not be instantiated. In other words, the combination of abstraction and typing is too restrictive. Once this is clear, the solution is easy: an abstraction barrier should be defined, which separates properties of (generic) component definitions that are inherited by subtypes from properties that will not be inherited. This is analogous to a technique known as modularisation in computer science, a program structuring principle that was most consequently applied in Modula-2 (Wirth 1982).
The abstraction barrier is defined here as to separate essential properties and incidental properties of a component. Essential properties are the properties of a component that are "typical", i.e. which are necessary to classify the component. Essential properties are defined in the component type (as before), and are inherited by subtypes. By contrast, incidental properties of a component are not typical, and may take varying forms. Incidental properties are no longer defined in the component type and therefore not inherited, but are defined in a specification of the component type. This indicates that a complete model will exhibit two choices for each component: its essential properties, generally reflected by the type, and its incidental properties, generally reflected by the specification. Furthermore it follows that component types become polymorphic due to the definition of an abstraction barrier: one type may have many specifications (many forms). In figure 2, the consequences for the model of the introduction of an abstraction barrier are reflected by the dashed lines.

There are two main reasons why a type may have more than one specification. Firstly this can occur because the type simply has instances with differing behaviour, although the essential properties are the same. In the example of the mechanical friction, this is the case. The second reason can be that one specification describes a behaviour in more detail than the other, i.e. they have a differing resolution. Which specification should be chosen for a component in a model then depends on the context for which the model is used.

Combining modularisation with typing and port based interfaces in a modelling tool gives such a system a number of additional features. The polymorphism of component types is the key element underlying these features. This has been expressed by saying that modularisation extends hierarchical modelling into polymorphic modelling. The concept of polymorphic modelling is useful, because:

- It enables the component library to be organised in a kind-of hierarchy, such that components are specialised incrementally down-wards.
- It improves classification of components by means of generic as well as specific typing.
- It improves understanding of models by separating essential and incidental characteristics.
- It supports the iterative and evolutionary nature of modelling more directly.
- It improves the possibilities for creating and maintaining (mutually consistent) alternative models.

In the next section, examples will be given which will show these features.
5. EXAMPLES

To learn about and to demonstrate the concept of polymorphic modelling, the techniques of typing, port based interfacing and modularisation have been implemented in the MAX system (Van Dijk et al. 1992). MAX stands for Modelling and Analysis eXpert, and is an expert system that allows the user to create and manipulate models (among others) in the bond graph language. It is a true modelling system in the sense that it does not actually support any computation with models. For these purposes, links to existing packages are (and will be) provided, e.g. to CAMAS for simulation. The examples described hereafter are all taken from modelling sessions with MAX.

5.1. Hierarchical component library

The modularisation of component definitions into a type and a specification has two important consequences for the component library of a computer based modelling tool: the component types are organised in a true component hierarchy, and for each type available specifications are listed. This is depicted in figure 3a, where a part of the component library of MAX is shown. The component types are part of a specialisation tree, and for the selected type the specifications are shown. In MAX, each of the types has at least a default specification, and therefore each of the types in the hierarchy can be instantiated.

Polymorphic modelling imposes a coherent structure on the collection of components contained in the library. Analogous to the situation in programming (Cardelli and Wegner 1985), it is expected that the ease of reusing common properties is incidental to the clarity and conceptual parsimony provided by the coherent structure. However, this requires the hierarchy to be carefully built up. It should factor out common properties of component types stepwise, such that meaningful generic components arise. The order in which common properties are refined is an important issue, as will become clear in the next section. In MAX, setting up the hierarchy has currently been done for a small number of relatively simple components only. Figure 3b depicts how the type specialisation from general one-port to electrical capacitance has been done. Bold-faced attributes have been defined at that level in the hierarchy, the other attributes are either inherited or default. The figure also shows which attributes currently have been assigned as possibly essential properties. This aspect is still under research.

5.2. Evolutionary modelling approach

Although modelling is an iterative and interactive process, the global working direction is of a top down fashion: one usually starts with decomposing the top level model into components, and in the next "loop" detail is added to the model by decomposing these components again etc., thereby gradually expanding the number of hierarchical layers of the model. This exactly maps to the modularisation as introduced above: during decomposition of the top level model, concentration is on determining the types of its components, while the next loop is concerned with determining the proper specification for each of these components, etc. A polymorphic modelling system separates types and specifications, and thus is well suited to support a top down modelling approach. In fact one uses the property here that a type can have specifications which differ in resolution. In case of monomorphic types this approach cannot be supported, because in order to determine the type of a component, one also has to make a choice for the internal structure right away. One could of course just not regard the internal structure at first (or use an empty or new type), and adjust it only later on. But this is more like a way around the lack of support!

MAX has been designed such that it does not force the user to apply a top down approach. The modeller can equally well start describing a specification of some component in an editor first, and after that select the type to which this specification belongs. In other words, a bottom up approach is enabled as well. Moreover, a mix of working in a top down and a bottom up...
fashion is possible. For example, the modeller can start top down by developing the top level model and proceed with deeply specifying one of the components (i.e. into several hierarchical levels), and decide on basis of the insight obtained during this to continue bottom up by adjusting the top level model again, i.e. making a new specification for it, etc. It is this mix that characterises the evolutionary nature of modelling. A real modelling process will never be purely top down or purely bottom up, and computer based tools should be adjusted to that. Polymorphic modelling enables the support of such an evolutionary modelling approach.

5.3. Creating alternatives or analogues

Previously it has been shown that polymorphic component types lead to a hierarchy of components, which can be depicted in a tree as in figure 3. This implies that apart from the root, each component has a more general supertype, and apart from the leaves, each component has more specialised subtypes. Now an interesting perspective opens up: components can be generalised and/or specialised. Generalising a component means nothing more than resetting the appropriate attribute from the essential properties. For example, an electric capacitor can be generalised into a type C component by removing its domain attribute (resetting it to "Domain"). In case of specialising, the effect is reverse: the appropriate attribute is given a value which is specific for the subtype in question. So a type C component can be specialised to a mechanical translation spring. These manipulations can be done without actually editing the network. With equal ease, one can vary between specifications of a component which differ in behaviour. Both options are only possible when polymorphic types are available, and they are interesting in order to create alternatives or analogues of a model. Especially in a design context this might be worthwhile. As can be imagined, the manner in which the hierarchy of components is built up largely determines which alternatives are easily created, and which alternatives require more effort. Therefore the design of the component hierarchy should be done with care.

6. CONCLUSIONS

Application of typing in computer based modelling tools is useful, because it enables hierarchical modelling and it provides for encapsulated, parametrised components. In addition to this, port based interfaces improved the support that modelling systems offer by allowing an object-oriented instead of a process-oriented approach in modelling, and by making it possible to represent models as networks of interconnected components. Modelling tools that incorporate typing and port based interfaces can therewith fully support the decomposition and its representation as is done in bond graph modelling.

However, the classification as common in bond graph models is not supported adequately, due to the fact that component types are monomorphic in these systems, i.e. generic components can not be defined. The use of subtyping and inheritance, such as introduced in Simula and common in object oriented programming, does not solve for this, because the internal structure of components can generally not be abstracted into generic component types. As has been shown, generic components can only be described if subtyping is combined with modularisation. A modularity means that a component is divided into two parts: a type that defines essential properties, and a specification that defines incidental properties. By allowing one type to have more than one specification, component types become polymorphic.

Polymorphic types enable modelling systems to capture classification as is done in bond graphs. Furthermore they result in a hierarchically structured component library. Finally they give modelling systems the possibility to link up with the evolutionary nature of modelling, and to facilitate the creation of alternatives and analogues of models.

Acknowledgements

The authors wish to acknowledge J. van Dijk, A.P.J. Breunese and J. van Amerongen for their valuable comments on the manuscript and their help in implementing polymorphic modelling, and ‘Unilever Research Laboratorium’, Vlaardingen, Netherlands, for financial support.

7. REFERENCES


