Publication I


© 1998 Cambridge University Press.  
Reprinted with permission
Towards a general ontology of configuration

TIMO SOININEN, JUHA TIIHONEN, TOMI MÄNNISTÖ, AND REIJO SULONEN
Helsinki University of Technology, TAI Research Centre and Laboratory of Information Processing Science, Helsinki, Finland

(Received October 31, 1997; Accepted February 5, 1998)

Abstract

This article presents a generalized ontology of product configuration as a step towards a general ontology of configuration, which is needed to reuse and share configuration knowledge. The ontology presented consists of a set of concepts for representing the knowledge on a configuration and the restrictions on possible configurations. The ontology is based on a synthesis of the main approaches to configuration. Earlier approaches are extended with new concepts arising from our practical experience on configurable products. The concepts include components, attributes, resources, ports, contexts, functions, constraints, and relations between these. The main contributions of this work are in the detailed conceptualization of knowledge on product structures and in extending the resource concept with contexts for limiting the availability and use of resources. In addition, constraint sets representing different views on the product are introduced. The ontology is compared with the previous work on configuration. It covers all the principal approaches, that is, connection-based, structure-based, resource-based, and function-based approaches to configuration. The dependencies between the concepts arising from different conceptualizations are briefly analyzed. Several ways in which the ontology could be extended are pointed out.

Keywords: Configuration Design; Product Configuration; Ontology; Conceptual Analysis; Knowledge Representation

1. INTRODUCTION

Configuration has been a fruitful topic of research in artificial intelligence for the past two decades. In the last five years, configuration has also become a commercially successful and important application of artificial intelligence techniques. Configuration as a task can be roughly defined as the problem of designing a product using a set of predefined components while taking into account a set of restrictions on how the components can be combined. The term product configuration is used to denote the routine engineering activity of this type in the sales-order-delivery process. The term configuration design may also encompass more design-oriented activities that take place in the product development process.

Most of the research on configuration has concentrated on problem-solving methodologies, such as constraint satisfaction (e.g., Mittal & Falkenheiner, 1990), resource-based configuration (Heinrich & Jüngst, 1991), and propose-and-revise type approaches (e.g., Balkany et al., 1993). Despite the research on various problem-solving techniques, a general ontology of configuration domain has not emerged. We think that defining a general ontology of configuration (Gruber, 1995) is an equally important research issue.

The importance of a general ontology of the configuration domain lies in facilitating reuse of the knowledge on configurable products and cooperation between different configurator agents based on different problem-solving methodologies in configuring a product (Wielinga & Schreiber, 1997). An ontology can also be used as a basis for documenting configuration knowledge in an easy-to-understand form. We believe the research on configuration would benefit from the common ground provided by a general ontology.

This paper aims at providing some steps on the road towards a general ontology of the configuration domain. The ontology presented synthesizes and extends earlier approaches. It contains a detailed conceptualization of the compositional structure of a product. The ontology integrates resource-based concepts to other approaches through extending the resource concept with contexts. Unlike earlier work, it also treats the concepts uniformly with respect to...
refinement and abstraction. Constraint sets are introduced to capture different points of view on the correctness of a configuration.

The ontology presented is a unified informal ontology for configuration domain. It is by no means the only correct ontology or necessarily general enough to capture all the relevant knowledge. We take a product-configuration point of view. This means that the ontology may not cover all the things required for configuration design. The ontology does not cover the knowledge related to the geometry, pricing, or optimality of configurations, either. These types of knowledge are often important for product-configuration tasks. Nor does the ontology include the construction and control knowledge on how to accomplish the configuration task, that is, what actions and in which order can be taken to configure a product (Cunis et al., 1989).

In Section 2, we discuss the method for specifying the ontology. In Section 3, we present the basic definitions of the ontology. The different concepts in the ontology are discussed in Section 4. Our ontology is discussed in relation to previous work in Section 5. Finally, in Section 6 conclusions are given and topics for future work are discussed.

2. METHOD

The earlier conceptualizations of configuration knowledge can be roughly classified to connection-based (Mittal & Frayman, 1989), resource-based (Heinrich & Jüngst, 1991), structure-based (e.g., Cunis et al., 1989), and function-based (Najman & Stein, 1992) approaches. The approaches have presented more or less explicitly a set of method-specific concepts for representing product knowledge needed in configuration problem solving. The conceptualizations have little in common except the central notion of a component.

Our aim is primarily to provide an ontology for the required forms of knowledge and its representation in a computer and only secondarily to provide computer support for configuration tasks. The concepts in the ontology have not been influenced by considerations of computational complexity of configuration task, although this is an important research issue on its own. Our ontology tries to be as method-independent as possible.

We try to synthesize the four approaches mentioned above, as they all seem to have gained at least some acceptance in the research community or have been used in commercial applications. Our own experiences on configurable products have led us to believe that all four types of modelling concepts are needed to compactly and adequately represent the knowledge on products (Tiihonen, 1994; Tiihonen et al., 1996).

When discussing an ontology, it is necessary to separate entities that occur in a configuration model or configuration of a particular product from meta-entities that are used to describe what the entities are. We base our ontology on the Frame Ontology of Ontolingua (Gruber, 1992). We next briefly describe the Frame Ontology. The basic concepts in Frame Ontology are classes, instances, relations, and functions. These can be defined to have properties, such as instances of a particular class are always in some relation or a particular relation is transitive. Classes collect together the definitions that apply to all their instances. A class $C_1$ is said to be a subclass of class $C_2$ if and only if every potential instance of $C_1$ is also an instance of $C_2$. The subclass-relation induces a taxonomy in which the definitions are monotonically inherited. A superclass-relation between classes is defined as the inverse of subclass in the obvious way. An instance $I$ is said to be a direct instance of a class $C$, if there is no class $C_2$ such that $C_2$ is a subclass of $C_1$ and $I$ is an instance of $C_2$. A direct subclass is defined similarly. For a more detailed account of Ontolingua, we refer to (Gruber, 1992).

3. FOUNDATIONS OF THE CONCEPTUALIZATION

Our ontology for configuration knowledge, that is, the knowledge that people have on configurable products, is based on the knowledge needed when configurable products are defined and configuration problems are solved. We distinguish between the following three classes of configuration knowledge. They exist independently of the problem solving methods.

Configuration model knowledge specifies the entities that may appear in a configuration, their properties, and the rules on how the entities and their properties can be combined. Configuration model knowledge contains all the information for tailoring the product to meet given requirements. In other words, configuration model knowledge specifies the set of correct configurations of a product with respect to the configuration model and requirements.

Configuration solution knowledge specifies a (possibly partial) configuration. A configuration specifies in sufficient detail what a real-world product instance must be like. A partial configuration leaves some aspects about the real-world product instance open.

Requirements knowledge specifies requirements on the configuration to be constructed. Requirements knowledge can in our view be specified with the same concepts as configuration model knowledge and configuration solution knowledge, although it plays a different role in problem solving (Wielinga & Schreiber, 1997). For this reason, we explicitly discuss only configuration model knowledge and configuration solution knowledge.

An ontology tries to define a set of concepts to represent knowledge. An ontology is needed to document the knowledge, to store it into a computer, or to make inferences. Our

\footnote{We omit here the precise definition of a “correct” configuration. For a more detailed analysis of the correctness of a configuration with respect to a subset of this conceptualization, we refer to Peltonen et al. (1998).}
The ontology of configuration knowledge has the following basic structure (Figure 1). We define a set of configuration model concepts as a top-level taxonomy of configuration specific classes and a set of configuration specific relation definitions. Distinguished configuration domain specific classes include constraint and property definition.

A configuration model of a particular product is defined as a set of product specific classes and a set of constraint and property definition instances. The product specific classes are direct subclasses of the configuration specific classes. We call these types to distinguish them from generic Ontolingua classes. Since we are interested in providing information system support for representing the knowledge, a representation of a configuration model as well as a configuration must be finite.

A configuration of a product with respect to a given configuration model is defined as a set of instances of the types occurring in the configuration model. We call such instances individuals to separate them from the generic Ontolingua instances. In addition, a configuration includes configuration specific relations, called properties, between the individuals.

Note that configuration specific classes do not occur in a configuration model of a particular product. Direct instances of configuration specific classes do not occur in a particular configuration, either. Unless otherwise stated, the configuration specific classes introduced are considered disjoint, that is, their instances are two disjoint sets. The configuration specific subclasses also cover their superclasses. This means that there are no instances that are instances of the superclass but not instances of any of its subclasses. Note that types are not required to cover their supertypes.

Individuals in a correct configuration (with respect to a given configuration model) must have the properties specified by their types. In this case, we say that an individual is valid with respect to its type. A correct configuration must also satisfy the constraints in the configuration model. When describing the concepts, we often use the words “must”, “is”, and “may” to specify certain restrictions. The intended meaning of these restrictions is that they hold in a correct configuration. If these restrictions do not hold, a configuration is incorrect. For simplicity, we do not consider the validity of types and their property definitions. We assume that the types and their properties in a configuration model are valid with respect to the classification taxonomy in the configuration model, that is, that the properties are inherited properly and that all the subclass-relations are explicitly presented.

The concepts of types, property definitions, constraints, individuals, and properties of individuals are defined in more detail in the next section. Figure 2 gives an overview of the classes and the taxonomy of the ontology. We make the typographical convention that the concepts in our ontology are typeset in small capitals when they are defined.

### 4. Concepts

#### 4.1. Taxonomy

In this section, we give basic definitions for the concepts in configuration model knowledge and configuration solution knowledge. We introduce the concepts of a type and an individual to distinguish between the entities that occur in configuration model knowledge and configuration solution knowledge. Types are further organized in classification...
taxonomies to enable compact representation of the knowledge common to different types, which is important for maintaining the knowledge. The class configuration type is introduced to gather the definitions common to some of the classes in our ontology. Direct subclasses of configuration type do not occur in a configuration model.

Configuration knowledge is commonly discussed using terms that do not distinguish between configuration model knowledge and configuration solution knowledge. For example, the sentence “Car has an engine as a part” can be interpreted in two ways. As configuration model knowledge the sentence can be understood as saying that every car individual must have an engine individual as a part. As configuration solution knowledge, it states that a configuration includes a car individual that has an engine individual as a part. We want to clearly separate these meanings by using the terms type and individual. Subclasses of configuration type, that is, types, occur in the configuration model. A type specifies intensionally the properties that all valid individuals of that type must have through property definitions. An individual is an entity that occurs in the configuration.

Types have a classification taxonomy with inheritance defined by isa-relation between types. The semantics of the taxonomy and inheritance are related to the direct-subclass-of-relation of Ontolingua as follows: \((T_1 \text{ isa } T_2)\) implies \((T_1 \text{ direct-subclass-of } T_2)\). We distinguish between the direct isa-relation and the transitive isa-relation similarly as the direct-subclass-of and subclass-of relations in Ontolingua. We refer
to subtypes and supertypes similarly as in Ontolingua to subclasses and superclasses. Unlike in Ontolingua, the taxonomy is explicitly represented through the isa-relation and “incidental” subclass-relations are not allowed. That is, \((T_1 \text{ direct-subclass-of } T_2)\) does not necessarily imply \((T_1 \text{ isa } T_2)\). We also require that the taxonomy is well-formed in the sense that it does not contain cycles. In other words, the transitive closure of the isa-relation must be antisymmetric and anti-reflexive. Note that we allow multiple inheritance.

A subtype may refine the property definitions it has inherited. Refinement of property definitions is semantically based on the notion that the set of potential valid individuals directly of the subtype is smaller than the set of valid individuals directly of the supertype.

An individual is directly of exactly one type. Similarly as in Ontolingua, an individual is indirectly also of all supertypes of its type. A valid individual of a type is defined to be directly of the type and an instance of the type (in Ontolingua terms).

Each type is either abstract or concrete. This is specified through an abstraction definition. An individual directly of a concrete type is accurate enough to be used in an unambiguous configuration. An individual directly of an abstract type provides only partial information on the real-world entity it represents. Abstract types naturally emerge when the knowledge common to a set of types are gathered in a supertype. A configuration that includes individuals directly of abstract types must be refined so that a final configuration contains only individuals directly of concrete types. It is a modelling decision which types are considered abstract and which concrete. Note that we do not assume that a valid individual of a concrete type would have fixed attribute values, for example. A concrete type may still allow variation in the properties of the individual. The abstraction definition is not inherited. It defines a property of direct individuals of the type only.

Each minimal (leaf) element in a taxonomy must be concrete. If there were an abstract type as minimal element in the taxonomy, then no individual of that type could ever be used in an unambiguous specification of a product.

**Example.** We shall use a running example for demonstrating the concepts we define. The sample product, Hospital Monitor (HMonitor), is loosely based on a successful real-word configurable product for clinical measurements. We have simplified the product considerably and altered it to emphasize some aspects of it.

Figure 3 shows the taxonomy of the types in the configuration model of the monitor. The arrows denote the isa-relation between different types. Some properties of the types are given in brackets next to the type. Note that for instance the type Module is abstract and therefore an individual directly of module may not occur in an accurate configuration. Such an individual must be refined to be of one of the concrete subtypes of Module. The meaning of the “ind” and “dep” definitions are explained in Section 4.3.

### 4.2. Attributes

Attributes represent the characteristics of the type. Types have attribute definitions that specify the possible values that attributes of valid individuals may have.

A configuration type specifies a set of attributes by means of an attribute definition, which consists of an attribute name, an attribute value type, and a necessity definition. This is an example of a property definition. When a configuration type has an attribute definition, a valid individual of the configuration type has an attribute according to the attribute definition and either exactly one (necessary attribute) or at most one (optional attribute) attribute value of the attribute type. One particularly important attribute type is physical quantity \(\text{Gruber}, 1992\) such as length or mass. As usual, attribute value types are considered classes and attributes are considered relations between individuals and instances of attribute value types \(\text{Gruber}, 1992\). A configuration model must define the instances that an attribute value type has. To represent an attribute without a value, the special instance undefined must be included in every attribute value type. The attribute value type or necessary/mandatory definition defined in a supertype may be refined in a subtype.

**Example.** Figure 4 presents property definitions of the types in our sample configuration model. In the property definitions, we typeset the names of concepts and other reserved words in bold. We abbreviate some of the terms to save space. The property definitions are presented using a very simple syntax. This syntax is by no means intended for real modelling tasks\(^3\). The syntax consists of sentences which are defined as a sequence of a type name, name of the definition given, and a definition delimited by brackets. A sentence is ended by a semi-colon. The definition may consist of several parts separated by commas. We use curly braces to indicate elements of definitions that are sets. Under “Taxonomy”, two attributes color and surface material are defined for the monitor. Attribute types and their attribute value instances are also defined. An individual of HMonitor must have a value for both the attributes since they are necessary. One can also use standard attribute value types such as integers, which are assumed to have an appropriate standard definition. The simple constraint \(c1\) associated with HMonitor restricts the combinations of the attribute values so that one cannot have a monitor with material plastic and color metal.

### 4.3. Structure

In this section, we define component types, component individuals, and their compositional structure as the basic con-

---

\(^3\) A real modelling language should, in the tradition of object-oriented languages, collect all the definitions pertaining to one type under one type-definition to facilitate easier maintenance of the knowledge. We use this type of syntax in the example to follow more closely the order in which concepts are introduced.
cepts for representing configuration model knowledge and configuration solution knowledge. The compositional structure is important for configuration because products are commonly described through their structure for design, manufacturing, or maintenance purposes.

A COMPONENT TYPE represents a distinguishable entity in a product that is meaningful for product configuration in the sense that a configuration is composed of COMPONENT INDIVIDUALS of component types. Component type is a sub-class of configuration type.

A component type specifies the parts of a valid component individual of the component type through a set of PART DEFINITIONS. The parts are realized with component individuals. A part definition specifies a PART NAME, a SET OF
POSSIBLE PART TYPES, A CARDINALITY, AN EXCLUSIVITY DEFINITION, AND A HAS-PART INHERITANCE DEFINITION. When discussing a particular part definition, we refer to the component type that defines its parts as the whole type.

A component type may occur as a part type in several different part definitions of one component type under different part names or in several different part definitions of different component types. A component type may refine its inherited part definitions by restricting the set of part types, the cardinality or has-part inheritance definition.

A part name distinguishes the role in which different component individuals are parts of a component individual of whole type. The part types indicate the component types whose component individuals are allowed to occur as parts with the associated part name. A cardinality specifies how many component individuals must occur as parts with the part name.

The exclusivity definition specifies that a part is either exclusive or shared. A valid exclusive part, that is, a component individual occurring as a part with a name that in-
In the example all component types are physical or logical entities and therefore exclusive, so we omit the exclusivity definitions (Figure 3a). Only the component type HMonitor is defined as independent (in the definition next to the component type in the figure). Therefore, only an individual of HMonitor can occur as a root of the has-part hierarchy of component individuals. HMonitor has the parts with part names base part, primary display, secondary display, secondary-display-adapter, and module (see “Structure” in Figure 4). Base part is of type base-unit and an individual of HMonitor must always have one base part. The primary display may be of types Disp-10 or Disp-14, that is, a 10-inch or 14-inch display. HMonitor may have one or two primary displays. The secondary display adapter and secondary display are optional parts of HMonitor, since the cardinality definitions for these includes the value zero.

The constraints c3 and c4 on possible connections under the heading “Topology” in Figure 4 demonstrates the need to refer to a certain part by its part name, without knowing the component type that the actual part will be an individual of.

None of the component types have has-part inheritance definitions, so we omit these from the definitions. An example of a has-part inheritance definition in the context of this example could be that Module also had a color attribute definition and the value of this attribute would have to be the same as the color of the monitor.

4.4. Topology

In this section, we define the topological concepts of a port type and a port individual that represent how component individuals can be connected together to form a working product (Mittal & Frayman, 1989). Connections can be physical or logical. Port definitions represent the compatibility of component individuals and specify the possible topologies of the product.

A port type is an intensional definition of a connection interface. Port type is a subclass of configuration type. A port individual represents a “place” where in a component individual some other port individual may be connected to. A port type has a compatibility definition that defines a set of port types whose port individuals can be connected to the port individuals of the port type. In addition, a port type defines a set of connection constraints. Only port individuals that satisfy the connection constraints defined by their port types may be connected to each other. A connection constraint may also specify that port individuals with given attribute values and of particular type must be connected. Connection constraints are a special case of constraints (see Section 5.7) that refer only to port and component types, their individuals and their attributes and attribute values.

A component type specifies its connection possibilities by port definitions. A port definition specifies a port name for the port, a set of possible port types, a cardinality, and connection constraints. A port definition can refine the set of compatible port types from those specified by the port type or those specified by a supertype of the component type. It can also specify additional connection constraints. Cardinality may also be refined.
Example. In our example there are two abstract port types, Video-in and Video-out (Figure 3b). These have concrete subtypes for standard resolution displays (VGA) and high-resolution displays (XGA). Video-out is defined to be compatible with Video-in and vice versa (“Topology” in Figure 4). They have no connection constraints. Thus, any port individuals directly of these types may be connected. VGA-out inherits the compatibility with Video-in from Video-out and XGA-in inherits the compatibility with Video-out from Video-in. The compatibility definition is refined for the XGA-out so that it is only compatible with XGA-in. Similarly, VGA-in refines the compatibility to VGA-out.

In effect, an XGA-out port individual can be connected to an XGA-in port individual only, a VGA-out port individual to individuals of both Video-in subtypes, an XGA-in port individual to port individuals of both Video-out subtypes, and an VGA-in port individual only to a VGA-out port individual.

An individual of base-unit has two display-out port individuals of VGA-out type. There is a connection constraint c3 attached to this port definition that if there is a base-unit and a primary display in the configuration, then a display-out of base-unit and the signal-in of primary display must be connected. An XGA-display adapter similarly has one display-out of XGA-out type, which must be connected to the signal-in of the secondary display (constraint c4). As implied above, a display unit has one signal-in port of type Video-in. This port definition is inherited by the 10-, 14-, and 21-inch display component types and refined to either VGA-input (10- and 14-inch) or XGA-input (21-inch). The fact that a property specification is a refinement and does not introduce a new property is implied by using the same property (part, port, etc.) name as in a supertype. These definitions together state that a hospital monitor has one or two VGA primary displays (10- or 14-inch) connected to the base-unit and possibly a secondary XGA (in this example the 21-inch only) display connected to the XGA display adapter.

A port definition specifies that a valid component individual of the component type has the number of port individuals specified by the cardinality with the given port name. The port individuals must be of the appropriate port types. This is conceptualized as the component individual, the port individual, and the port name being has-port-relation. A port individual cannot exist in a configuration without being associated with a component individual.

Two component individuals that have port individuals of compatible port types may be connected to each other through their port individuals if the port and component individuals satisfy the connection constraints. This is conceptualized as the two port individuals being in connected-to-relation. Two component individuals are connected to each other, represented by them being in the connected-to-relation, if some of their port individuals are in the connected-to-relation. At most, one port individual can be connected to a port individual. Both connected to-relations are symmetric and antireflexive to prevent a port individual and a component individual from being directly connected to itself. The intended meaning is that port individuals represent the places at which two different component individuals can be connected to each other. Port individuals of one component individual cannot be connected to each other without another component individual and its attendant port individuals.

Similarly to has-part-relation, it is sometimes necessary to refer to the component or port individuals that are transitively connected to each other, that is, that are in the transitive closure of the connected-to-relation.

4.5. Context

In this section, we define a natural way of restricting the applicability of certain pieces of knowledge to certain “parts” of the configuration. This restriction is defined as a context in which the piece of knowledge holds. The context concept will be used in Section 4.6 to enrich resource-oriented concepts.

When specifying a configuration model, often some things in the model are true only in a certain context. We define a context as a set of component individuals in a configuration. There are three natural contexts that arise from the concepts introduced above:

- **Component type set context**, defined as the set of individuals in a configuration that are of the component types in a given set of component types.
- **Mereological context**, defined as consisting of a component individual and the set of component individuals that are its parts.
- **Topological context**, defined as consisting of a component individual and the set of component individuals that are connected through port individuals to it.

We distinguish between an **immediate mereological context** and a **transitive mereological context**. An immediate mereological context contains the component individuals that are direct parts of the component individual that specifies the context. A transitive mereological context contains also the component individuals that are transitively parts of the component individual that specifies the context. Two individuals of the same type are not necessarily in the same mereological context, as they need not be parts of the same larger whole. Analogously to the mereological contexts, we define an **immediate topological context** and a **transitive topological context**.

A context is specified by a context definition. A context definition consists of a set of component types (component type set context), or a component type and the definition whether the parts or connected component individuals of the component type form the context, and whether the context is immediate or transitive. These are referred to
as *atomic contexts*. A context specification may also be a union or intersection or complement with respect to each other of any number of atomic contexts. The *union of two contexts* is the context formed by the union of the sets of component individuals in the two contexts. *Intersection and complement of two contexts* are defined similarly.

### 4.6. Resources

In this section, we define the resource-oriented concepts, which are needed for modeling the production and use of some more or less abstract entity or the flow of such entities from one component individual to another (Heinrich & Jüngst, 1991). Examples of resources include space in rack and power.

**Resource type** intensionally defines the properties of the resource. Resource type is a subclass of configuration type. We call an individual of a resource type a resource production individual, since in effect represents the fact that a resource is available or needed in a configuration. A resource type has a *computation definition* that specifies whether the resource production or use must be:

- *satisfied*, in which case the quantity of resource produced must be equal to or greater than the quantity of the resource used; or
- *balanced*, in which case the quantity of resource produced must be equal to the quantity of the resource used.

The computation definition also specifies a *unit of measure* (Gruber, 1995). In addition, it specifies how the production and use of the resource type by several component individuals are combined. This is done through a *total production function* and a *total use function*. The prototypical case is that the quantities of the resource produced or used are added together to get the total quantity. The total production and use functions also define how the production or use of subtypes of the resource type are combined to the production or use of the resource type. A subtype of a resource type may refine the attribute definitions and the computation definition with respect to satisfaction and balancing and the total functions.

A component type specifies *production definitions* and *use definitions* the resource types it produces and uses. Both production and use definitions specify a *set of possible resource types* produced or used, a *property definition*, a *magnitude range* (Gruber, 1995), and a context definition. The produced or used resource must be of one of the possible resource types. The property definition is a special case of constraints (see section 4.8) that specifies a restriction on the attribute values of the resource produced or used. The magnitude range specifies how much of the resource type can be produced or used in the units of measure associated with the resource type. The context definition specifies a restriction on “where” in the configuration the resource is or should be available for use. A subtype of a component type may refine the possible types of the resource, the property definition, the magnitude range, or the context definition.

The motivation for introducing the context in which a resource is available is that unrestricted flow of resources from producers to users may model a product inadequately. A resource is only available to component individuals that are in the same context as the producing component individual. The total production of some resource type in the configuration may be enough to satisfy the use but the production may be divided between contexts in such a manner that not all users are satisfied.

**Example.** In our example there are two resource types: module slots and display power (Figure 3b). Both are required to be satisfied and their production and use functions are defined simply as sums of the resource production (Figure 4 under “Resources”). The unit of measure for them are Pieces and Watts, respectively. There are no property definitions for resources in this example, so we omit them.

A base-unit has three module slots which are available to All. We define All as a special component type set context consisting of all component types in the configuration model. A module uses either one or two module slots in the context all. Subtypes of module refine this amount of slots used to one or two (M-RS). These definitions, together with the constraint (2 (Figure 4 under “Taxonomy”)) stating that there must be at most one HD-Module individual in a configuration, mean that one cannot fit the modules M-NM and M-AHD together with the M-RS-module into the base-unit.

Note that we assume here implicitly that there can exist at most one Hmonitor individual in a configuration. Otherwise, a module could get the module slots from a different monitor individual than the one of which the module is a part. To prevent that from happening, one should define a transitive mereological context consisting of the monitor and its parts for the module-slot resource production.

A base-unit and an XGA-adapter produces 170 W and 200 W of display power in the primary display and secondary display contexts, respectively. The primary display context and secondary display contexts are defined as the immediate topological context defined by the base-unit individual and XGA-adapter, respectively, and the component individuals immediately connected to them. The 10-, 14-, and 21-inch displays use 60 W, 100 W and 150 W of display power in the context All, respectively. These definitions mean that it is not possible to have a configuration where there are two 14-inch displays connected to the base-unit.

Similarly as for module slots, the context of production for display power should have to be defined as the intersection of the mereological context specified above and the topological contexts specified here to correctly model a configuration with several monitor individuals.

A resource is produced by component individuals and used by component individuals in some quantities. This is con-
ceptualized as PRODUCES- and USES-relations between component individuals, resource production individuals, and MAGNITUDES. A resource production individual in a produces- or uses-relation to a component individual must respect the production or uses definitions of the component type of the component individual, that is, the resource production individual must be one of the possible resource types and the magnitude must be in the magnitude range. Every valid component individual that produces or uses a resource must be in the produces- and uses-relations to at least one resource production individual, or possibly several of them, and quantities specified by its component type. A valid resource production individual must not exist in a correct configuration without being in a produces- and uses-relations to some component individuals.

The effect of context specifications must be taken into account when deciding whether resource production and use are satisfied or balanced. If both the producing component type and using component type of the resource type define contexts, then the corresponding individuals must be in the intersection of the contexts for the using component individual to be able to use the resource produced by the producing component individual.

Context definitions together with the possibility of a resource using component individual getting its resource from several producing component individuals (or the other way round) makes it necessary to allocate the shares of the magnitude of the resource produced to the using component individuals. For brevity, we do not discuss allocation in detail in this paper.

4.7. Functions

In this section, we consider a configurable product from the point of view of the functions that the product instance provides to the customer, the user of the product, or the environment in which the product instance will be situated. We motivate the need for distinct concepts for representing functionality and introduce the concepts of a function type and a function individual. We define the part definition for function types that decompose functions to their subfunctions. Finally, we define how functions are mapped to the technical description of the product and how the functions may have dependencies on each other. The concepts introduced so far are referred to as technical concepts since they have risen from the technical point of view on the product.

A complex product is often configured in two stages (Tihonen et al., 1996). In sales configuration, the product may be specified in terms of functions. In engineering configuration, the resulting functional specification is used as an input that is refined to a technical description of the system. Engineering configuration operates within the domain of the technical concepts and on the knowledge on which combinations of technical concepts realize the functions in the functional specification.

A functional specification may, for example, specify that a telecommunication switch must provide access to at least 1000 subscribers. The function is mapped to a combination of component individuals in certain relations to each other that realizes the function. A function cannot be easily represented by any of the concepts introduced above nor can it be considered the property of any one configuration specific class. A function is not considered a resource, since resources in our ontology are intended to model the technical rules that the product must obey. In addition, a function might not be produced by a set of component individuals as a resource is but by several component individuals in given relations to each other.

The basic concept in the functional view is FUNCTION TYPE. It is a subclass of configuration type. A function type is an abstract characterization of the product that a customer or sales person would utilize to describe the products uses. We call a function individual a FUNCTION. A PART DEFINITION of a function type corresponds to a part definition of a component type with the exception that the possible part types must be function types.

The relation between technical concepts and functions and their properties is expressed as IMPLEMENTATION CONSTRAINTS. They are a special form of general constraints (see Section 4.8). Several different combinations of technical concepts may realize the same functions and one combination of technical concepts may realize several functions.

There can be constraints on how different functions and their attribute values can be combined. These SPECIFICATION CONSTRAINTS are a special case of the generic constraints (see Section 4.8) that only refer to functional concepts. They are used similarly as other constraints to restrict the combinations of functions that a product can implement.

EXAMPLE. The main function of the product is to monitor environment (Hmonitoring in Figure 3c). This is decomposed to subfunctions of displaying the results of the measurements and the optional functions of measuring gases and liquids (Figure 4 under “Functions”). The result displaying function is necessary and can be standard or high-resolution. Note that this functional decomposition does not include all the functional aspects of the product, for example, the modules M-NM and M-AHD cannot be specified by functions. In this example, it is not possible to specify an incorrect combination of functions. If that were not the case, specification constraints could be used to restrict the combinations of functions to feasible ones or one could rely on the technical definitions to rule out impossible configurations.

The constraints c5 to c8 specify the implementation constraints in our example. These specify what the technical part of the configuration must be like in order for the functions to be present in the configuration. Note that one module can implement several functions and one function can be implemented by several alternative modules. The func-
tions high-resolution and standard display are implemented by a combination of an XGA-display-card and a 21-inch display and a base-unit and a 10-inch or 14-inch display, respectively.

Note that the implementation constraints do not function correctly if there were more than one monitor individual in a configuration. In such a case, the implementation constraints would have to additionally specify that the component individuals that implement the functions must be transitively parts of the same monitor individual.

4.8. Constraints

In this section, we define at an abstract level a general mechanism, constraints, for specifying the interdependencies of the individuals and their relations. We then define a mechanism for dividing a set of constraints to subsets that specify the product from different points of view. Examples of constraints have appeared in the examples in previous sections.

A constraint is a formal rule, logical or mathematical, or a mixture of these, which specifies a condition that must hold in a correct configuration. Constraints are a modelling mechanism that can be used when the other concepts in our ontology are not suitable for capturing knowledge on some aspect of a product. A constraint may specify arbitrarily complex interactions between types, individuals of types, and their properties using the terminology of the concepts. We assume the existence of a constraint language with enough expressive power to express the desired concepts. The only restriction we set on a constraint language is that it must be possible to evaluate whether a constraint is satisfied, violated or its truth value is unknown with respect to a given configuration.

We view a constraint as a constraint instance which has a constraint expression. We refer to constraint instead of constraint instance for brevity. Special cases of constraints, has-part inheritance definitions, property definitions of resource types, connection constraints, specification constraints, and implementation constraints, have already been mentioned. A constraint language should provide special support for specifying these types of constraints.

It is often necessary to define subsets of constraints, called constraint sets, that limit the allowed configurations from specific points of view on the product. A constraint is member of at least one constraint set. A configuration satisfies a constraint set if and only if all the constraints in the constraint set are satisfied. Correctness of a configuration can be checked from a given point of view by checking whether the corresponding constraint set is satisfied.

Technical and marketing constraints are examples of constraint sets. The technical constraints limit the configurations on the basis of which combinations are technically feasible. Marketing constraints limit the combinations on the basis of product policy, that is, which of the technically feasible combinations a company is willing to sell. Technical constraints may be further divided into, for example, technology and manufacturing constraints.

5. DISCUSSION AND COMPARISON

5.1. General

The ontology presented synthesizes the conceptualizations of the connection-, resource-, product structure-, and function-based approaches. From the connection-based approach, we have included the concepts component, port, and connection constraint. The resource-based approach is present in our ontology through the concept of resource that components produce and use. From the product structure-based approach, we have included the concept of compositional structure that is represented through part definitions of components and constraints on the possible parts of the components. The functional approach is included in the concept of a function that is implemented by components.

Our ontology is by no means the first to synthesize some of these conceptualizations. However, none of the previous approaches has combined all the concepts. We are not aware of any conceptualization or system that would allow directly modelling the example product. In the conceptualization by Mittal and Frayman (1989), the connection-oriented concepts are accompanied by a limited form of interactions called functions. Clarke (1989) and Pernler and Leitgeb (1996) have presented approaches that extended the function-based approach and combined it with the structure-based approach. A set of modelling primitives combining some aspects of the connection and product structure approaches was presented in the configuration design ontology (Gruber & Olsen, 1996). An independent approach combining product structures and connections was presented by Axling and Haridi (1994). An approach which combined to some extent connections, resources, and structure was presented by Stumptner et al. (1994). Kramer (1991) presented a combination of structure- and resource-based concepts.

Some concepts in our ontology overlap in the sense of formal expressiveness. The constraint concept alone would be enough for representing the knowledge captured by the other concepts. In our view, an ontology of product knowledge cannot be solely judged by its expressive power in the formal sense. A configurator may in principle represent the configuration knowledge using any expressive enough language such as constraints, first-order logic, or even Turing machines. However, higher level concepts for representing typical forms of configuration knowledge will result in a more compact and understandable representation of a configuration model. An ontology should also reflect the way product experts such as product developers or sales managers view a product. This facilitates maintaining the knowledge, which is a crucial issue for a configurator. Of course,
the expressive power of an ontology should be adequate to capture the necessary knowledge.

There can exist mappings between the different concepts in our ontology or other conceptualizations that would allow mapping a configuration model represented using one subset of concepts to another representation using another subset of concepts while preserving the set of correct configurations. Although the possible mappings between different conceptualizations or different parts of our conceptualization are interesting, they are less important than the clarity of the configuration models built on the conceptualization.

Many configuration concepts can be modeled using other concepts appropriately (Stumptner et al., 1994; Gruber & Olsen, 1996; Heinrich & Jüngst, 1996). We believe that the configuration domain typical pieces of knowledge should be modeled with a specialized construct conceptualized just for that purpose. This makes the modelling process easier and explicates the reason for a particular interaction between the components in a product. If a concept is modeled using another concept, the origin and meaning of it may be lost. The models may also become more complex.

5.2. Integration

The main concepts in our ontology, component, resource, port, and function are integrated loosely and as such they preserve most of their meaning in the previous approaches. Unlike in many previous approaches, the main concepts are treated uniformly with respect to several criteria. The main concepts are all defined both as types and individuals to explicitly separate configuration model and configuration solution knowledge. The main concept types can all be organized in classification taxonomies and have attribute definitions. They can be specified as abstract or concrete to distinguish between accurate and inaccurate information.

Organizing the different concepts in a taxonomy to explicitly represent their common properties and to reduce the maintenance effort is a general idea in configuration design (e.g., Cunis et al., 1989; Searls & Norton, 1990; Heinrich & Jüngst, 1991). The explicit distinction between abstract and concrete component types has been made by others (Kramer, 1991; Axling & Haridi, 1994; Gruber & Olsen, 1996), but only Weida (1996) considered it a more generic mechanism which applies to other concepts as well. We have followed this approach. Together with refinement of properties, this provides a powerful mechanism for representing parametric components and hierarchical partial choice (Kramer, 1991).

One aspect of integrating concepts arising from the different approaches is to consider whether the component types, resource type, port types, and function types are disjoint concepts, that is, whether there are component resources, component ports, and so on. We omit these types of entities from our ontology because we want to modularize the ontology with respect to the intended use of the concepts by using only one concept for modelling one type of phenomenon. The situations where a component can also be considered a resource can be modeled as a component type producing a corresponding resource type. Other combination concepts can be modeled similarly.

5.3. Structure

In general, the earlier work on structure in configuration has only considered direct has-part-relation and has not explicitly considered all the semantic restrictions that representing a has-part-relation properly requires (Artale et al., 1996). Our view of a compound component individual is different from that of the notion of “whole” in classical mereology (Artale et al., 1996) in that a compound component individual is an explicitly modeled entity with its own properties. We do not want to impose the requirement of classical mereology that all parts of a compound component type must be modeled to account for the properties of the compound component type arising from the interactions between its parts. This divergence from classical mereology implies that shared parts need not fulfill the immediate inferior condition (Artale et al., 1996) required for representing intermediate parts properly. The condition in essence states that a component individual must not be part of another component individual both directly and indirectly. Exclusive parts fulfill this condition by definition.

The structure of the product can be specified in a more flexible manner in our ontology than in the previous approaches. The previous work has usually included a mechanism to specify alternative types for parts, possibly through subtyping, and the number of parts (Cunis et al., 1989; Searls & Norton, 1990; Peltonen et al., 1994; Axling & Haridi, 1994). Possibility to specify a part name to distinguish between parts of the same component type has been rarer (Searls & Norton, 1990; Peltonen et al., 1994). The dependencies and sharing between the whole and its parts have been either implicitly fixed one way or neglected. The has-part inheritance mechanism between the properties of wholes and the properties of parts and transitivity of has-part-relation as explicit modelling concepts have to our knowledge been neglected with a few exceptions (e.g., Sattler, 1996).

5.4. Resources

Our conceptualization of resource interactions is an extension of the model presented by Heinrich and Jüngst (1991) in the following respects. The produced resource types can be abstracted and resource production and use allow variation in the types and amounts produced. The shared and exclusive use of resource is generalized via the use of contexts. Context also provides a natural integration of the resource-oriented modelling to structure-oriented and connection-oriented modelling. The combination of the different
approaches through contexts seems a very powerful extension of the resource oriented modelling.

As resources have attributes, it is possible to specify characteristics, such as quality of the resource type that is produced or used. In addition to resource balancing by adding the quantities together, we also allow resource satisfaction and arbitrary functions on how the different amounts are added together.

5.5. Connections

The connection-oriented concepts are basically the same as presented by Mittal and Frayman (1989). As a generalization, we allow parameterizing component types with respect to the port types and number of port individuals. Integrating the port concept to the structure-oriented concepts, alluded to by Mittal and Frayman (1989), is explicitly dealt with. The interactions between the structural concepts and connection concepts could be made more limited by requiring that all component individuals that are parts of one component individual should be connected. We do not; however, require this since it would force the modeler to model all the connections, some of which may be of no interest from the configuration point of view. Similarly, the connections could be restricted to the component individuals that are direct or transitive parts of the same component individual as is done by Gruber and Olsen (1996). We do not want to enforce this modelling restriction either.

5.6. Functions

It has been argued that there is no clear distinction between components and functions (e.g., Gruber & Olsen, 1996; Wielinga & Schreiber, 1997). We argue that there is a need to keep these two concepts separate, as the sales persons and customers can be more interested in functions than the technical structure of the product. Modeling the product through technical concepts does not answer this need of describing and understanding a product through its functions. Function modelling cannot take into account everything that a customer might require but some set of functions is usually identifiable. Most of the research on configuration design has not explicitly modeled the functional domain in the sense that we present it. The conceptualization of the functional knowledge in our ontology is similar to those presented by Clarke (1989) and Pernler and Leitgeb (1996).

5.7. Constraints

Constraints have been used in almost all of the work on product configuration to represent the dependencies between components. Our conceptualization of a constraint as an object and an expression is similar to that presented by Gruber and Olsen (1996). However, we do not commit to a particular language or particular forms of constraints for representing them. The generality of the language is necessary to capture the complex interdependencies between the entities that occur in products. Of course, as much of the product knowledge as possible should be modeled using other concepts than constraints, but there are often complex interactions in products that cannot be captured by the other concepts.

Introducing constraint sets for representing different points of view on valid configuration is to our knowledge an extension to previous work. It has its roots in our observations on how configurable products are managed by companies. The need for supporting different points of view stems from the different types of sales, engineering, etc. processes within a company each having different ideas of what constitutes a valid configuration.

6. CONCLUSIONS AND FUTURE WORK

The ontology presented in this paper covers the connection-, resource-, structure-, and function-oriented approaches. It is the most generic ontology in this sense that the authors know of and extends the previous conceptualizations in several ways. We have tried to make minimal ontological commitments when specifying the concepts originating from different approaches and their interactions. Instead of committing to a particular form of interaction, we have provided flexibility by allowing explicit specification of the interactions.

Our ontology does not have a minimal number of concepts in the formal sense for representing configuration knowledge. We do not consider this a problem. The clarity of configuration models should not be compromised by minimizing the number of concepts in a modelling language. We believe to have struck a good balance between minimizing the number of concepts and making configuration models understandable.

The ontology presented should be extended and refined to adequately cover all configuration knowledge. A general ontology should include geometric, pricing, and optimality-related knowledge and the knowledge on how to configure a product. Such an ontology should be validated by empirically modelling different kinds of products on the basis of its concepts. We intend to proceed in this direction in the future. The relevance of the ontology depends mostly on how easy it is to model different kinds of products.

The ontology should also be formalized into a formal ontology to make the underlying assumptions and restrictions clearer. This would facilitate a rigorous analysis of the ontology and the different types of languages that can be based on this ontology. An interesting topic would be the study of the complexity of the inference mechanisms for such languages.

It would be interesting to study whether the concepts and their interactions could be further refined and restricted to more powerful modelling mechanisms. An interesting subject for further research would be to investigate hierarchi-
General ontology of configuration

cal refinement along has-part-relation facilitated by the has-part inheritance. The interaction between resources and ports through contexts could be generalized to a more bond graph-like modelling as is done by Snively and Papalambros (1993). This would necessitate modelling the flow of resources through port individuals and component individuals at a more detailed level. The use of contexts could be extended to allow modelling various context dependent constructs, such as constraints that hold in a given context only.

ACKNOWLEDGMENTS

We gratefully acknowledge the financial support of Technology Development Centre Finland and Helsinki Graduate School of Computer Science and Engineering (HeCSE). We thank Hannu Peltonen for his comments on how to improve the presentation of the paper and for the comments on the concepts presented in this paper. We thank Asko Martio for the comments on the concepts presented in this paper. We also thank the anonymous reviewers for their suggestions that certainly improved this paper.

REFERENCES


Timo Soininen is a researcher in the Product Data Management Group (PDMG) at the TAI Research Centre of Helsinki University of Technology (HUT) and a graduate student at HeCSE. His main interests in product configuration are high-level, logic-based representation of configuration knowledge and analysis of computational properties of related reasoning tasks. He has participated in a state-of-the-practice survey of 10 cases on product configuration in the Finnish industry and development of a framework for structured analysis of configuration tasks and processes in a manufacturing company. He currently participates in national research projects on Product Data Management (PDM) which is a joint project of HUT and six industrial partners and Design for Configurability (DFC) which is a joint project of HUT, Tampere University of Technology and four industrial partners.

Juha Tiihonen also works in PDMG as a researcher. His main interest is product configuration, particularly methods and tools that would enable product experts to build and maintain product configuration models without a programming background. He is currently finishing his Licentiate’s thesis on the state-of-the-practice survey in the Finnish industry. He is the principal developer of the framework for product configuration. He currently participates in the national research projects on PDM and DFC.
Tomi Männistö is a researcher in PDMG and graduate student at HeCSE. His main interests lie in the data modelling of configurable products. He is especially interested in the modelling and management of the evolution of both product descriptions and product instances. He currently participates in the national research projects on PDM and DFC.

Reijo Sulonen is a professor of Information Processing Science at HUT. His research interests include data base systems, product data management, process modelling, software engineering, and electronic media.