

YMIR: A SHARABLE ONTOLOGY FOR THE FORMAL REPRESENTATION OF ENGINEERING-DESIGN KNOWLEDGE

L.K. Alberts

This paper discusses the use of a domain-independent, sharable ontology for the formal representation of engineering design knowledge, called YMIR. The central concept in YMIR is that of a *primitive generic system model* which represents the basic dependencies between the behaviour and the form of a technical system. We believe that it is important to integrate existing formal approaches in the different engineering disciplines as much as possible in modelling design knowledge. To this end, we explain how these primitive generic components can be defined in terms of System Theory. Engineering design, in this context, consists of synthesising complex specific system models by combining and instantiating more elementary generic system models, which in turn are built from primitive models. Furthermore, we present the structures in YMIR for representing knowledge about this synthesis process in terms of system models. Based on these knowledge structure concepts, a basic inferencing strategy for the synthesis of technical systems is proposed.

1. Introduction

This paper discusses the use of a sharable ontology for the formal representation of engineering design knowledge in different design domains, called YMIR. Furthermore, it proposes a basic inferencing strategy for the synthesis of technical systems given the knowledge structures defined in YMIR. Engineering design is regarded as generating a description of a technical system that can be physically realised, given an abstract function to be performed and a set of requirements on the physical properties of the final artifact. The approach taken in the development of YMIR adheres to what is generally known as the ‘product’ school in AI. In other words, our primary aim is not to imitate the process of designing as performed by human designers, but rather to obtain the same, or preferably better results (the products).

One of the major problems in modelling design knowledge is in finding a useful set of concepts that the knowledge should refer to, or, in more fashionable terms, an *ontology*. These concepts should be general enough for describing different types of design knowledge in different design domains, but specific enough to do justice to the particular nature of the task at hand: the design of technical systems. This ontology should serve as the basis for a formal description of the knowledge involved in such a task. Although the notion of an ‘ontology’ is well-known in a number of other disciplines, it only recently gained momentum in the AI community. We will start by giving our interpretation of this notion and the benefits of a domain-independent sharable ontology for engineering design in section 2.

The central concept in YMIR is that of a *primitive generic system model* which represents the basic, generalised dependencies between the behaviour and the form of a technical system. Each engineering design problem inherently implies searching available forms that show particular behaviours, which can be used to obtain a required function. We believe that it is important to integrate existing formal approaches in the different engineering disciplines as much as possible in modelling design knowledge. As Sussman argued ¹ about the need for a reorientation toward rigid mathematical approaches for engineering tasks like design:

“It is better to have a naive theory of sophisticated knowledge, than to have a sophisticated theory of naive knowledge.”

To this end, we explain in section 3 how the relation between behaviour and form of a primitive generic system model can be described in terms of System Theory.

Engineering design, in this context, consists of synthesising complex specific system models by combining and instantiating more elementary generic system models, which in turn are built from primitive models. Furthermore, we demonstrate how the application of a general system model for a particular context can be represented by assigning a function to the model. Finally, the need for representing system models at different levels of *abstraction* is discussed.

Next, in section 4, the structures for representing knowledge about this synthesis process in terms of system models are presented. The knowledge about generating system models can be either fundamental (based on general principles), represented in knowledge structures called *generic components*, or experiential, the knowledge structures for which are *prototypes*. It will turn out that the concept of a function is central in distinguishing fundamental from experiential knowledge.

Based on the available knowledge structures in YMIR, a basic inferencing strategy for the synthesis of technical systems is introduced in section 5.

2. The Benefits of a Shared Ontology

2.1. THE ROLE OF ONTOLOGIES IN KNOWLEDGE ENGINEERING

An important property of an ontology, in our interpretation, is that it gives a precise and unambiguous definition of concepts necessary for representing knowledge, without reference to computational aspects such as data structures, operations on these data structures and control. This puts ontologies at the *knowledge level* (Newell, 1982). As a consequence, considerations which have to do with the nature of the task at hand, for instance design,

¹ In an invited talk during the AAAI Fall Symposium Series in the symposium on Design from Physical Principles (Cambridge, October 23-25, 1992).

can be stated explicitly without confusing them with implementational ingenuities.

Descriptions at the knowledge level can in turn be divided into a conceptualisation and formalisation (Genesereth and Nilsson, 1987). A *conceptualisation* consists of “the objects presumed or hypothesized to exist in the world and their interrelationships”. Here, an object can be anything about which we want to say something (or: ‘have knowledge about’). The *formalisation* of knowledge entails the representation of knowledge about the domain as sentences in a formal language, in terms of the objects and relations in the conceptualisation of that domain. Such a formal language must have a syntax and semantics. The syntax describes the elements of the language and how these elements may be combined to form sentences. The exact relation between the formalisation and the conceptualisation is defined by the semantics, which formally relates sentences and elements of the language to objects and relations of the conceptualisation. An ontology, like a conceptualisation, describes the objects and relations required for the definition of the semantics of the knowledge representation language.

For a formal representation language, the actual *domain knowledge* can be separated from the *inference mechanism*. In the case of predicate logic, for instance, domain knowledge consists of logical sentences and the inference mechanism is defined by the rules of inference (modus ponens or modus tollens, for example). We believe that considerations about the kind of inference mechanism that is required, are not part of the ontology. However, there is a slight ambiguity here, since the knowledge used by an inference mechanism can be further divided into object and strategic knowledge (or: meta-knowledge (Morgado, 1987)).

Basically, the distinction between *object* and *strategic knowledge* separates knowledge concerning the solution of a problem from the knowledge about the problem-solving process, respectively. Here, strategic knowledge ‘guides’ the inference mechanism by specifying which object knowledge should be used where in the process. In our opinion it should not be considered part of the inference engine, which may be a general task-independent algorithm, because it is specific for the problem-solving task at hand. On the other hand, it differs from the object knowledge in that it does not specify information about the problem itself, but about the way the problem-solving process should proceed. It seems appropriate to require that separate ontologies are defined for both levels. The ontology for object-level knowledge defines concepts for the description of the problem domain, whereas the ontology for the strategic level defines concepts related to the problem-solving task. Here, we will focus on an ontology for object design knowledge, and only hint at some of the concepts required for strategic design knowledge.

2.2. YMIR: AN ENGINEERING-DESIGN ONTOLOGY

In sections 3 and 4, we propose a domain-independent shared design ontology, called YMIR ², intended to serve as a basis for the development of knowledge-based systems in different engineering design domains. Before we present YMIR, the need for and benefits of a domain-independent design ontology are discussed.

Since we want to define an ontology for engineering design in general (the task-domain), the concepts defined in this ontology will be abstractions or generalisation of the concepts employed in the individual design domains. In theory, a separate ontology for each of the design domains could be defined with separate tasks of the form “engineering design for domain A” etc. However, we prefer to have a single task, engineering design in general, with a single ontology, that can be ‘instantiated’ for the different design domains.

We want our ontology to be applicable in a wide range of design domains for two main reasons. Firstly, from a knowledge engineering point of view, it allows for the development of a single basic architecture for the knowledge base and inference engine for different design domains. Furthermore, it allows for sharing and reuse of general knowledge that is applicable to multiple domains. Secondly, from the task perspective domain-independency helps to unravel general principles behind design as a phenomenon.

The ontology, instantiated for a particular domain, has to be sharable for all agents (both humans and computer programs) involved in a design process in that domain. The motivation for this requirement can be summarised as the need for (a) a universal vocabulary which serves as an interface between all participating agents; (b) an integrating model for the different views on and aspects of a design; (c) an unambiguous definition of the meaning of all concepts employed. This notion of sharability of the ontology is closely linked to a number of research efforts in the field of concurrent engineering, e.g. (Cutkosky et al., 1993).

YMIR defines a taxonomy of concepts, that we feel are required for the definition of the semantics of a language for representing object knowledge in engineering design. At the bottom of this taxonomy are a number of concepts defining the design artifact models the design knowledge has to refer to. These concepts are formally defined in the field of System Theory. These concepts are in turn used to define *knowledge-structure* concepts for the ‘chunking’ of the design knowledge: dividing the body of knowledge into manageable pieces. These knowledge-structure concepts are still irrespective of the syntactical particularities and the accompanying inference mechanism

² The ontology has been named after the giant and ancestor Ymir, a character in ancient Germanic mythology. Ymir was killed by the gods Odin, Vili, and Ve, after which they created the earth, sky and waters from the different parts of Ymir’s body. By analogy with this legend, the concepts of our ontology are intended as the basic material to construct a ‘world’ of knowledge-based systems for engineering design from.

of the final knowledge representation language.

3. Design Artifact Models in YMIR

3.1. THE MUTUAL DEPENDENCY BETWEEN BEHAVIOUR AND FORM IN DESIGN

In essence, engineering design entails finding a configuration of certain physical elements into a single artifact that can perform the required function. These physical elements have particular geometrical and material properties, called the *form*, and display a certain *behaviour* that is dependent of the form. A *function* is the required part of the combined behaviour of a configuration of elements. ‘Tuning’ the behaviour of a configuration of physical elements to a particular function requires particular values for the forms of those elements. These values should be in accordance with the requirements on the geometrical shape and materials of the final artifact.

The distinction between function and behaviour can be defined as: a behaviour represents a generalisation over a wide range of possible functions. Conversely, a function represents a class of specific instances of different behaviours; these instances all satisfy the requirements specified in that function.

The above description of design has to be further refined for practical situations, because the initial specification of the system to be designed is at a much higher level of abstraction than the descriptions of the available physical elements. This implies that the original specifications have to be ‘translated’ into a set of functions and constraints on the physical characteristics at intermediate levels of abstraction.

In most traditional CAD-approaches, the tasks a) determining the requirements on the combined behaviour imposed by the system specifications, and b) establishing the physical characteristics of configurations of physical elements are treated in isolation more or less. Only at particular points in the design process, the results of each of the tasks are compared and adjusted where necessary. As a consequence, many iterations are needed to find a match between ‘what is required’, given a system specification, and ‘what is possible’, given the available physical elements.

A more fundamental approach to solving this dichotomy is to explicitly represent the *inherent dependencies between the behaviour and the physical characteristics* of the available (combinations of) physical elements and of their abstracted models. In the following sections, we explain how this can be done using a system-theoretical approach.

3.2. NETWORK MODELS IN SYSTEM THEORY

The central concept in System Theory is that of a system. The first property of a system is that it can be any entity that is separable from the rest of the universe (the environment of the system) by means of a physical or conceptual boundary (Karnopp and Rosenberg, 1975). The environment of a system includes everything not defined as that system. Thus, the environment affects the system and is in turn affected by changes in the system. It is important to realise that no ‘real’ system can be modelled exactly. In general, *models* of real systems are used, to predict only certain aspects of the system response to inputs.

The second property of any system is that it can be decomposed into interacting elements. These interacting elements can in turn be described as (sub)systems, where each subsystem is part of the environment of the other subsystems. Systems can be decomposed into subsystems until a collection of primitive parts has been obtained.

The third important property of systems is that the characteristics of a system or subsystem can be determined from the characteristics of the constituent subsystems or parts. This is often modelled as a (sub)system having both “internal” and “external” characteristics (Livesley, 1975). The internal characteristics of a (sub)system can be inferred from the external characteristics of the constituent subsystems or parts. The external characteristics of the compound (sub)system are only concerned with what is observed at the boundaries, where the (sub)system interacts with its environment. The transition from internal to external characteristics of a system is also broadly known as ‘black-boxing’: the compound can in turn be regarded as a system, the internal characteristics of which can be discarded when determining the interaction of this new system with other systems.

TABLE I
Engineering system variables; taken from (Melosh, 1990)

Engineering system	Explicit variable	Implicit variable	Material relation
Structural	displacement	force	Hooke’s law
Thermal	temperature	heat flow	Fourier’s law
Fluid	fluid head	fluid flow	Darcy’s law
Electrical	voltage	current	Ohm’s law
Magnetic	magnetic potential	magnetic flux	Ampere’s law

In general, a description of a system in System Theory consists of a set of equations relating the relevant system variables. These variables represent the important system characteristics, like physical and economical quanti-

ties. The particular type of system models we are interested in is known as *network models* (Macfarlane, 1970). We have chosen for this particular class because of its importance as a widely accepted representation for describing technical systems in many engineering domains. Examples of such network models are bondgraphs (Karnopp and Rosenberg, 1975) and Finite Element techniques (clearly written introductions to this field are (Livesley, 1975) and (Melosh, 1990)).

Network models are based on the conservation of energy in a system. The different subsystems of a system describe particular operations on a ‘flow’, that is related to the flow of energy, through the system. This is often represented as a *network* of interacting subsystems. A *port* models the place at which a (sub)system can be interconnected with (a port of) another (sub)system, and where the flow can enter or leave the (sub)system. For each port of a (sub)system, a pair of conjugate variables is distinguished which together determine the energy flowing into or out of that subsystem. Connecting ports belonging to different systems is equivalent to combining sets of equations (represented by those systems) and appropriately propagating the values of corresponding variables.

The operation of a system is described in terms of the power-defining variables, which can be divided into *explicit* (or: across) and *implicit* (or: through) variables. Examples of the explicit and implicit variables for systems in different domains are given in table I. For each domain, the relations between the explicit and implicit variables, i.e. the behaviour of the system, depend on the geometrical and material properties, i.e. the form, of the system. Connecting systems or parts at their ports implies that the explicit variables belonging to the ports take the same value, and the sum of the implicit variables equals zero. These constraints are called the *boundary conditions*.

In the following subsection we will demonstrate how the relation between the behaviour and the form can be explicitly represented. It is shown how the behaviour and form of a system can be inferred from the behaviours and forms of the constituent subsystems using the boundary conditions.

3.3. GENERIC SYSTEM MODELS

The elements in network models in System Theory, the parts of a system, are defined in terms of elementary operations on the power or energy flow. This suits many of the analysis and simulation tasks in engineering, but seems less ideal for synthesis purposes, the ‘core’ activity in design. To this end we introduce the concept of *generic system models* (GSMs). Complex GSMs can be composed from *primitive* GSMs which cannot be further decomposed, and which describe the basic relations between behaviour and form of technical systems. We will now introduce a formal description of

generic system models in terms of network models in System Theory.

For a particular domain, the *system variables* of a generic system model are defined as the implicit and explicit variables used in network models in System Theory. It is also possible to distinguish *derived system variables*, that can be expressed in terms of system variables, but may have a particular meaning for design. For example, in general the potential or kinetic energy of a system would be a derived variable.

A third type of variables are the *form-related variables*, that serve as parameters in the equations relating the system variables. In terms of the matrix notation used in Finite Element approaches, these are the variables in the expressions making up the stiffness matrix. Examples are area of an object and (modulus of) elasticity of an object's material in structural engineering, that relate forces and displacements in the system description. Analogous with system variables, we can define a fourth type: *derived form-related variables*. For instance, if length and width are form-related variables used in the equations describing the behaviour, the area might be a derived variable if it does not appear in the behaviour.

The reason for distinguishing between 'core' and derived variables is, that it allows for the definition of a *standard description* of a GSM describing the relations between the system variables using the form-related variables. This description can be extended with additional expressions for entities other than the system flows, resulting in an *extended description* for a GSM, which also contains derived variables. For example, the cost of a system may be a mathematical function of the material and geometrical properties, reflected in the form-related variables, or the dissipated energy of the system, in turn derivable from the system variables.

Each variable has a label, a quantity-type, a unit, a value domain, and a time-dependency.

Now that we have defined the relevant variables for constructing the mathematical models represented by GSMs, we will refine these models by distinguishing three aspects, present in each model:

- the *behaviour* of a GSM consists of the equations that describe the relation between the system variables and that may contain any number of form-related variables as parameters. The *extended behaviour* also specifies equations with derived system variables.
- the *form* of a GSM consists of all the geometrical and material-related information contained in the form-related variables of the behaviour. Only those form-related variables that are required to describe the relation between the behaviour of a system and certain form characteristics, are part of the basic form description. All other geometrical or

material characteristics are represented as derived form-related variables. The relations between basic and derived form-related variables can be described in an *extended form*.

- the *structure* of a GSM consists of a list of ports, and specifies the possible connections of a GSM with other GSMs. The ports can be denoted by the pairs of system variables (implicit and explicit) that belong to that port. Each port is uniquely assigned to a particular GSM.

Several examples of the behaviour, structure and form will be presented in the following sections.

Now that we have defined what we mean by a GSM, we are faced with the problem of determining which elements to distinguish as the set of *primitive* GSMs in the different design domains. It is important to realise that primitive GSMs *as a concept* are part of the ontology, but that their *instantiations* for a particular design domain requires domain expertise. Thus, the choice for a particular set of primitives in a domain is not dictated by the ontology. Any set is acceptable, as long as it allows for the description of all systems of interest in that domain, and of the knowledge required for the design of these systems.

3.4. COMPOSING GSMS FROM OTHER GSMS

For a generic system model consisting of a combination of other GSMs, we have to distinguish between *internal* and *external* aspects. The internal structure specifies which connections between GSMs are made, i.e. which ports are connected. The internal behaviour consists of the equations that result from applying the general boundary conditions to the variables at connected ports in the equations representing the behaviours of the constituent GSMs. As a consequence, explicit variables at connected ports are set to equal, and the sum of the implicit variables (including external sources) is set to zero.

The external structure of a system consists of the external ports with which it can be connected to other systems, or which form the boundary connections of the complete system. The corresponding external behaviour consists of an expression for the relation between the system variables at the external ports. Note that this is the same distinction in internal and external descriptions of a system as in the black-box approach in System Theory.

For the form, the distinction internal versus external has a slightly different interpretation. What is important here, is the fact that the form of a complete system also consists of the *totals* of the individual form-related variables of the constituent GSMs, e.g. total length, total weight etc. These accumulations over the values of certain form-related variables require the

introduction of a ‘new’ variable which does not necessarily have a particular meaning for the description of the behaviour of the system. However, totals *are* important for capturing the intuitive interpretation of form and shape. Therefore, they can be added as *derived* form-related variables. Another important use for derived form-related variables, in this context, is in describing ‘emerging shapes’ that arise after combination of individual forms. E.g., three lines that only have a length, introduce the concept perimeter if they are combined in a triangle.

A special type of generic system models are those models of which only the external aspects are known, i.e. nothing is known about the subsystems or primitives the system consists of. Such a model will be referred to as a *generic black-box model*: a generic system model of which only the external behaviour, structure and form are known.

EXAMPLE 1. Suppose we define the rod in structural engineering as a primitive GSM, and want to construct a system consisting of two of these primitives. In this model of a rod it is assumed that the thickness of the rod is negligible compared to the length. Furthermore, only axial forces and displacements are considered. The boundaries of the system are a rigid wall (no displacement) and an external force (see figure 1). These are not part of the system, but define the specific boundary conditions for the complete system. They represent external influences on that system. The relevant system variables are the forces p and displacements d , and the form-related parameters are length l , cross-sectional area A , and modulus of elasticity E .

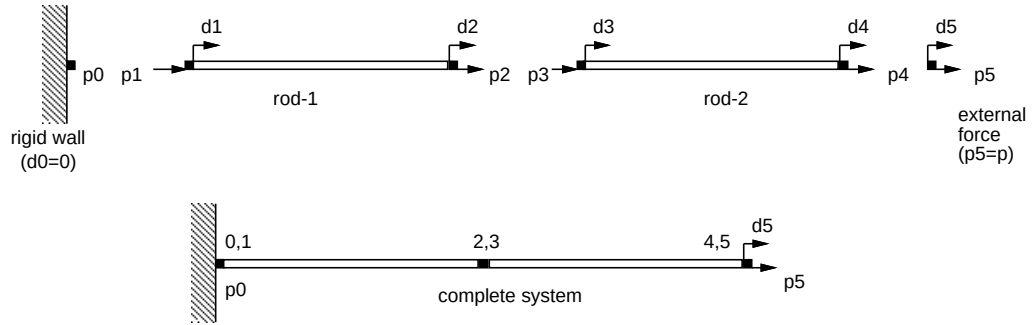


Fig. 1. A system of 2 rods

Using the matrix notation employed in Finite Element approaches, the behaviours of rod-1 and rod-2 are:

$$\begin{bmatrix} k_1 - k_1 & \\ -k_1 & k_1 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \end{bmatrix}, \text{ and } \begin{bmatrix} k_2 - k_2 & \\ -k_2 & k_2 \end{bmatrix} \begin{bmatrix} d_3 \\ d_4 \end{bmatrix} = \begin{bmatrix} p_3 \\ p_4 \end{bmatrix}, \text{ with } k_i = A_i \cdot E_i / l_i.$$

The structures³ are $\{(p_1, d_1), (p_2, d_2)\}$, for rod-1, and $\{(p_3, d_3), (p_4, d_4)\}$, for rod-2. For the description of the form, only A_i and l_i are available for the geometry, and E_i for the characteristics of the material.

For the complete generic system model, the internal structure specifies that port 2 and port 3 are connected: $\{((p_2, d_2) \cap (p_3, d_3))\}$. The external structure defines port 1 and port 4 as the external system ports: $\{(p_1, d_1), (p_4, d_4)\}$.

The internal behaviour of the system can be constructed by applying the boundary conditions that result from the internal structure to the behaviours of rod-1 and rod-2: $d_2 = d_3$, and $p_2 + p_3 = 0$. The resulting internal behaviour is:

$$\begin{bmatrix} k_1 - k_1 & 0 & 0 \\ -k_1 & k_1 & 0 \\ 0 & 0 & k_2 - k_2 \\ 0 & 0 & -k_2 & k_2 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \end{bmatrix} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix}, \begin{matrix} d_2 = d_3 \\ p_2 = -p_3 \end{matrix}$$

The external behaviour describes the relations between d_1 , p_1 , d_4 and p_4 . It follows from expressions for the internal behaviour that $d_2 = (k_1 \cdot d_1 + k_2 \cdot d_4) / (k_1 + k_2)$. Substitution results in the following expression for the external behaviour:

$$\begin{bmatrix} k_1 \cdot k_2 / (k_1 + k_2) - k_1 \cdot k_2 / (k_1 + k_2) \\ -k_1 \cdot k_2 / (k_1 + k_2) & k_1 \cdot k_2 / (k_1 + k_2) \end{bmatrix} \begin{bmatrix} d_1 \\ d_4 \end{bmatrix} = \begin{bmatrix} p_1 \\ p_4 \end{bmatrix},$$

Applying the situation specific boundary conditions $d_1 = 0$, and $p_4 = -p$, results in the following partial instantiation of the external system behaviour:

$$\begin{bmatrix} -k_1 \cdot k_2 / (k_1 + k_2) \\ k_1 \cdot k_2 / (k_1 + k_2) \end{bmatrix} \begin{bmatrix} d_4 \end{bmatrix} = \begin{bmatrix} p_1 \\ -p \end{bmatrix}$$

For the form of the complete system, we may add an extra expression for the total length, $l_{system} = l_1 + l_2$, in addition to the form-related variables from the individual GSMs, where l_{system} is a derived form-related variable.

⊠

³ A structure is represented as a list of ports, defined by their variable-pairs, and connections between ports. The fact that two ports are connected is represented by the symbol ' $\cap$ ' between the pairs of variables belonging to the ports.

3.5. FUNCTION VERSUS BEHAVIOUR

Now that we have defined generic system models as the basic building blocks for modeling a design artifact, we have to define how a generic system model is related to the function of the technical system it represents. A function, in this context, is regarded as a particular application of the general behaviour a technical system displays.

Generic system models describe general behaviours, each of which has a wide variety of possible applications. This generality is reflected in the variables of the equations describing the behaviour. The values of the variables are unconstrained within their domains. For a particular function, these variables can be further *instantiated* with particular values or value ranges. Another possibility is that only a specific *part* of a behaviour is required for a particular application. The variables and parameters in the part which is irrelevant for obtaining the function can be thought to have ‘arbitrary’ values. These values determined by other requirements than those specified in the function. This implies that a *function* is defined as an instantiation of a part of a behaviour or of a complete behaviour. Functions may range from highly generic (applicable to many situations) to very specific.

A particular behaviour may be used for a multitude of functions, by choosing different instantiations and/or parts. Vice versa, a particular function can be realised by many different behaviours. This fact makes it possible to generate a large amount of systems with different functions with only a very limited amount of primitive GSMs.

EXAMPLE 2. A very simple example of a generic function that is often employed in the design of analog circuits, is a system for obtaining integer fractions of some voltage source, drawn in figure 2.

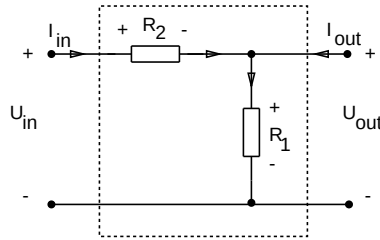


Fig. 2. Taking an integer fraction of a voltage source.

We will apply some short-cuts by specifying potential differences (voltages) instead of individual potentials, e.g. $u_{in} = u_{in}^+ - u_{in}^-$. The internal structure of the system is $\{((u_{in}^+, i_{in}) \cap (u_{R_2}^+, i_{R_2})), ((u_{R_2}^-, i_{R_2}) \cap (u_{R_1}^+, i_{R_1})), ((u_{R_1}^-, i_{R_1}) \cap$

$(u_{in}^-, i_{in}^-), ((u_{out}^+, i_{out}^+) \cap (u_{R_1}^+, i_{R_1}^+)), ((u_{out}^-, i_{out}^-) \cap (u_{R_1}^-, i_{R_1}^-))\}$. For the external structure we get: $\{(u_{in}, i_{in}), (u_{out}, i_{out})\}$.

The internal behaviour can be described as

$$\begin{bmatrix} 1/R_1 & 0 \\ 0 & 1/R_2 \end{bmatrix} \begin{bmatrix} u_{R_1} \\ u_{R_2} \end{bmatrix} = \begin{bmatrix} i_{R_1} \\ i_{R_2} \end{bmatrix}.$$

Since the boundary conditions require that $i_{R_1} = i_{R_2} = i_{in}, i_{out} = -i_{R_2} + i_{R_1}$, $u_{out} = u_{R_1}$, and $u_{in} = u_{R_1} + u_{R_2}$, the external behaviour is

$$\begin{bmatrix} 1/R_2 & -1/R_2 \\ -1/R_2 & 1/R_1 + 1/R_2 \end{bmatrix} \begin{bmatrix} u_{out} \\ u_{in} \end{bmatrix} = \begin{bmatrix} i_{out} \\ i_{in} \end{bmatrix},$$

At this point, we still have to talk about behaviour; except for the boundary conditions no constraints have been put on the domains of the variables. However, if we want to represent that the output voltage has to be an integer fraction of the source (input) voltage, we have to describe this requirement as a function. The relevant part of the behaviour is the relation between u_{out} and u_{in} if $i_{out} = 0$, i.e. with no other components connected to the output port.

It follows from the above matrix equations that $u_{out}/u_{in} = R_1/(R_1 + R_2)$, and since the requirements on the instantiation are $u_{out}/u_{in} = n, n \in \mathbb{N}^+$, it follows that $R_1/(R_1 + R_2) = n$.

The basic information concerning the form of the system, is contained in the variables R_1 and R_2 . It may be extended by introducing a variable for the total resistance of the circuit: $R_{total} = R_1 + R_2$.

⊠

What we cannot describe yet, is how the values of (derived) variables of a system are determined by other, external factors in a particular application described in a function. In the case of System Theory, the latter information is considered part of the environment of the overall system and is only modelled implicitly as some external source acting on the system. However, we want to be able to capture this type of information explicitly in cases where it helps to understand the purpose of a system.

The *purpose* of a system is defined as the function of that system together with the context in which the system is to perform the function. The *context* is that part of the environment of a system, that cannot be represented as an interacting system or in terms of derived variables. A *context description* of a system is defined as a relation between context variables, where a *context variable* may refer to any entity that is deemed an essential characteristic of the context of the system. The description of the *influence* of the context on the system consists of expressions relating the context-variables and the (derived) variables in the system description.

3.6. GENERIC SYSTEM MODELS AT DIFFERENT LEVELS OF ABSTRACTION

In section 3.1 we explained that, in general, that there is an abstraction gap between the initial specification of the system to be designed and the physical elements from which the actual artifact is to be composed. To allow for a gradual refinement in the description of the artifact at intermediate levels of abstraction, we require the availability of generic system models at different levels of abstraction.

Within YMIR, this is incorporated by defining different sets of primitive GSMs for each of the required abstraction levels. The contents of these levels is based on the technical vocabularies in use for a particular domain, therefore we have labeled them *technology-based* layers or levels (TL for short). Each TL represents an abstraction of the levels below, which implies that only certain information is propagated upward from the lower levels. As a consequence, the generic system models of higher levels contain idealisations or approximations from those at lower levels. These idealisations should be made explicit if we want to be able to translate design descriptions from one level to another. Basically, they are reflected in changes in the domain, and/or the time-dependency of the variables.

It is useful to distinguish between abstractions that concern all the GSMs in the layers involved, and abstractions between specific GSMs. A *general abstraction* involves changes in all the variables (whether they be system, form-related or context variables) that refer to a particular quantity type. The changes due to a general abstraction are irrespective of the particular GSM a variable belongs to. For example, the domains of all voltages are changed by abstracting from analog signals to digital signals.

A *case-specific abstraction* implies a translation between individual GSMs in different layers. These mappings involve changes in the relations (describing behaviour, structure and form) between the variables of the relevant GSMs. At the same time, specific variables may be changed in addition to the changes involved in general abstraction. E.g., the domains of the variables in a particular GSM may have to be adjusted to reflect the conditions for which the approximations made in that GSM hold.

Concealing part of the internal structure (and therefore internal behaviour) in a black-box is not considered a reason in itself for introducing another technology-based layer. In contrast, this kind of abstraction might be labeled *aggregation*. In general, a mapping of a design description between two levels will also involve some kind of aggregation in addition to one or more of the types of abstraction mentioned above. For reasons of consistency, we require that the two are conceptually separated: aggregation only happens within a TL, and abstraction, as we defined it, only between TLs.

Besides defining the notion of a technology-based layer, we also have

to define the *hierarchy of TLs* of increasing abstraction from the physical reality. Examples of such TL-hierarchies can be found in (Alberts et al., 1992b) and (Alberts et al., 1992a). Given a particular base level, at which realisable system descriptions can be specified, the rest of the TLs can be defined recursively. At each of these layers, specification of the complete technical system - except for the information ‘hidden’ by the abstraction - should be possible.

This gives rise to the following set of requirements for the introduction of another technology-layer, more abstract than the current one(s): (1) at least one of the changes specified under general abstraction should be involved (the abstraction must entail a reduction), and (2) a complete set of PGCs should be specified, sufficient for describing an entire technical system. Furthermore the abstractions between the TLs should guarantee that a generic system model at lower levels should imply (in the logical sense) the information contained in corresponding models at higher levels. Otherwise, there is no guarantee that the generic system models at the physical level still adheres to the initial design specifications.

3.7. DESIGN IN TERMS OF GENERIC SYSTEM MODELS

At the beginning of this chapter we explained how the design of technical systems can be regarded as generating a realisable description of a system given an abstract functional specification including a set of performance requirements. In this section, more precise definitions of the concepts required for this view of design are given.

A *functional specification* of a technical system is the ‘overall’ function that the system is required to perform. If the function is to be performed in a particular context, this context may also be included in the functional specification. In addition to the required relation between the system variables, constraints on specific individual (derived) system or form-related variables - loosely referred to as performance requirements - may be specified in the functional specification. For example, it may be clear from the start that the power consumption, or the length of the system must be within a certain range.

A *realisable description* of the required system is a description of the instantiated behaviour, form and structure of that system at the lowest TL. These instantiations have to be in accordance with the functional specification after the abstractions involved in the translations between the subsequent TLs.

The synthesis process involves finding a combination of primitive generic system models, the behaviour of which should contain the function specified in the functional specification. Obtaining the required function from the behaviour requires determining the relevant part of that behaviour and in-

stantiation the corresponding variables and parameters with specific values specified in the function. The instantiation should furthermore satisfy the constraints specified in the performance requirements.

Once a description of the required technical system in terms of instantiated primitive GSMs of a particular TL has been obtained, the results have to be translated to a lower TL. Proceeding from one technology-based layer to another entails a translation in terms of the primitive generic system models and the sets of variables at the levels involved. The results obtained at a particular level, after translation, serve as a refined functional specification at the next lower level.

The process of combining and instantiating primitive GSMs, and translating the results to lower levels is repeated until a realisable description has been obtained. In general, each of the steps of this synthesis process will require knowledge to avoid the generation of a prohibitively large amount of unsuccessful combinations, instantiations, and translation of GSMs. The following section is dedicated to this problem.

4. Knowledge Structure Concepts in YMIR

4.1. CHUNKING DESIGN KNOWLEDGE IN TERMS OF GSMs

According to the view unfolded in the previous section, design of technical systems can be regarded as combining primitive GSMs and instantiating them in accordance with the functional specification. In a product-oriented approach, as taken in YMIR, the design knowledge is described in terms of the product: the design description. Since design descriptions are composed of GSMs in YMIR, the knowledge can be described in terms of GSMs as well. Design knowledge, in this context, specifies ‘useful’ combinations, instantiations, and translations of GSMs for construction realisable technical systems that can satisfy certain functional specifications.

An important aspect in *structuring* the design knowledge that has to be described is the division of the body of knowledge into manageable and meaningful pieces. This is often referred to as the ‘chunking’ of knowledge. The possibilities for chunking the knowledge largely depend on the contents of that knowledge. In this section we define how the experience with obtaining functions from primitive GSMs can be described and structured in terms of the concepts defined in section 3. This results in the definition of an additional set of concepts, which are called *knowledge structure* concepts and which also belong to YMIR.

The recursive nature of the synthesis process allows for a separate treatment of the two basic ‘steps’ involved: solving the design problem at a particular TL, and finding a translation of the results obtained at a TL into a problem specification for another TL. In this paper we only discuss the

knowledge structure concepts for the knowledge involved in synthesis at a particular level. For a definition of the knowledge structure concepts for representing the knowledge about translations between different TLs, we refer to (Alberts, 1993).

The primitive GSMs represent *generally* applicable technical principles in each design domain. These technical principles are formally related to the underlying physical and mathematical laws and principles. Such fundamental theoretical knowledge in a domain is often referred to as ‘school-book’ knowledge or first principles (e.g. (Davis, 1983) and (Williams, 1990)). Complex GSMs, consisting of combinations of primitive GSMs, which are still completely generic (i.e.: which have not been instantiated yet) are also considered part of this *fundamental* design knowledge. The knowledge structure concept for this type of knowledge is called *generic component*, and is defined in section 4.2.

In contrast, the knowledge of how to apply this fundamental knowledge to *particular* design problems to obtain a particular function or purpose is based on experience. Such *experiential* design knowledge can be described in terms of *instantiated* combinations of primitive GSMs. Experiential design knowledge is structured in *prototypes*, the definition of which is given in section 4.3.

4.2. GENERIC COMPONENTS

For the synthesis process, the main objective of recognising generic components is to suggest sensible general starting points for synthesising GSMs that can perform a particular function. Three different types of generic components have been defined that allow for a recursive description of the knowledge described in a generic component. These are primitive, blank and compound generic components.

A *primitive generic component (PGC)* forms the smallest possible ‘chunk’ of synthesis knowledge and is defined as a description of a primitive GSM. This description consists of the behaviour, structure and form of the primitive GSM.

A *blank generic component* describes a combination of generic black-box models. This allows for descriptions of how a complex GSM can be obtained by cleverly connecting less complex GSMs without having to refer to the internal characteristics of these constituent GSMs. A blank generic component is defined as a description of a GSM which is a combination of generic black-box models. This description consists of at least: (1) the behaviour, structure and form of the generic black-box models, (2) the internal structure of the GSM, which refers to the constituent generic black-box models, and (3) the external structure of the GSM.

In section 3.4 we explained how the internal and external behaviour, as well as the form of a system can be obtained from the internal and external structure of the system and the behaviours and forms of the constituent PGCs. However, it may be useful to also explicitly specify the external behaviour and form of the generic component, to avoid repeated calculation of that behaviour each time the generic component is selected in the synthesis process.

EXAMPLE 3. A typical application of the feedback principle in electronic circuits is the adjustment of the properties of amplifiers by feeding back some fraction of the output voltage to the input. In terms of blank generic components, this can be described as a feedback-loop consisting of two generic black-box models, one representing the original amplifier circuit (with amplification factor A) and the other the feedback branch (with amplification factor B). The resulting circuit is described as a blank generic component, whose internal behaviour consists of the combined external behaviours of the constituent GSMs. Its external behaviour results from the internal behaviour and the external structure.

This is illustrated in figure 3, where the generic black-box models GSM-1 and GSM-2 represent *all* generic system models that have the specified external behaviour. Note that in the figure the equations describing these external behaviours of GSM-1 en GSM-2 have been graphically represented as *virtual* components. These components do *not* suggest an internal structure of other generic components. For an actual combination of specific GSMs they have to be substituted with the corresponding structure of that combination. In short: instead of drawing a black-box with two ports and specifying the dependencies between the port variables, we have represented these dependencies more conveniently as a network within the black-box.

Instead of using the stiffness matrix, we switch to the network descriptions in use in electrical engineering. This notation is equivalent to the one used in section 3.4, but more convenient for electrical networks in many cases. In this field, the stiffness matrix is known as admittance matrix, the inverse of which is called the impedance matrix Z . A third possibility is to use the so-called transmission or chain matrix T . For the theory behind this phenomenon, we refer to (Baher, 1984), for example.

The external behaviour of GSM-1 - the amplifier circuit that has to be adjusted - is:

$$\left. \begin{aligned} u_1 &= R_i \cdot i_1 \\ u_2 &= A \cdot R_i \cdot i_1 + R_o \cdot i_2 \end{aligned} \right\} \Rightarrow [Z_{GSM-1}] = \begin{bmatrix} R_i & 0 \\ A \cdot R_i & R_o \end{bmatrix}$$

For the external behaviour of GSM-2, $[Z_{GSM-2}]$ is undefined. Instead, we take the transmission matrix $[T_{GSM-2}]$:

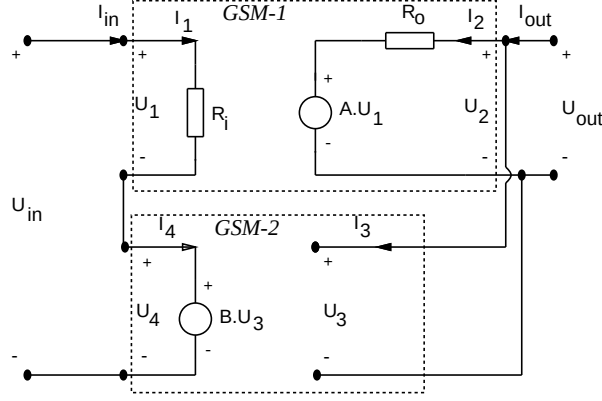


Fig. 3. The feedback principle in amplification

$$\left. \begin{array}{l} u_3 = \frac{1}{B} \cdot u_4 + 0 \cdot -i_4 \\ i_3 = 0 \cdot u_4 + 0 \cdot -i_4 \end{array} \right\} \Rightarrow [T_{GSM-2}] = \begin{bmatrix} \frac{1}{B} & 0 \\ 0 & 0 \end{bmatrix}$$

The external structure of GSM-1 is $\{(u_1, i_1), (u_2, i_2)\}$, and that of GSM-2 is $\{(u_3, i_3), (u_4, i_4)\}$. According to our terminology, $\{A, R_i, R_o\}$, and $\{B\}$ serve as the form-related variables in the behavioural descriptions of GSM-1 and GSM-2, respectively.

For the complete feedback network, the internal structure⁴ is defined by

$$\{((u_{in}^+, i_{in}) \cap (u_1^+, i_1)), ((u_{in}^-, i_{in}) \cap (u_4^-, i_4)), ((u_1^-, i_1) \cap (u_4^+, i_4)), ((u_2^+, i_2) \cap (u_3^+, i_3)), ((u_2^-, i_2) \cap (u_3^-, i_3)), ((u_{out}^+, i_{out}) \cap (u_2^+, i_2)), ((u_{out}^-, i_{out}) \cap (u_2^-, i_2))\}$$

The external structure of the complete system can be described as $\{(u_{in}, i_{in}), (u_{out}, i_{out})\}$. The boundary conditions that can be inferred from the internal and external structure are

$$\begin{array}{ll} u_{in} = u_1 + u_4 & i_{in} = i_1 = i_4 \\ u_{out} = u_2 & i_{out} = i_2 + i_3 \end{array}$$

Based on these boundary conditions and the internal behaviour - which consists of the combinations of the behaviours of GSM-1 and GSM-2 -, we can deduce the external behaviour:

⁴ Here, each port has been decomposed in its two terminals, denoted by + and -. This is a direct consequence of using voltages (potential differences), instead of individual potentials.

$$\left. \begin{aligned} u_{in} &= R_i \cdot (1 + B.A) \cdot i_{in} + B.R_o \cdot i_{out} \\ u_{out} &= A.R_i \cdot i_{in} + R_o \cdot i_{out} \end{aligned} \right\} \Rightarrow [Z_{total}] = \begin{bmatrix} R_i \cdot (1 + B.A) & B.R_o \\ A.R_i & R_o \end{bmatrix}$$

In order to better understand the technical consequences of this use of feedback, it is useful to rewrite the behavioural descriptions as a number of partial relationships. In electrical engineering a system is often ⁵ described in terms of its amplification factor $\mathcal{A} = \frac{u_2}{u_1}$ given $i_2 = 0$, its input impedance $\mathcal{Z}_{in} = \left| \frac{u_1}{i_1} \right|$ given $i_2 = 0$, and its output impedance $\mathcal{Z}_{out} = \left| \frac{u_2}{i_2} \right|_{\substack{i_2=0 \\ u_2=0}}$ given $u_{1,i_2=0} = u_{1,u_2=0}$. Calculation of the three fractions for the complete system shows that

- for the amplifier model: $\mathcal{A} = A$, $\mathcal{Z}_{in} = R_i$, and $\mathcal{Z}_{out} = R_o$, whereas
- for the complete system: $\mathcal{A}' = A/(1 + B.A)$, $\mathcal{Z}'_{in} = (1 + B.A) \cdot R_i$, and $\mathcal{Z}'_{out} = R_o/(1 + B.A)$.

In other words: the result of the feed-back connection of any two systems that show the external behaviours of GSM-1 and GSM-2, respectively, is a new system of which the amplification factor and the output impedance are decreased by a factor $(1 + B.A)$, and the input impedance is increased by the same factor.

⊠

This example illustrates a number of properties of the descriptions of generic components in general. First, alternative representations of the standard description of a generic components are possible, without loss of information. The relationships between these different representations can be formally defined, thus allowing for automatic transformation. This is important, since we want to be able to stay as close to the descriptions used in a particular sub-domain as possible, without the introduction of inconsistencies or loss of generality.

Secondly, extended descriptions make it possible to extend the standard description of a generic component with additional expressions for entities other than the system flows. This allows for the representation of specific partial relationships that are particularly meaningful from an engineering point of view. In the above example, for instance, a lot more could be inferred directly from the descriptions of the input impedance, the output impedance and amplification factor of the components involved, than from the standard description of the behaviours of these components.

Thirdly, the purpose of a generic component for a particular context cannot be represented in the description of a generic component. In the case of the above example, such information typically will consist of the possible

⁵ As a matter of fact, the virtual components used in the graphical representation of the external behaviour of the amplifier model GSM-1, represent exactly these three fractions. The input resistance is equal to R_i , and the output portion of GSM-1 - consisting of A and R_o - represents the Thévenin equivalent of the amplifier.

uses for the result of combining the two systems, e.g. to obtain a system that resembles an ideal amplifier more closely, with a new amplification factor that is approximately $\frac{1}{B}$ - provided $A \gg 1$ - and, therefore, independent of the original amplification factor $\mathcal{A}$. This implies that the resulting amplification factor $\mathcal{A}'$ will be independent of any unwanted transistor characteristics (like dependence on the temperature) that may have been influencing the original factor $\mathcal{A}$. This type of information can be described in a prototype, as will be explained in section 4.3.

Finally, a *compound generic component* describes a combination of generic components. Besides PGCs and blank generic components, other compounds can be incorporated as well. As a consequence, it is possible to describe and structure the fundamental knowledge in a recursive way. This is convenient both for reducing the amount of redundancy in the resulting overall structure of the knowledge and for limiting the complexity of the individual knowledge structure concepts. The description represented by a compound generic component consists of: (1) the constituent generic components, (2) the internal structure of the resulting GSM, which refers to the GSM in the constituent generic components, and (3) the external structure of the resulting GSM.

4.3. PROTOTYPES

4.3.1. Related research

A lot of work is done on the use of prototypes as the basic generic elements for the structuring of design knowledge, e.g. (Murthy and Addanki, 1987), (Addanki and Davis, 1985), (Mittal and Frayman, 1989) and (Gero, 1990). We have taken Gero's interpretation of prototypes - the so-called 'design prototypes' - as a starting point for describing context-dependent knowledge. A design prototype is "a conceptual schema for representing a class of a generalised heterogeneous grouping of elements derived from alike design cases that provides the basis for the start and continuation of a design. Design prototypes provide this basis by bringing all the requisite knowledge appropriate to the design situation together in one schema". (Gero 1990, p. 30). Important for our discussion is the fact that the proposed schema involves the definition of the function, behaviour and form of a design prototype, together with relevant design knowledge.

There are a number of differences between the function, behaviour and form of design prototypes and the properties we defined for generic components. First of all, the definitions are slightly different; e.g. "behaviour" is defined in (Gero, 1990) as: "[a description of] how the structure ⁶ of the

⁶ The term 'structure' in design prototypes refers to what we have called the form.

artifact achieves its function”. In contrast, in a GSM the behaviour forms the basis for the definition of several functions.

The most important difference, however, is the way in which the relevant variables are assigned to either the function, the behaviour or the form. In the case of GSMs, whether a variable is deemed part of the behaviour or the form, is dictated by the ontology. In contrast, the attributes of a design prototype are defined in a more top-down manner. Basically, it starts with determining required functionalities and working back to component behaviours that can fulfill these functions. This decomposition is not dictated by a clear conceptualisation of the domain, however. As a result, there is no bottom-line for the decomposition, and the decomposition as well as the assignment of variables to either the behaviour or the form are ‘ad-hoc’ to a certain extent.

In the next section, we discuss how an interpretation of the notion of design prototypes in terms of GSMs can be defined as an additional knowledge structure concept in YMIR. This concept will be called simply prototype.

4.3.2. *Prototypes in YMIR*

Within YMIR, a *prototype* assigns a function and the corresponding context to a GSM, thereby representing the particular purpose of that GSM. Prototypes are by definition more specific than generic components. The only constraints on the variables and parameters in the expressions describing a generic component are those imposed by the structure. These constraints are a consequence of the boundary conditions that apply when two generic system models are connected. In contrast, prototypes also contain constraints represented in the function that are a consequence of specific choices, based on experience instead of a fundamental property of the GSMs.

A *basic prototype* describes how a particular function can be obtained for a specific context by combining and instantiating GSMs. The result of the combination is again a GSM; the instantiation assigns particular values or ranges to variables in the behaviour, structure and form of that GSM. The function in a basic prototypes describes an instantiation of the *part* of the behaviour of this GSM that is considered relevant for describing the purpose of the GSM. The description of a basic prototype consists of: (1) a function, (2) a context (which may be empty), and (3) the GSM that performs the function. This information suffices to reconstruct how the function is obtained by partially instantiating and combining the general behaviour of the GSM specified in the basic prototype.

EXAMPLE 4. The information concerning the purpose of the feed-back assembly in example 3 can be described in the function of a basic prototype.

The internal and external structure are the same as that of the generic component described in the example.

The function should describe the relevant part of the external behaviour of the assembly. To this end, it is useful to explicitly represent the dependencies that reflect the adjustments to the ‘original’ amplifier GSM-1. We therefore describe $\mathcal{A}'$, $\mathcal{Z}'_{in}$, and $\mathcal{Z}'_{out}$ in terms of $\mathcal{A}$, $\mathcal{Z}_{in}$, and $\mathcal{Z}_{out}$, instead of A , R_i , and R_o . The result is:

$$\mathcal{A}' = \mathcal{A}/(1 + B.\mathcal{A}), \mathcal{Z}'_{in} = (1 + B.\mathcal{A}).\mathcal{Z}_{in}, \text{ and } \mathcal{Z}'_{out} = \mathcal{Z}_{out}/(1 + B.\mathcal{A}).$$

The instantiations required for “ideal” amplifier behaviour are:

$$\mathcal{A}' \approx 1/B, \mathcal{Z}'_{in} \gg \mathcal{Z}_{in}, \text{ and } \mathcal{Z}'_{out} \ll \mathcal{Z}_{out}.$$

⊠

A *blank prototype* describes how a function can be composed of subfunctions, but does not specify how these subfunctions can be obtained from GSMs. The *composition* of subfunctions described by the function of a blank prototype is defined as an instantiation of a part of the combination of those subfunctions. A combination of a collection of functions $f_1, f_2 \cdots f_n$ is defined as the Cartesian product $f_1 \times f_2 \times \cdots \times f_n$ of those functions. This relation between the composite function and the combination of the subfunctions is the same as that between a function and a behaviour. Just as a function describes an instantiated (part of a) behaviour for a particular context, the composite function in a blank prototype may have a context. The description of a blank prototype consists of: (1) the subfunctions, (2) the contexts of the subfunctions (some or all of which may be empty), (3) the composite function, and (4) the additional context information (if specified) for the composite function.

Analogously to compound generic components, *compound prototypes* describe combinations of basic, blank and/or other compound prototypes. However, combining prototypes also requires the composition of the functions in these prototypes into a composite function, besides the combination of the GSMs embodied in the prototypes. This composition is the same as for blank prototypes, including the possibility for adding context information. The description of a compound prototype consists of: (1) the prototypes that are compounded, (2) the composite function, and (3) the additional context information (if specified) for the composite function.

EXAMPLE 5. In structural engineering, different types of trusses can be represented as prototypes, that describe specific instantiations of the generic component ‘truss’ for particular applications. (For a detailed description of applications of trussing in terms of YMIR, see (Alberts, 1993))

Three rods pinned together to form a triangle represent the simplest type of truss. Such a combination can be represented as a generic component, analogically to the one in example 1. The k -factors in the stiffness matrix have to be extended in this case to include the angles between the rods in a truss. The resulting generic component may be used to construct more complex coplanar structures, which can in turn be represented as generic components.

Specific classes of trusses can be distinguished by the requirements on the values of angles between and the lengths of the rods, for instance. Such information represents an instantiation of the general case of a truss. We could, for instance, describe the common types of *Warren*-, *Howe*-, and *Pratt*-trusses as basic prototypes with fairly generic functions.

For a particular design situation, the trussing principles described by these basic prototypes can be further refined in compound prototypes. The refinements consist of additional instantiations of the variables, resulting in more specific functions, and selection of a particular part of the general behaviours of the generic components referred to in the general trussing prototypes. For example, if the common types of trusses are used for the support of roofs, they will have sloping top chords, which further constrains the angles between the individual rods of the trusses. In contrast, for bridge trusses the top and bottom chords are more or less in parallel. Furthermore, the applied loads will be predominantly on the top chords for roof trusses, whereas in bridge trusses these are on the bottom chords.

⊠

The above definition of blank and compound prototypes allows for the construction of hierarchies of prototypes in which the *level of specificity* gradually increases. Multiple hierarchies can be constructed from a limited set of basic and blank prototypes. As a consequence, the knowledge is also structured in accordance with its reusability. The more specific the function in the prototype, the less reusable the knowledge described in that prototype will be.

5. Design as Synthesis Based on YMIR

5.1. THE BASIC OPERATIONS

Now that we have defined the basic knowledge structures for describing experience with the combination and instantiation of primitive GSMs (represented in PGCs), we will discuss the use of such experiential knowledge for synthesising design descriptions. The *basic operations* in reasoning with generic components or prototypes are again instantiation, combination and translation. Proceeding from one TL to another entails a translation between

the vocabularies of PGCs at the levels involved, based on the approximations and assumptions constituting the difference in abstraction. The results obtained at a particular level, after translation, serve as a new and more specific functional description at the next lower level. The basic operations are employed until a realisable description of the design is obtained, that satisfies the functional specification.

Finding a (partially) instantiated composition of PGCs that satisfies the functional specification at the current TL does not guarantee, however, that this solution will also be satisfactory at a lower level. This is a direct consequence of the fact that information on the behaviour and form of a PGC at higher levels only provides an *estimation* of the physical characteristics of the final result. It may appear at lower levels, where the amount of information concerning the exact relation between behaviour and form increases, that the required performance cannot be met after all and certain design decisions have to be reconsidered. This makes the synthesis process inherently iterative. Another difficulty is that a chosen configuration of PGCs at the current level in general can be implemented in several ways by components of the next lower level. This phenomenon is again directly related to the fact that higher-level components are abstractions of (sets of) components at lower levels.

It is clear from the above description of the synthesis process, that at many points in the process well-considered choices have to be made. In general, there will be no simple algorithmic solutions available to guide these choices without running into computational problems. Therefore, we have decided to choose a knowledge-based approach. On the one hand, object knowledge suggests sensible starting points for synthesis from PGCs, either originating from general experience represented as generic components, or from context-dependent experience described in prototypes. On the other hand, strategic knowledge may be added to guide the selection of object level knowledge.

As explained before, a discussion of the strategic knowledge lies outside the scope of this paper. However, even without the specification of strategic knowledge, which is primarily infused by efficiency requirements, it is possible to give a basic inferencing strategy for reasoning with generic components and prototypes.

5.2. ODIN: A BASIC INFERENCING STRATEGY

The synthesis task within a TL can be regarded as finding an instantiated combination of PGCs (i.e. a specific system model) that satisfies the functional specification. Generic components and prototypes entail knowledge about how to bridge the gap between the PGCs and the functional specification, and can be used both in a top-down and a bottom-up fashion.

Top-down means the *selection* and instantiation of a prototype or generic component that matches a required function, as a result of which a decomposition into PGCs, (less complex) generic components or (less complex) prototypes is obtained. Bottom-up implies that PGCs, generic components or prototypes have to be *combined* and appropriately instantiated, the result of which should match a required function. In most practical situations, the actual design process will require both possibilities.

For the basic strategy for synthesis based on YMIR, called ODIN⁷, we assume that the process is roughly carried out in a top-down fashion from the original functional specification to a realisable description of the required system, with intermediate descriptions at all TLs. Furthermore, we employ the heuristic that the most efficient way to use the knowledge is to start with the most specific context-dependent design knowledge and then gradually proceed toward general design knowledge.

Phase A. Synthesis within a TL

Given a functional specification at a particular TL, the synthesis process consists of the following steps.

1. Select and instantiate applicable prototypes

The synthesis process starts with restating the functional specifications in terms of sub-functions, based on previous experience represented in (compound) prototypes and the mathematical characteristics of the functional specs. This greatly helps in focussing the generation of possible solutions, or, in terms of search, in pruning the search-space beforehand. Furthermore, it helps to reduce the complexity of the original problem by cutting it up into more or less independent sub-problems.

Prototypes are particularly useful in this context because they represent classes of functions, defined in the generic function. The available prototypes are tried iteratively in the following sequence (i.e. starting with the most complete information):

1. basic prototypes: a direct mapping onto a generic function is known, as well as a solution in terms of GSMs;
2. compound prototypes that refer to basic or (other) compound prototypes for which a solution in terms of GSMs is known.
3. compound prototypes that refer (also) to blank prototypes: for some of the subfunctions only a functional decomposition is known;
4. blank prototypes: only a functional decomposition is known.

⁷ Odin was one of the three gods, or Aesir, that built the world from Ymir's body. Each of these gods gave particular abilities to the people they created to live in this world called Midgard. Odin bestowed man with vitality and spirit, Vili gave him feeling and mind, and Ve provided man with sensory powers. The basic inferencing strategy based on YMIR has been named after Odin, because it defines just enough of the architecture of a knowledge-based system for engineering design to perform the elementary 'vital functions'.

If all prototypes that apply to the current functional specification have been employed, the result will be a description consisting of primitives, generic black-box models and generic functions, that are instantiated with the values that follow from the functional specification.

2. Select and instantiate applicable generic components

The remaining generic black-box models and functions, for which the required primitives are still unknown, can be matched against the available generic components. This requires the temporary instantiation of generic components with the values that follow from the functional specification and the functions in the prototypes.

Starting with the most complete information about the constituent primitives, the available generic components are tried in the following sequence:

1. PGCs that directly match the refined specs;
2. compound generic components referring to PGCs or other compounds for which a solution in terms of primitive GSMs is known;
3. blank generic components, that specify a decomposition of the intermediate GSM into less complex generic black-box models.

The process iterates through this sequence until a decomposition consisting solely of primitive GSMs and black-box models, both with a partial instantiation, has been obtained.

3. Repeat previous steps until no further refinements are possible

For the remaining partially instantiated generic black-box models, the behaviours of which can be interpreted as generic functions, both the prototypes and the generic components can again be searched for possible decompositions. The whole process is inherently iterative, because after each instantiation of a combination of prototypes or generic components, other prototypes and generic components may become applicable for that particular situation as well.

In case all applicable generic components and prototypes have been tried and applied where appropriate, the process continues with the next step:

4. Combine and instantiate PGCs for remaining black-box models

It can be foreseen that after all available generic components and prototypes have been tried, the design description will still contain generic black-box models. In that case, a bottom-up synthesis process can be initiated, which blindly searches for *combinations* of instantiated prototypes, generic components and PGCs with the partially instantiated black-box models. Preferably, some strategic knowledge should guide this generation process.

Phase B. Synthesis between TLs

The result of the previous phase in the synthesis process is a system model consisting of a combination of instantiated PGCs at the current TL. Using the abstraction knowledge (see (Albert, 1993)), this system model is

translated to the next lower TL. The process is then repeated from phase A.

5.3. CLASSES OF KNOWLEDGE-BASED SYNTHESIS

Based on the amount of available information regarding the decomposition into PGCs and their instantiation, we can classify the different ways in which the above synthesis process may result in a realisable description, as follows:

- routine synthesis: by combining and instantiating prototypes for which a solution in terms of primitive GSMs is known
- innovative synthesis: by combining and instantiating generic components and blank prototypes
- creative synthesis: by combining and instantiating PGCs ‘from scratch’
- inventive synthesis: the process requires the generation of new PGCs.

The latter class of synthesis can hardly be called design anymore, since it assumes that new engineering principles can be defined somehow. In general, the development of primitive generic components like a transistor in electrical engineering, or a I-beam in structural engineering, takes years of research and a major technological breakthrough. For most practical design problems, the first three classes will all be needed to a certain extent. Depending on whether the emphasis is on routine or on creative synthesis, we might classify a design process as ‘routine’, ‘innovative’ or ‘creative’, respectively. This classification roughly coincides with those proposed in (Brown and Chandrasekaran, 1989) and (Gero, 1990).

6. Conclusions, Status, and Further Research

In this paper, we have concentrated on the theoretical basis for YMIR, our ontology for the formal representation of engineering-design knowledge, and ODIN, the basic inferencing procedure for synthesis based on YMIR. The practical application of YMIR is illustrated in detail elsewhere (Alberts, 1993).

In conclusion, YMIR defines the necessary concepts for the ‘chunking’ and formal representation of engineering design knowledge in terms of generic system models. These models are based on a solid theoretical foundation provided by System Theory. The knowledge structures in the ontology allow for the incorporation of both first principle knowledge, represented in primitive generic components, and experiential knowledge about the application of these primitives. The level of ‘specificity’ of the knowledge gradually increases from PGCs via generic components to prototypes. This allows for all major classes of synthesis, namely: routine, innovative and creative. The ontology and corresponding basic synthesis strategy are applicable to all engineering design domains that allow for a system-theoretical formulation of

the technical system to be designed. As a consequence, the same basic architecture for knowledge-based design systems in each of the technical domains can be used, and the reusability and sharability of the acquired knowledge can be increased.

We are currently implementing a knowledge representation ‘shell’ on the basis of YMIR, using Quintus-Prolog, C and Devguide (a graphical interface) on a network of Sparc-1 workstations. This shell will be interfaced with tools for the manipulation and calculation of the behavioural expressions, e.g. Abacus and Mathematica, and tools for the graphical representation of the form information, such as AutoCad. In the mean time, we continue working out design cases for different domains ‘on paper’. In particular, we are reconstructing cases and design heuristics in bridge design, analog circuit design and compressor design in terms of the concepts in YMIR. Using the resulting knowledge for synthesising new designs should provide us with more insight into the required strategic knowledge.

Acknowledgements

The research described in this report is part of the STEVIN-project, a research programme aimed at supporting the design of technical systems, at the University of Twente. I would like to thank Frank Dikker, Nel Wognum, Nicolaas Mars, and Frank Tempelman, my colleagues in the STEVIN-project, for their contributions and support.

References

- Addanki, S. and Davis, E.: 1985, A representation for complex physical domains, *Proc. IJCAI'85*, pp. 443-446.
- Alberts, L.K.: 1993, *YMIR: an ontology for engineering design*, PhD-thesis, University of Twente, Enschede, The Netherlands.
- Alberts, L.K., Wognum, P.M., Mars, N.J.I.: 1992a, Structuring design knowledge on the basis of generic components, in Gero, J.S. (ed), *Artificial Intelligence in Design '92*, Kluwer, Dordrecht, pp. 639-656.
- Alberts, L.K., Wognum, P.M., Mars, N.J.I.: 1992b, A multi-level primitive generic components model, in Tzonis, A. and White, I.(eds), *Automation Based Creative Design (ABCD): Current Issues in Computers & Architecture*, Elsevier.
- Baher, H.: 1984, *Synthesis of electrical networks*, John Wiley & Sons, Chichester.
- Brown, D.C., and Chandrasekaran, B.: 1989, *Design problem solving: knowledge structures and control strategies*, Morgan Kaufmann, San Mateo, CA.
- Cutkosky, M.R., Englemore, R.S., Fikes, R.E., Genesereth, M.R., Gruber, T.R., Mark, W., and Tenenbaum, J.M.: 1993, PACT: An experiment in integrating concurrent engineering systems, *IEEE Computer*.
- Davis, R.: 1983, Reasoning from first principles in electronic troubleshooting, *Int. J. Man-Machine Studies*, **19**, pp. 403-423.
- Genesereth, M.R., and Nilsson, N.J.: 1987, *Logical foundations of artificial intelligence*, Morgan Kaufmann, Los Altos.

- Gero, J.S.: 1990, Design prototypes: a knowledge representation schema for design, *AI Magazine*, **11**(4), pp. 26-36.
- Karnopp, D. and Rosenberg, R.: 1975, *System Dynamics: a unified approach*, John Wiley & Sons, N.Y.
- Livesley, R.K.: 1975, *Matrix methods of structural analysis*, Pergamon Press, Oxford.
- MacFarlane, A.G.J.: 1970, *Dynamic system models*, George G. Harrap & Co. Ltd, London.
- Melosh, R.J.: 1990, *Structural engineering analysis by finite elements*, Prentice-Hall, Englewood Cliffs, N.J.
- Mittal, S. and Frayman, F.: 1989, Towards a generic model of configuration tasks, *Proc. IJCAI'89*, pp. 1395-1401.
- Morgado, E.: 1987, Meta-knowledge, meta-rules and meta-reasoning, in Shapiro, E. (ed), *Encyclopedia of AI*, p. 598-603.
- Murthy, S.S., and Addanki, S.: 1987, PROMPT: An innovative design tool, *Proc. AAAI'87*, pp. 637-642.
- Newell, A.: 1982, The knowledge level, *Artificial Intelligence*, **18**, pp. 87-127.
- Williams, B.C.: 1990, Interaction-based invention: designing novel devices from first principles, *Proc. 2nd AAAI Workshop Model Based Reasoning*, AAAI, Boston, MA, pp. 168-175.