



UvA-DARE (Digital Academic Repository)

Modelling domain knowledge using explicit conceptualization

Abu Hanna, A.; Jansweijer, W.N.H.

DOI

[10.1109/64.331490](https://doi.org/10.1109/64.331490)

Publication date

1994

Published in

IEEE Expert

[Link to publication](#)

Citation for published version (APA):

Abu Hanna, A., & Jansweijer, W. N. H. (1994). Modelling domain knowledge using explicit conceptualization. *IEEE Expert*, 9(2), 53-64. <https://doi.org/10.1109/64.331490>

General rights

It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations

If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: <https://uba.uva.nl/en/contact>, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.

Modeling Domain Knowledge Using Explicit Conceptualization

Ameen Abu-Hanna and Wouter Jansweijer, University of Amsterdam

APPPLICATIONS ARE CHARACTERIZED by the tasks and domains involved. For example, we can talk about a planning application for an elevator system, or the task of diagnosing digital devices in general. Knowledge modeling can therefore be divided into two conceptual subactivities: modeling the task and modeling the domain knowledge. If properly designed, the separate parts can be reused: Considerable parts of a task model can use different domain models, and vice versa. However, this does not always happen. Commonly, domain knowledge is reentered for every new knowledge-based system, making system construction expensive.

Parts of the domain knowledge partake in many tasks and can be reused if properly specified and organized. Moreover, this organization can be designed to support flexible reasoning. In this article we discuss an explicit *conceptualization* of the domain knowledge, at the heart of its organization. A conceptualization is the objects presumed to exist and the relationships and functions among them.¹ We augment the conceptualization with reusable annotations to form theories about the domain knowledge. The annotations and the conceptualization guide the construction of applications and support flexible reasoning during problem solving.

Task-oriented approaches and the conceptualization. Most approaches to the modeling of problem-solving knowledge are task-oriented and usually provide a *task-method specification* in which tasks are realized by problem-solving methods. They focus on analyzing and modeling the tasks where the domain knowledge is implied by the way it will be used. For example, the *Task Structure View* prescribes a way to structure task knowledge, specify control knowledge, and access domain knowledge.² A task-method specification such as the Task Structure View is a recursive decomposition of tasks into problem-solving methods that can be decomposed into subtasks. A problem-solving method points to the domain knowledge needed to perform the task to which the method belongs. This view virtually structures, albeit implicitly, the domain knowl-

edge. From a domain viewpoint, two important issues related to the task-oriented approaches are the reusability of domain knowledge and the flexibility of reasoning.

Reusability. Can domain knowledge be modeled without a task in mind? On the other hand, can problem solvers be specified without an idea of the domain knowledge available? These questions show the two extremes of the *interaction hypothesis problem* (the interaction being between the task and the application domain).³

We take the view, consistent with the *relative interaction principle* of CommonKads,⁴ that although modeling domain knowledge should consider its use, a considerable body of the knowledge is useful for a broad family of tasks. This knowledge can be reused, possibly after its customization. For exam-

A FRAMEWORK BASED ON EXPLICIT CONCEPTUALIZATIONS COMPLEMENTS THE TASK-MODELING APPROACH. IT SUPPORTS FLEXIBLE REASONING DURING PROBLEM SOLVING, AND LETS DOMAIN KNOWLEDGE BE REUSED.

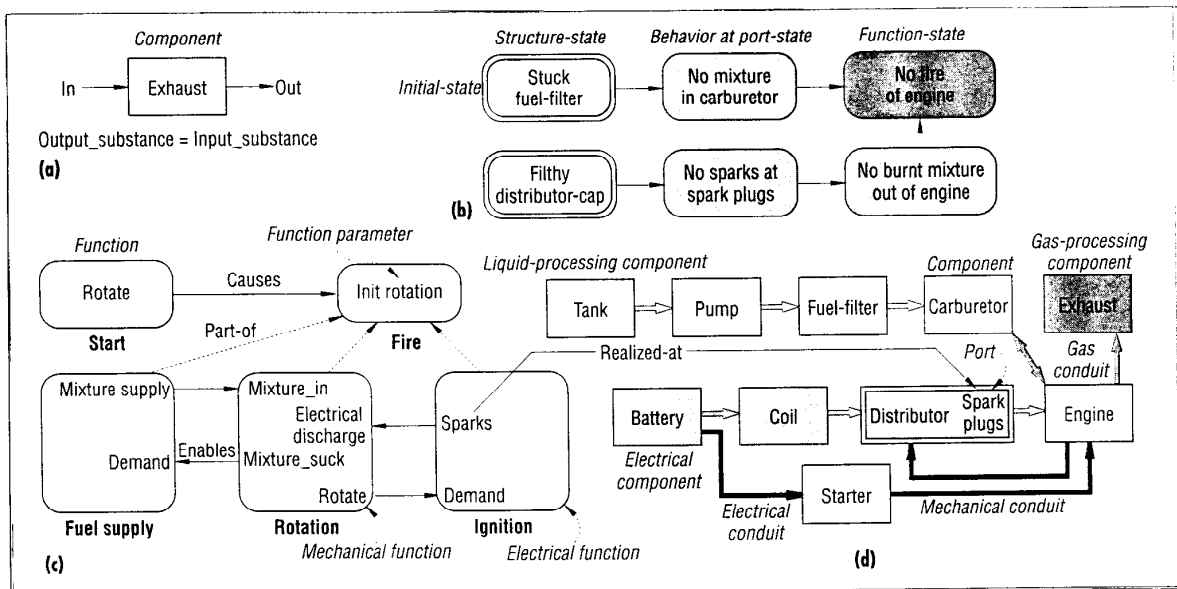


Figure 4. Four types of car models: (a) behavior, (b) state, (c) function, and (d) structure.

1. Computationally-efficient(Function-model).
2. Describes-physical-world(Structure-model).
3. Predictive(Behavior-model).
4. No-function-in-structure(Behavior-model).

Figure 5. Part of a model-type theory including mainly annotations of model types.

annotations, we might find relevant knowledge that can give rise to the execution of new problem-solving methods. This is especially useful when a problem-solving method is stuck because the knowledge it needs is not available; the navigation of the graph may discover new paths of reasoning.

The conceptualization, which underlies the ontology, provides general classes of domain knowledge that together with their annotations form a convenient medium that the task and its methods can use. The conceptualization and ontology are task-independent in the sense that they serve a variety of tasks without specifically committing to one. Theoretically, the problem-solving methods that use the conceptualization are not different from other problem-solving methods.

Framework for modeling domain knowledge

Using the idea of explicit conceptualization, we developed a framework for modeling the domain knowledge, that complements the task-modeling viewpoint. Like the

task-model approach, our framework makes relevant metaknowledge explicit, and it abstracts from implementational detail to provide a knowledge-level description.

This framework must meet five knowledge-engineering requirements:

- Robustness of reasoning.
- Reusability across tasks.
- Reusability across applications.
- Modifiability and maintainability.
- Support for knowledge acquisition.

We use five key techniques to meet these requirements. First, our framework supports different model types. Second, the conceptualization links the models with each other. Third, we augment conceptualization terms with reusable annotations that are meaningful to the task. Fourth, the framework explicitly maps knowledge to its role in reasoning. Fifth, we distinguish different generality levels of domain knowledge. Although we focus mainly on technical device models, our framework is more general—its components have no inherent device dependency.

We'll now describe our framework in more detail and show how these techniques meet the requirements.

Representing model types. Tasks reuse knowledge that can be preelicited and organized in a lump unit called a model. A model is a description of a world that might be the real world or an abstract world (such as another model). It consists of statements in some language that follows a defined syntax.

Models can view the world from different viewpoints, implying different types of models of the same world. The different viewpoints can reflect, for example, the *structural view* (such as components and connections, as in Figure 2), the *predictive view* (such as behavior and function of structural components), or the *physical view* (such as mechanical or electrical). In this article we focus on the structure and predictive models because they are widely used in many tasks in model-based reasoning, and are often available from the design process.

According to their viewpoints, we label a model by a type (structure, for example) that is a property of the model. This assumes that there are models that are not specific for one task only, but that correspond to a body of knowledge to which a variety of tasks frequently refer. So, by labeling this knowledge with a model type and including the notion of a model type in the explicit conceptualization, we provide a handle so the task can compactly refer to the domain knowledge. For example, a task that reasons about the effects between adjacent physical components may require a "topological model" as a condition for its performance. Alternatively, the problem-solving viewpoint could define model types that are specifically compiled for the task; however, this compilation often is not reusable for other tasks.

Figure 4 shows four types of car models. The description of an entity is in italics next to its symbol. For example, the bold solid arrow connecting the rectangles labeled

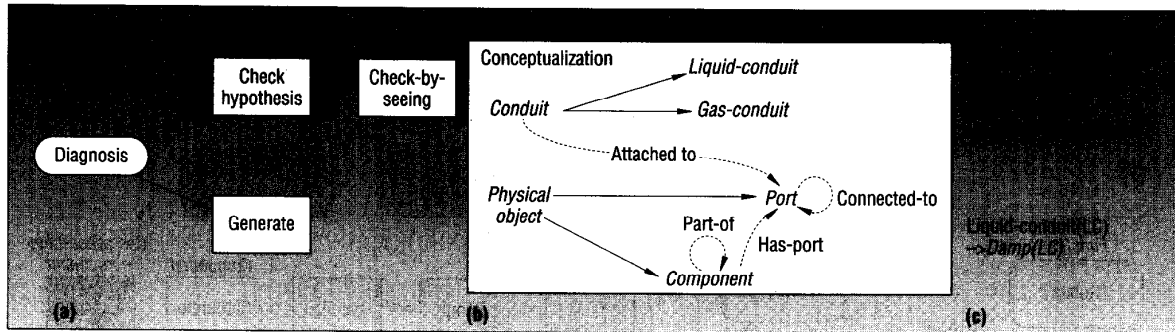


Figure 3. The task structure's use of the conceptualization and annotation: (a) task-method decomposition of diagnosis, (b) graphical specification of the conceptualization (ontology), (c) annotations of terms from the conceptualization.

The ontology in Figure 1b treats "component" and "engine" as concepts although they have different natures; while the concept "engine" appears in the theory about the car, the concept "component" often does not. The latter term has a metacharacter and is more general. If the term "component" appears in the knowledge base, then it usually merely points to the ontology. Such distinctions are important, and we aim to identify and use them to organize knowledge.

An ontology is different from a syntax specification. First, an ontology is more expressive because it can prescribe the semantics of the terms (for example, the intensional definition "primitive components are components without structural-part-of relations"). Second, an ontology facilitates portability because it describes the knowledge base's meaningful structure without committing to a specific internal representation. This means that while systems may vary in their syntax, they can still commit to the same ontology. Third, as we illustrate in this article, an ontology can be meaningful not only during knowledge acquisition or verification, but also during problem solving. This, however, demands additional knowledge around the conceptualization.

Uses of ontologies. An ontology can be viewed as an interface between the knowledge base and the external world. It has four major uses:

- **Sharability:** Knowledge engineers who wish to share their knowledge bases should commit to the same term definitions, as determined by the ontology.⁸
- **Knowledge acquisition:** Because the ontology eventually includes application-specific terminology, it can be used in knowledge-acquisition tools that directly interact with domain experts; it effectively avoids errors in the acquired knowledge because the constraints on the form and contents of the knowledge can be applied.⁹

- **Knowledge organization:** The ontology can organize knowledge bases so they are better accessed, modified, and reused.
- **Reasoning:** The ontology can be used together with the knowledge base as a model that a task can consider during reasoning.

We are most concerned with the last two uses. Generally, the ontology is meaningful only for the knowledge engineers who adapt the tasks and problem-solving methods accordingly, because they share the background theory that lets them interpret the terms' meaning. However, if we "ground" the meaning of some terms into the problem-solving methods, then the conceptualization can be meaningful to the reasoning itself.

Explicit conceptualization and the task design. Consider a simple diagnostic task that operates on a structure model by asking about the components and conduits that are "upstream" from a complaint. For example, if the complaint concerns the output of component *C* in Figure 2a, then the task will check component *C* first, conduit *b* next, then component *B*, and so on, until it arrives at a fault. We can view this method as a subtask that is decomposed into two methods: suggest-hypothesis (component or conduit) and check-hypothesis. We can further specify the check-hypothesis method when knowledge about conduits is available. For example, in Figure 2b, if we know that conduit *a* is transparent and transports gasoline, then we can (partially) check this conduit by seeing the gasoline flow or by feeling that the conduit is damp. One way to add this knowledge is to treat check-hypothesis as a subtask and decompose it into more specific problem-solving methods that apply to the features of conduits (or components).

This solution is not realistic, and it hinders reuse. In a car domain, for example, there are about 60 different material types, from nickel to nylon, each with its own features. This

would cause enormous proliferation of checking problem-solving methods. Much of the knowledge we are trying to capture is not specific to a theory of diagnosis but can be modeled as part of the domain and reused for different tasks.

We need a more general theory about the domain that exempts the problem-solving methods from overcommitments to the domain specifics. For example, the theory should know that gasoline, water, and oil are all liquids, and that transparent conduits that transport liquid can be seen flowing. Only at this point is the fact that something can be "seen flowing" (which is reusable knowledge for many tasks) related to its role in diagnosis. Such a theory can be based on an ontology that includes such terms as "conduits" and "liquids" as part of the conceptualization, and that can be enriched with such annotations as "seen flowing."

Figure 3 shows a possible configuration of the simple diagnostic task that operates on an ontology. The ontology in the figure is a graphical representation of an abstract specification of the conceptualization. The solid arcs in the ontology denote generalize-specialize relations, and the dashed arcs denote other relations between concepts. The check-by-seeing method in the task structure is added as a submethod of the check-hypothesis method. The knowledge engineer should map terms such as "seen flowing" into its diagnostic context.

The specification of the conceptualization and its annotation can support flexible reasoning because they can drive the tasks and methods without prespecifying, and sometimes overconstraining, their control. For example, when reasoning about the concept "conduit" in Figure 3, we know that it is attached to a port, which in turn can be connected to other ports; that it belongs to a component that can be part of other components; and so on. While navigating the graph of the ontology and considering the attached

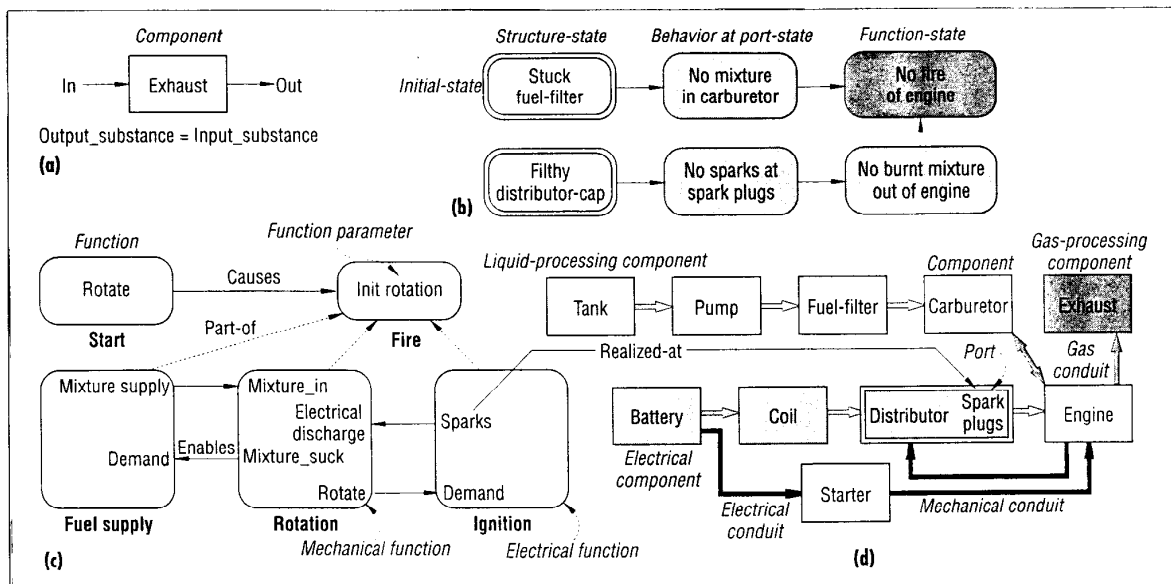


Figure 4. Four types of car models: (a) behavior, (b) state, (c) function, and (d) structure.

1. Computationally-efficient(Function-model).
2. Describes-physical-world(Structure-model).
3. Predictive(Behavior-model).
4. No-function-in-structure(Behavior-model).

Figure 5. Part of a model-type theory including mainly annotations of model types.

annotations, we might find relevant knowledge that can give rise to the execution of new problem-solving methods. This is especially useful when a problem-solving method is stuck because the knowledge it needs is not available; the navigation of the graph may discover new paths of reasoning.

The conceptualization, which underlies the ontology, provides general classes of domain knowledge that together with their annotations form a convenient medium that the task and its methods can use. The conceptualization and ontology are task-independent in the sense that they serve a variety of tasks without specifically committing to one. Theoretically, the problem-solving methods that use the conceptualization are not different from other problem-solving methods.

Framework for modeling domain knowledge

Using the idea of explicit conceptualization, we developed a framework for modeling the domain knowledge, that complements the task-modeling viewpoint. Like the

task-model approach, our framework makes relevant metaknowledge explicit, and it abstracts from implementational detail to provide a knowledge-level description.

This framework must meet five knowledge-engineering requirements:

- Robustness of reasoning.
- Reusability across tasks.
- Reusability across applications.
- Modifiability and maintainability.
- Support for knowledge acquisition.

We use five key techniques to meet these requirements. First, our framework supports different model types. Second, the conceptualization links the models with each other. Third, we augment conceptualization terms with reusable annotations that are meaningful to the task. Fourth, the framework explicitly maps knowledge to its role in reasoning. Fifth, we distinguish different generality levels of domain knowledge. Although we focus mainly on technical device models, our framework is more general—its components have no inherent device dependency.

We'll now describe our framework in more detail and show how these techniques meet the requirements.

Representing model types. Tasks reuse knowledge that can be preelicted and organized in a lump unit called a model. A model is a description of a world that might be the real world or an abstract world (such as another model). It consists of statements in some language that follows a defined syntax.

Models can view the world from different viewpoints, implying different types of models of the same world. The different viewpoints can reflect, for example, the *structural view* (such as components and connections, as in Figure 2), the *predictive view* (such as behavior and function of structural components), or the *physical view* (such as mechanical or electrical). In this article we focus on the structure and predictive models because they are widely used in many tasks in model-based reasoning, and are often available from the design process.

According to their viewpoints, we label a model by a type (structure, for example) that is a property of the model. This assumes that there are models that are not specific for one task only, but that correspond to a body of knowledge to which a variety of tasks frequently refer. So, by labeling this knowledge with a model type and including the notion of a model type in the explicit conceptualization, we provide a handle so the task can compactly refer to the domain knowledge. For example, a task that reasons about the effects between adjacent physical components may require a "topological model" as a condition for its performance. Alternatively, the problem-solving viewpoint could define model types that are specifically compiled for the task; however, this compilation often is not reusable for other tasks.

Figure 4 shows four types of car models. The description of an entity is in italics next to its symbol. For example, the bold solid arrow connecting the rectangles labeled

“starter” and “engine” in Figure 4d is annotated with “mechanical conduit.” These descriptions belong to the conceptualization and will be organized and used in reasoning. Relations (such as “causes” in Figure 4b) may have the same name in the model and the conceptualization (just like the relation “structural-part-of” in Figure 1).

In Figure 4a, the exhaust component’s behavior model says that the expected substance at the exhaust’s output port is the same as the substance that goes into the exhaust through the input port. (Substances are part of a material model type that we have omitted from this description.) Figure 4b shows the state model, which depicts the effects of initial states on the car’s condition. In this case it reflects (fault) states of structural components. The other states concern behavior and function. Figure 4c shows part of a car’s function model, which exhibits the relations between the parameters of the functions “start” and “fire,” and those of “fuel-supply,” “rotation,” and “ignition,” which constitute the decomposition of the “fire” function. Figure 4d shows part of a structure model whose components are distinguished according to the substance or material that they process or transport.

Relations between model types also exist. Figure 4 shows only one such relation: “realized-at,” which connects the functional parameter “sparks” of the “ignition” function with the port “spark plugs” of the component “distributor.”

Support for reasoning. Multiple models benefit the robustness, performance, and explanation of the reasoning system. First, multiple model types, such as structure and predictive models, make it possible to escape to alternative models once reasoning gets stuck in one model. Second, partial models of different types can compensate for the incompleteness of the individual models. Third, provided that the “right” model can be selected, multiple models provide efficiency because of either their contents (a function model is usually more efficient than a behavior model) or form (a certain form might compile better to the specific task¹⁰). Finally, because each model type reflects a specific understanding of reality, multiple models are attractive for explaining from different viewpoints.

It is also useful to represent statements about model types (for example, their definition) and their characteristics (for example, that the behavior-model submits to the “no-

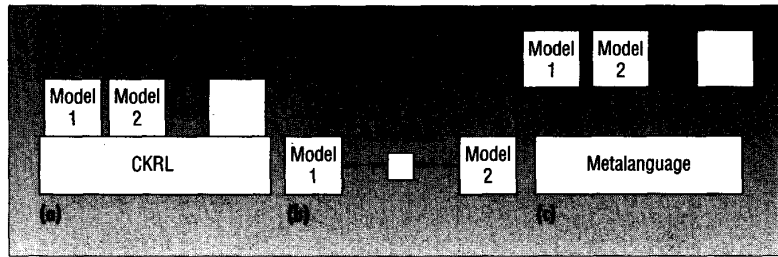


Figure 6. Basic integration architectures: (a) common knowledge representation language architecture, (b) translation architecture, (c) metalanguage integration architecture.

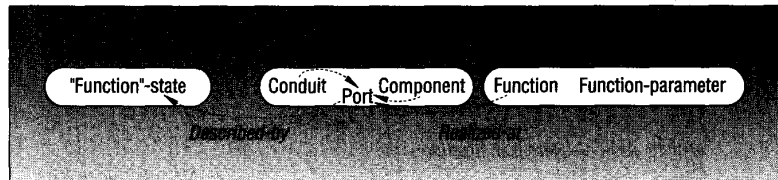


Figure 7. The integration architecture at the knowledge level, for the models in Figure 4.

function-in-structure” principle). In other words, these statements denote a model-type ontology and model-type annotations. These statements form theories about model types. Figure 5 shows part of a *model-type theory* that describes mainly model-type annotations. Model-type theories help the reasoner select the appropriate model types because they help assess the merits, application conditions, and efficiency of models.

The representation of model types and their characteristics only partially fulfills the requirement of robustness and efficiency of reasoning. A satisfying solution will not always be obtained after selecting one model, so we need to interrelate models to dynamically allow overstepping of their borders during problem solving. This *opportunistic* switching between models requires links between elements of different model types.

Linking multiple models through conceptualization. There are three basic architectures for integrating multiple models. Figure 6a shows the *common knowledge representation language architecture*, which defines one language that acts as the communication medium for representing all the models. Put in programming terminology, the models interact through global variables. Figure 6b shows the *translation architecture*, in which a syntactical translation mechanism between the models enables different languages for the various model types. This requires that when a model type is changed or added, all other model types must be changed or translated. Figure 6c shows a *metalanguage integration architecture* like that used in the ontology

knowledge base in Ontolingua.⁸ This allows the models to be independent from each other but communicate by a different language—a metalanguage of these models. A combination of these basic architectures can be used.

Choice of the metalanguage architecture. Our framework uses the metalanguage integration architecture that includes the “semantics” of models, because it allows the models to be loosely coupled and lets us add or modify models without much concern about the other models.

A model can be characterized by a set of ontological terms used in its conceptualization; this forms a good basis for an underlying metalanguage. For example, in Figure 4c the terms “electrical-function” and “function-parameter” belong to the function model. This metalanguage is at the knowledge level because it describes the models without specifying the actual formalism in which they are represented.

Figure 7 shows a part of the ontology of the structure, function, and state models from Figure 4. Each model type is represented explicitly and points to a set of underlying concepts and relations, shown as a rounded box. These concepts are *meta-domain concepts*; they type the underlying domain concepts. For example, the meta-domain concepts “component” and “function” type the domain terms “engine” and “fuel-supply” that appear in device models.

The *meta-domain relations* link meta-domain concepts, including those that belong to different model types. For example, the relation “realized-at” in Figure 7 associates “function” with “port,” which belong to the function and structure model types, respectively.

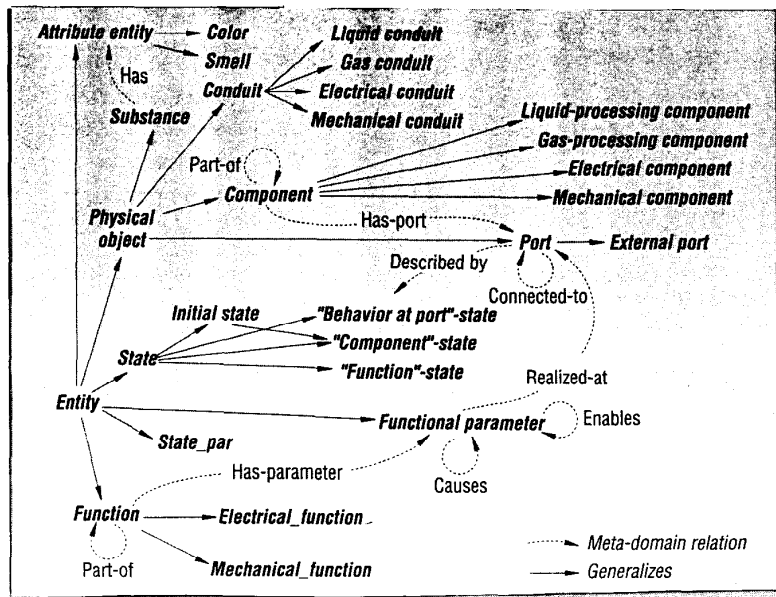


Figure 8. A part of the ontological semantic network (OSN) for the car domain. Bold, italic labels are meta-domain concepts; a solid arrow from Concept 1 to Concept 2 indicates that Concept 1 generalizes Concept 2 (that is, Concept 2 is more specific than Concept 1). A dotted line with a roman label indicates a meta-domain relation.

1. Fails-abruptly(Electrical-function).
2. Cheap-to-observe(External-port).
3. Observable(Smell).
4. Observable(Color).
5. Causes(A, B) \Rightarrow Occurs-before(A, B).
6. Can-be-seen(Physical-object).
7. Arguments-type-of(Has-port, Component, Port).

Figure 9. A part of a field theory.

The organization of the meta-domain vocabulary. In our framework, an ontological semantic network (OSN) organizes the meta-domain concepts in a generalize-specialize hierarchy and maintains meta-domain relations between these concepts; this forms the heart of the integration.

Figure 8 depicts a part of an OSN. The node labels (such as "component") and dotted arcs (such as "has-port") belong to the meta-domain vocabulary (concepts and relations). The solid arcs in the OSN denote the generalize-specialize relations. These relations imply a directed graph in the OSN that includes all nodes. The hierarchy is not necessarily a tree, because a node may specialize more than one concept.

Statements that define and annotate the meta-domain vocabulary, just like with model types, can be represented in a theory. We call this a *field theory* because the OSN and this theory are classified at the same generality level of knowledge, which we call

"field," and which we'll describe later. Figure 9 shows a field theory of seven statements about the terms in the OSN in Figure 8. For example, the first statement says that an electrical-function fails abruptly, and the fifth statement says that if A (a function or state, for example) causes B, then A occurs before B. The seventh statement specifies the argument types of the relation "has-port" and hence is part of the ontology.

Specialized concepts in the OSN inherit their parent's annotations. For example, "component" and "port" will inherit the statement that a "physical object" can be seen.

Ideally, meta-domain concepts and relations do not include task roles. For example, the term "hypothesis" does not appear in the OSN, but it is explicitly mapped onto a concept in the OSN, such as "component."

The OSN supports reusability and robustness. The meta-domain relations glue the meta-domain concepts and serve as the communication means between the model types: Switching between models is achieved by navigating through the OSN, beginning with a concept that belongs to some model type, and ending with a concept that belongs to another. The OSN's connectivity and the annotations of the concepts and relations in the field theory let the domain knowledge flexibly drive the tasks, and therefore support the robustness of the reasoning.

The OSN supports reusability across tasks.

First, we explicitly map task roles onto meta-domain vocabulary. In other words, meta-domain concepts and relations are the main "entricks" to the framework. This offers advantages over approaches that do not distinguish between task roles and application-domain terms, because we can reuse some domain knowledge in different roles. Second, because of the OSN's general character, it can capture a great amount of domain knowledge that is reusable across many tasks. The OSN provides a basis for multiple tasks, but it can be extended or customized with terms to facilitate a special family of tasks, without hindering its generality. For example, the addition of a meta-domain concept such as "observable" can be considered as oriented toward tasks that require human intervention for obtaining observables.

Representing knowledge at different generality levels. The *generality level* indicates the scope that the domain knowledge covers. Four basic levels—*case*, *domain*, *field*, and *model-type*—correspond to important landmarks on the generality spectrum. Each generality level has two defined *knowledge constructs*: The first organizes the vocabulary at that level (for example, the OSN), and the second is the theory formed by statements about the vocabulary (for example, the field theory).

In Figure 10, each shell denotes a generality level and shows parts of the two knowledge constructs at that level (rectangles denote theories). For example, the hierarchy on the left in the model-type shell denotes the organization of the model types, while the rectangle on the right denotes the model-type theory.

Case-level knowledge describes specific devices (or systems in general). The knowledge constructs are the set of case-level vocabulary terms (such as "c38" in Figure 10), and the theories about the systems or devices, which are expressed by statements about the vocabulary (for example, that ports an1 and p1 of components zd11 and c23 are connected). The vocabulary is formed by instantiations of the domain vocabulary. A case theory is the actual system model, such as the ones in Figure 4 if one treats the names of the entities as instantiations of entities from the domain level. Because the names in the figure are unique, we prefer names such as "fuel-filter" on "fuel-filter1." This level is reusable only across different devices that share the same model.

Domain-level knowledge describes classes of devices. The knowledge constructs are the organization of the domain-level vocabulary,

such as “diode” (this organization resembles the OSN except that the scope of the terms here is less general), and the domain theories, which contain statements about this vocabulary (for example, that diodes are sensitive to polarity or that a diode has a port called “anode”). Figure 11 shows part of a domain theory for the car domain. This level provides reusability across a family of devices sharing the same domain (cars, for example).

Field-level knowledge describes meta-domain knowledge. The knowledge constructs are the semantic network (the OSN) and the field theories (see Figure 9), which include statements about this vocabulary (for example, that components are replaceable). Because the field theories not only annotate the vocabulary terms but also include part of the ontology, they provide the meaningful structure of the domain and case theories underneath. For example, the fact that an1 may be connected to p1 is implied by the relation “connected-to,” which is denoted by the arrow that originates from and points to the concept “port” in the OSN. The field theory should define this relation (not shown for simplicity). The field level provides reusability across classes of domains (for example, the car and electrical domains share the same OSN). Unlike the domain level, instantiations of the field-level vocabulary do not appear in case models.

Model-type-level knowledge describes model types. Like the field level, it has a meta-domain nature. The first knowledge construct is the classification of model types.¹¹ For example, a behavior model is a kernel model. Roughly speaking, a kernel model describes a predefined set of structural elements and each element’s local behavior. Kernel models are compositional; for each device there exists a specific configuration of the elements whose behavior in the small explains the behavior of the device in the large. The second construct is the model-type theories (see Figure 5), with their statements (for example, a structure model has high connectivity among its constituents). This level may be reused with different classes of fields, domains, and case models.

Figure 10 shows only a selection of concepts, relations, and statements. For example, to keep the figure simple, we do not show the “is-a” relation between “anode” in the domain shell and “port” in the field shell, nor the mapping between the relation “has-port” in the field shell and the relation with the same name in the domain shell. The organization of a vocabulary is an abstract view of the theories.

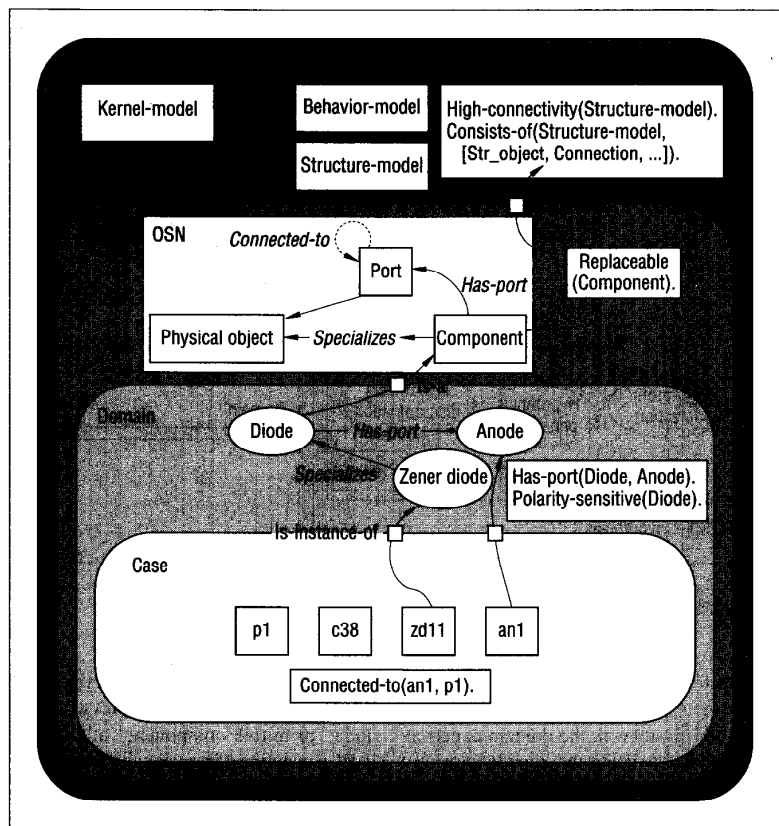


Figure 10. Partitioning the domain knowledge according to generality levels.

For example, the arrow labeled “has-port” connecting the “component” to the “port” nodes in the OSN corresponds to a statement in the field theory that specifies the “has-port” relation. The theory can also state whether each component must have at least one port.

Types of statements. The theories at the model-type, field, and domain levels may have three types of statements: definitions, conditions, and annotations. Stating that a “primitive component” is a “component” with no “part-of” relations, at the field level, is a definition of the term “primitive component.” This intensional definition lets us derive the primitive components in the domain and eventually the case models. Stating that each port must belong to a component is a (necessary) condition; if the condition does not hold for an object, then that object cannot be a port. Finally, stating that a “component” is replaceable, without further defining “replaceable,” is an annotation.

While definitions and conditions mainly allow sharing between knowledge bases that commit to them, annotations are mainly bridges between the knowledge base and the

1. Results-in-noise(Rotation-function).
2. Usually-accessible(Spark plugs).
3. Smells-like(Gasoline, Gasoline-odor).
4. Smells-like(Mixture, Gasoline-odor).
5. Smells-like(Burnt-mixture, Burnt-odor).

Figure 11. A part of a domain theory. All statements reflect annotations.

task. The problem-solving methods themselves ought to handle the annotations (such as “replaceable”), as we described earlier.

The above examples are all from the field level, but the situation in the model-type and domain theories is similar. For example, to say that the constituents of a structure model are structural objects and connections is part of a structure-model definition; to say that diodes are polarity-sensitive is a domain-level annotation.

Support for reusability, knowledge acquisition, and modifiability and maintainability. Representing knowledge at generality levels directly affects not only reusability across applications, but also knowledge acquisition, because the knowledge-acquisition sources

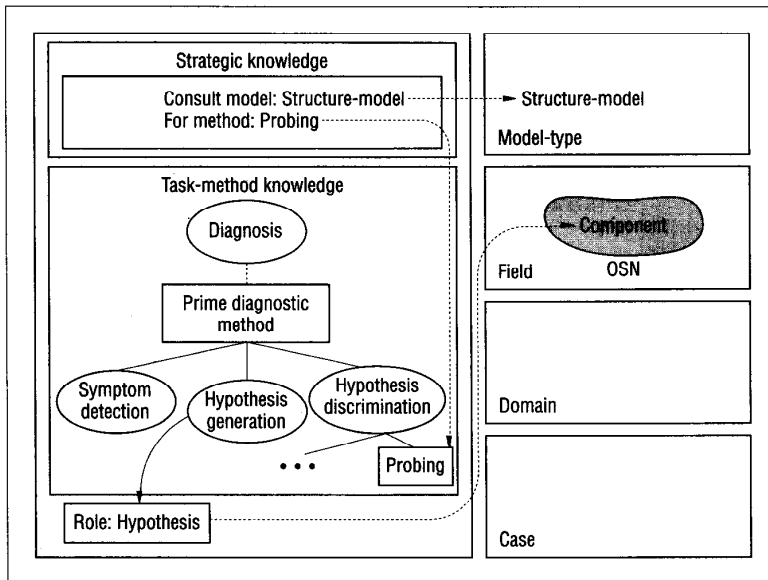


Figure 12. Reasoning models and domain models in perspective.

at the different generality levels can be identified: the user of the device or domain expert at the case level, the domain expert at the domain level, and the knowledge engineer at the field and model-type levels. Moreover, acquisition of knowledge at a generality level can be guided and constrained by the theories relevant to that knowledge.

The generality levels are inherently parsimonious: An operation performed on one level (such as consistency checking) will automatically hold for the levels underneath. This assists modifiability and maintainability.

The generality levels also support modularity, which is maximal cohesion and minimal interaction between knowledge chunks. This supports modifiability because it helps identify the knowledge chunks that need to be updated because of a change in the knowledge base. Cohesion appears in the framework in two forms: breadth and depth. Breadth corresponds to the generality level: The knowledge at each level shares the same scope. Depth reflects the membership to a model type, cutting through the shell structure in Figure 10. For example, structural knowledge crosses all generality levels.

The explicit mapping between the generality levels (shown as arrows between the shells in Figure 10), and the use of the OSN as the integration medium, explicitly represent the interaction between the knowledge chunks. This supports maintainability because, when a change occurs in one knowledge chunk, the channels of interaction between the chunks can be directed to control (and sometimes minimize) the amount of

change in other chunks. For example, a knowledge engineer who decides to add the node "primitive component" at the field level, can define it intensionally as a component with no internal structure. At the domain level, only the domain terms classified as "component" should be checked to see if they do not have a "structural-part-of" relation. More important, the explicit mappings allow the knowledge engineer to redefine the mappings. For example, the relation "has-part" at the field level can be mapped on a relation with a different name or even different scheme at the domain level. Because of the ability to redefine mappings, the framework allows the reuse of the same field-level knowledge with different idiosyncratic *representations* of domain and case models.

Applying our framework in Musarela to diagnostic reasoning

We have implemented our framework in Musarela,¹² a system that acquires multiple models according to the knowledge structures at the four generality levels. Musarela can be used for systematically documenting domain knowledge, for guiding the construction of problem solvers, and during reasoning. For knowledge acquisition, Musarela is meant to be used by a knowledge engineer, a domain expert, and a device user. We have used it to construct a model of a car that we used in a diagnostic application.

The framework in perspective. Musarela is meant to operate in a broader framework that models the reasoning as a task-method decomposition of the problem-solving behavior.² The task-method construct is governed by strategic knowledge that can be implemented by special problem-solving methods. Figure 12 illustrates this broad framework and shows how the reasoning (diagnosis in this case⁶) relates to our modeling framework. The diagnostic task is realized by the prime diagnostic method, which in turn consists of three subtasks: symptom detection, hypothesis generation, and hypothesis discrimination. The last can be realized by the "probing method" (and possibly other methods). Each task is associated with roles that the domain knowledge eventually plays. For example, the hypothesis generation subtask is associated with the role "hypothesis," which is mapped to the "component" concept at the field level. In Figure 12, the strategic knowledge is illustrated by the statement that a structure model is to be used when executing the probing method.

Reasoning. A strategic module (see Figure 12), which we'll call "strategy," controls and manipulates symptom detection, hypothesis generation, and hypothesis discrimination. The strategy achieves the task goals and recovers from possible failures. The strategy also includes the problem-solving methods that use the conceptualization when new avenues of reasoning are sought.

Many strategic decisions are based on the contents of the various theories, particularly the field theory. Interpreting the theories is straightforward if a human performs the strategic component of reasoning. However, automating the interpretation is not trivial. The theories include knowledge about certain concepts, but this does not mean that the reasoner can "understand" them. During system construction the knowledge engineer should specify how the problem-solving methods and the strategy must deal with the terms (such as "cheap-to-observe") introduced in the theories.

Whenever the strategy requires knowledge regarding a concept *C* (such as a node in the OSN), it follows this line of reasoning: If the strategy requires knowledge about *C* itself, the strategy looks into the relevant field statements that annotate *C* (for example, annotating "electrical-function" as "fails-abruptly"). However, if the strategy is evaluating alternative knowledge sources that are related to *C*, it proceeds differently. It forms a set of concepts that, according to the OSN, are related

Related work

Our framework is similar to other work on ontologies, task-oriented approaches, and model integration methodologies.

Work on ontologies has been quite popular (for example, Ontolingua¹ and CommonKADS²). There is a proposal for a Knowledge Interchange Format,³ which is used in Ontolingua, a tool for portable ontology specification. A number of projects use Ontolingua, such as Games, which focuses on ontologies in the medical domain.⁴ Ontolingua seems to be an attractive option for expressing Musarela's statements in the domain, field, and model-type theories. The fundamental difference between our framework and these approaches is that we want to augment the conceptualization with annotations and use it during problem solving, instead of mainly in an "off-line" manner that enables knowledge reuse and guidance in knowledge acquisition. Moreover, our framework introduces model types and generality levels, which define important criteria for cohesion of knowledge and contribute, respectively, to a content theory about ontologies, and reusability across applications.

The goals of such task-oriented approaches as the Task Structure View,⁵ KADS⁶, CommonKADS, and Commet⁷ are beyond domain knowledge. Our framework, on the other hand, focuses on structuring domain knowledge, making only general assumptions about the task. Therefore, the framework is meant to complement and be used in symbiosis with tasks that have an explicit conceptual model. Our framework would need to work with the Task Structure View, enrich the "domain knowledge layer" in KADS and CommonKADS, and constitute Commet's "model component." As with the work on ontologies, another major difference between our way of structuring domain knowledge and the task-oriented approaches is that we explicitly represent model types and generality levels, which supports knowledge reusability and robustness of reasoning.

Various approaches to model integration share some of our aims. Model graphs have

been suggested where abstraction and simplification relations interconnect models.⁸ Our framework types models and provides the model-type theories for representing such relationships. In our earlier work⁹ and in other research,^{10,11} as in much of the later work on Functional Representation,¹² various models are used in an efficient but basically prespecified order to solve problems. The Task Structure View, as specified in Generic Task Problem Spaces (GTPS)¹³ and implemented in Soar,¹⁴ can model problem solving with Functional Representation to allow flexible reasoning. Our framework extends this by letting one or more GTPS work with Musarela's knowledge constructs. A GTPS, or a problem-solving method in general, that uses the model-type theories, the OSN's infrastructure, and the field theory allows flexible guidance of problem solving, and opportunistic switching between models.

References

1. T.R. Gruber, "A Translation Approach to Portable Ontology Specifications," *Knowledge Acquisition*, Vol. 5, No. 2, June 1993, pp. 199-220.
2. B.J. Wielinga et al., "Towards a Unification of Knowledge-Modeling Approaches," in *Second-Generation Expert Systems*, J.-M. David, J.-P. Krivine, and R. Simmons, eds., Springer-Verlag, New York, 1993, pp. 299-335.
3. M.R. Genesereth and R.E. Fikes, "Knowledge Interchange Format Version 3.0 Reference Manual," Tech. Report Logic 92-1, Logic Group, Stanford Univ., Stanford, Calif., 1992.
4. G. van Heijst et al., "A Case Study in Ontology Library Construction," to be published in *Artificial Intelligence in Medicine*, June 1995.
5. B. Chandrasekaran, T.R. Johnson, and J.W. Smith, "Task-Structure Analysis for Knowledge Modeling," *Comm. ACM*, Vol. 35, No. 9, Sept. 1992, pp. 124-137.
6. B.J. Wielinga, A.T. Schreiber, and J.A. Breuker, "Kads: A Modeling Approach to Knowledge Engineering," *Knowledge Acquisition* (special issue on the KADS approach to knowledge engineering), Vol. 4, No. 1, Mar. 1992, pp. 5-53. Also in *Readings in Knowledge Acquisition and Learning*, B. Buchanan and D. Wilkins, eds., Morgan Kaufmann, San Francisco, 1992, pp. 92-116.
7. L. Steels, "The Componential Framework and its Role in Reusability," in *Second-Generation Expert Systems*, J.-M. David, J.-P. Krivine, and R. Simmons, eds., Springer-Verlag, New York, 1993, pp. 273-298.
8. P. Struss, "What's in SD? Towards a Theory of Modeling for Diagnosis," in *Readings in Model-Based Diagnosis*, W. Hamscher, L. Console, and J. de Kleer, eds., Morgan Kaufmann, San Francisco, 1992.
9. A. Abu-Hanna, R. Benjamins, and W. Jansweijer, "Device Understanding and Modeling for Diagnosis," *IEEE Expert*, Vol. 6, No. 2, Apr. 1991, pp. 26-32.
10. L. Chittaro et al., "Developing Diagnostic Applications Using Multiple Models: The Role of Interpretive Knowledge," *Industrial Applications of Knowledge-Based Diagnosis*, G. Guida and A. Stefanini, eds., Elsevier Science Publishers, Amsterdam, 1992, pp. 219-263.
11. J. Sticklen and B. Chandrasekaran, "Integrating Classification-Based Compiled-Level Reasoning with Function-Based Deep-Level Reasoning," *Applied Artificial Intelligence*, Vol. 3, Nos. 2-3, 1989, pp. 275-304.
12. B. Chandrasekaran, "Functional Representation and Causal Processes," in *Advances in Computers*, Vol. 38., Academic Press, Orlando, Fla., 1994, pp. 73-143.
13. T.R. Johnson, *Generic Tasks in the Problem-Space Paradigm: Building Flexible Knowledge Systems While Using Task-Level Constraints*, PhD thesis, Ohio State Univ., Columbus, Ohio, 1991.
14. J.E. Laird, A. Newell, and P.S. Rosenbloom, "SOAR: An Architecture for General Intelligence," *Artificial Intelligence*, Vol. 33, No. 1, Sept. 1987, pp. 1-64.

to *C* (for example, "function-parameter" is related to "function" by "has-parameter"). For each element *E* in this set, the strategy evaluates the corresponding model type (for example, "function-model") by interpreting the relevant statements in the model-type theory (for instance, "computationally-efficient" annotates "function-model"). The strategy then chooses the most promising *E* and reasons about its corresponding model type using the appropriate problem-solving method.

There is much room for specifying how broadly the strategy should search for rele-

vant knowledge, and at what cost this search should proceed. Moreover, the user needs to specify the heuristics for evaluating the potential lines of reasoning. The current problem solver that uses Musarela is not fully automated; it is the user that actually chooses how to proceed from possible alternatives provided by the system.

A diagnostic session. To show how our knowledge-modeling framework is used, we'll present a diagnostic session in the car domain, using the following knowledge constructs:

- The structure, behavior, function, state, and material case models (Figure 4 shows the first four). Recall that the entities in the case models ("fuel-filter," for example) are to be treated as instantiations of the vocabulary at the domain level.
- The samples of model-type, field, and domain theories in Figures 5, 9, and 11. Observe that statements 3 and 4 in the field theory, and statements 3, 4, and 5 in the domain theory, relate to the material model type.
- The OSN in Figure 8.

General modeling principles

Our framework follows these general modeling principles, some of which are shared by task-oriented approaches:

- *Making relevant metaknowledge explicit.* The explicit conceptualization and the model types comprise knowledge about the underlying knowledge at the domain and case levels. Task-oriented approaches apply this principle by explicitly representing knowledge about the problem-solving behavior in terms of tasks and methods.
- *Linking multiple models using a common knowledge-level theory.* The integration of models using the meta-domain terms at the field level is at the knowledge level because it abstracts from implementational detail. The modeling of task knowledge in the task-oriented approaches also provides a knowledge-level description of problem solving.
- *Decoupling the knowledge from its role in reasoning.* We realize this by explicitly mapping the task or problem-solving roles onto their counterparts in the framework.

Most other approaches (CommonKADS,¹ for example) also use this principle, which such approaches as Protégé-II² extend to suggest ontologies of mapping relations between tasks and domains.

- *Relative interaction.* This means that a good modeling methodology can span the continuum of coupling between task and domain knowledge from weak to strong. We achieve this by modeling the domain knowledge without strong assumptions about the task, but letting the organization of the knowledge be extensible to accommodate task-oriented knowledge. This principle is borrowed from CommonKADS and provides a compromise between the two extremes of the interaction hypothesis problem.
- *Explicitly distinguishing knowledge of different generalities.* We follow this principle by representing knowledge at different levels of scope of applicability (generality).
- *Maximum cohesion and explicit interaction.* Cohesion appears in the framework

in two forms, breadth and depth. Breadth corresponds to the generality level. Depth reflects the membership to a model type. The explicit mapping between the generality levels, and the use of the OSN as the integration medium, explicitly represent the interaction between the knowledge chunks. This principle resembles the well-known modularity principle in software engineering.

References

1. B.J. Wielinga et al., "Towards a Unification of Knowledge-Modeling Approaches," in *Second-Generation Expert Systems*, J.-M. David, J.-P. Krivine, and R. Simmons, eds., Springer-Verlag, New York, 1993, pp. 299-335.
2. J.H. Gennari et al., "Mapping Domains to Methods in Support of Reuse," *Proc. Knowledge Acquisition Workshop (KAW '94)*, SRDG Publications, Dept. of Computer Science, Univ. of Calgary, Alberta, Canada, 1994, pp. 24-1-24-20.

The symptom in this example is "while starting, engine makes normal noise but does not fire." Suppose there are various problem-solving methods for hypothesis generation that can work with different model types. The first step is to examine the symptom to get a handle to the framework's models, and to register observables that might help during reasoning. The symptom is characterized, possibly by the user, as "abrupt," which means that the failure is severe, not just a slight degradation of the car's performance. The system also records the observation of "normal noise."

We need an entry into a model type that is related to the symptom. Using symptom detection, the diagnostic system determines that the term "fire" belongs at the domain level and is a function at the field level. A function belongs to the function model (this follows from the specification of the function model in the model-type theory). The function model is our starting point in the reasoning. We have just used the mapping mechanisms to arrive at the model-type level.

The second step is to generate hypotheses by using the function model at the level where the "fire" function appears. The hypothesis-generation step results in suspecting the functions "fire" and "start" (Figure 4c). Now we turn to hypothesis discrimination. We find that the "rotate" parameter of the "start" function causes the initiation of the rotation in the "fire" function. From the

field theory (fifth statement) we know that the "rotate" parameter gets its value before the initiation of the rotation of the engine. This information, coupled with the fact that normal noise was heard and that rotation results in noise (according to the domain theory), indicates that the "rotate" parameter is probably not the problem. This kind of inference is called *corroboration*, which means the abduction of a premise based on observing the implicant. This kind of reasoning is not always sound; it strongly depends on the domain and the device at hand.

The "fire" function remains in focus. The OSN indicates that the concept "function" is, in principle, decomposable, and the strategy stays with the function model type because the model-type theory indicates that function models are efficient.

Another cycle of hypothesis generation occurs, this time at a finer level of the function model. The diagnostic system decomposes the "fire" function (Figure 4c) and generates the subfunctions "fuel-supply," "rotation," and "ignition" as suspects. In hypothesis discrimination, the field theory tells us that electrical functions tend to fail abruptly. This information, together with the fact that the symptom is classified as abrupt, focuses attention on the "ignition" function.

The diagnostic strategy selects the structure model because there is no further decomposition of the ignition function, because a function has a parameter that can be real-

ized at a structural port (according to the OSN), and because the model-type theory says that structure models describe the physical world (which makes them applicable for probing). If we apply this line of reasoning to the "ignition" function, we arrive at the "sparks" functional parameter and end up with the "spark plugs" port of the "distributor" component (see Figure 4d).

The diagnostic system may ask to probe the car at any point starting from "spark plugs" or upstream from it. It selects a measuring point at the spark plugs based on test-cost considerations (not shown here), which would include the fact that spark plugs are usually accessible, according to the domain theory. If we assume that the user cannot observe whether there are sparks (for example, because the spark plugs in this car—contrary to what the domain theory suggests—are difficult to access), the diagnostic system consults the OSN to see what additional knowledge it has about ports. The "port" concept in the OSN in Figure 8 is related to other meta-domain concepts.

In this case the generalize-specialize relations concerning "port" do not help because they do not provide knowledge about the spark plugs (they are not annotated in the field theory). The terms "component," "functional-parameter," and "port" do not help either, because "component" has no annotations in the field theory, "functional parameter" was where we started, and there are no

other ports to observe that are connected to the plugs (we could have considered other ports upstream, but they were costly to observe). So, "behavior at port-state," which is connected to "port" via the relation "described-by," is the most attractive alternative. We have landed in the state model to which "behavior at port-state" belongs. The exact entry in the state model is the state "no sparks at spark plugs" (see Figure 4b).


The diagnostic system applies hypothesis discrimination again. According to the state model, the initial state "filthy distributor-cap" may cause the assumption of no sparks at the plugs. It also means that there will be no burnt mixture out of the engine and no firing of the engine. The latter covers (and hence is consistent with) the observations. The diagnostic strategy might still need more evidence for the hypothesis. We must verify the expectation that there is no burnt mixture out of the engine. We need more expectations to match with observations. The system looks for a model that can predict results of abnormal behavior. The strategy uses the model-type theory to determine that the behavior model is suitable because it is in accordance with the no-function-in-structure principle.

Another round of hypothesis discrimination takes place. The exhaust's behavior model (see Figure 4a) predicts that "no burnt mixture" that enters the exhaust out of the engine causes "no burnt mixture" out of the exhaust. The exhaust output is an external port that the field theory suggests is usually cheap to probe. The OSN states that materials have "smell" and "color" attributes that can be observed, according to the field theory. The domain theory says that unburnt mixture smells like fuel. The system asks the user to perform the test. The test result matches the expectation. The system considers this as enough evidence to believe that there are no sparks at the spark plugs.

In this stage, the diagnostic strategy aims to verify the reason for the absence of sparks. Hypothesis discrimination operates on the structure model and suggests probing the coil's output. The user reports that the output is correct. This singles out the distributor component, which is presented as the diagnosis. The system might suggest the distributor cap is filthy as a possibility it knows about (see Figure 4b).

In summary, the diagnostic task employed multiple models in a cooperative manner to arrive at a solution. In symptom detection, the diagnosis system classified the symptom

as functional, which led to selecting the function model. The system used field and domain knowledge to filter and focus the hypothesis set in hypothesis generation. It used the OSN to integrate the model types and find alternative reasoning avenues. The alternatives corresponded to model types whose merits were evaluated using the model-type theory.

 OUR ARTICLE FOCUSED ON structure and predictive model types. However, there is also a need to investigate and introduce other model types and to study their characteristics, which will be captured in statements of the model-type theories.

It would also be useful to represent unifying modeling theories such as the Bond-Graph theory, which views the behavior in the world in terms of generalized variables and equations, and hence is attractive for modeling devices and processes.¹³ These general theories, which contribute to a content theory of conceptualizations, will mainly be reflected at the field level. The importance of these theories stems from their wide scope of applicability, which supports the neutrality of the models from a task viewpoint. Such "neutral" theories could be a widely reusable core of knowledge that can be augmented with more specific knowledge. For example, the generalized variables in the Bond-Graph theory (generalized flow, for example) could be reflected in high-level nodes in the OSN's hierarchy, which could then be refined.

We used Musarella in guiding the selection of the problem-solving methods of Faulty-II during the configuration of its task-method decomposition. Faulty-II is a Common-KADS model of the task, inference, and strategic reasoning.⁶ We then used Faulty-II as a problem solver with Musarella. Although the integration of the two resulted in flexible reasoning (because of the use of the multiple models), the integration is not yet fully exploited. The current problem-solving methods do not take full advantage of the specification of the conceptualization and its annotations. There is much room for encoding the general problem-solving methods that can reason about the conceptualization and also for experimenting with different strategies for using the framework. Although we have not fully evaluated our approach, it is a good starting point with respect to our requirements and goals.

Acknowledgments

We thank Richard Benjamins and Bob Wielinga for their contributions to this work and Gertjan van Heijst and Remco Straatman for comments on earlier drafts of this article. We also acknowledge B. Chandrasekaran, who provided insightful comments on this work and on the relation between tasks and domains in general. This research has been supported by the Dutch Government under code SKBS-A2.

References

1. M.R. Genesereth and N.J. Nilsson, *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann, San Francisco, 1987.
2. B. Chandrasekaran, T.R. Johnson, and J.W. Smith, "Task-Structure Analysis for Knowledge Modeling," *Comm. ACM*, Vol. 35, No. 9, Sept. 1992, pp. 124-137.
3. T. Bylander and B. Chandrasekaran, "Generic Tasks in Knowledge-Based Reasoning: The Right Level of Abstraction for Knowledge Acquisition," in *Knowledge Acquisition for Knowledge-Based Systems*, Vol. 1., B. Gaines and J. Boose, eds., Academic Press, San Diego, 1988, pp. 65-77.
4. B.J. Wielinga et al., "Towards a Unification of Knowledge-Modeling Approaches," in *Second-Generation Expert Systems*, J.-M. David, J.-P. Krivine, and R. Simmons, eds., Springer-Verlag, New York, 1993, pp. 299-335.
5. W.F. Punch and B. Chandrasekaran, "An Investigation of the Roles of Problem-Solving Methods in Diagnosis," *Proc. 10th Int'l Workshop Expert Systems and Their Applications*, EC2, Nonterre Cedex, France, 1990, pp. 25-36.
6. V.R. Benjamins, *Problem-Solving Methods for Diagnosis*, PhD thesis, Univ. of Amsterdam, Amsterdam, 1993.
7. T.R. Johnson, *Generic Tasks in the Problem-Space Paradigm: Building Flexible Knowledge Systems While Using Task-Level Constraints*, PhD thesis, Ohio State Univ., Columbus, Ohio, 1991.
8. T.R. Gruber, "A Translation Approach to Portable Ontology Specifications," *Knowledge Acquisition*, Vol. 5, No. 2, June 1993, pp. 199-220.
9. G. van Heijst et al., "A Case Study in Ontology Library Construction," to be published in *Artificial Intelligence in Medicine*, June 1995.
10. B. Chandrasekaran, "Models Versus Rules, Deep Versus Compiled, Content Versus Form: Some Distinctions in Knowledge Systems Research," *IEEE Expert*, Vol. 6, No. 2, Apr. 1991, pp. 75-79.



INSTRUCTION-LEVEL PARALLEL PROCESSORS

edited by H.C. Torng and Stamatis Vassiliadis

Explores the potential, design, and implementation of instruction-level parallelism (ILP) in modern processors. The introductions that begin the seven chapters in this book provide the background material necessary for comprehending the subjects studied. These chapters explain and provide solutions to the true data dependency problem and the control dependency problem and discuss machine organizations resulting from the compaction of instructions into VLIW. The book introduces superscalar processors that use dynamic schemes to detect and execute parallel instructions and addresses memory accesses, interrupt handling, and expected enhanced performance from ILP processors.

Chapters: Introduction; Dependencies Among Instructions and Their Resolution; Branch Handling; Code Scheduling and VLIW Machines Organizations; Superscalar Machine Organizations; Memory Accesses, Interrupt Handling, and Multithread Processing; Measurements of Instruction-Level Parallelism; References.

432 pages. 1994. Hardcover. ISBN 0-8186-6527-0.
Catalog # 6527-01 — \$48.00 Members \$36.00

INTERCONNECTION NETWORKS FOR HIGH-PERFORMANCE PARALLEL COMPUTERS

edited by Isaac D. Scherson and Abdou S. Youssef

Addresses the basic problems encountered in the design, analysis, use, and reliability of interconnection networks. These four problem areas are examined in the 10 chapters following the introduction to this book. The second chapter examines the fundamental questions of network structure, description, and construction. Subsequent chapters introduce interconnection topologies, discuss the four main classes of routing, present performance evaluation techniques, and deal with the mapping and embedding problem. Other chapters address the issues of survivability achieved by both hardware and software techniques and detail important research on the methods used to diagnose failures.

Chapters: Introduction, Foundations, Topologies, Permutation Routing, Nonuniform Routing, Deadlock-Free Routing and Multicasting, Performance Evaluation, Mapping and Embedding, Partitioning, Fault Tolerance, Fault Diagnosis.

816 pages. 1994. Hardcover. ISBN 0-8186-6197-6.
Catalog # 6197-01 — \$68.00 Members \$51.00

IEEE COMPUTER SOCIETY PRESS

▼ Call toll-free: 1-800-CS-BOOKS ▼

▼ Fax: (714) 821-4641 ▼

▼ E-Mail: cs.books@computer.org ▼

11. A. Abu-Hanna et al., "Functional Models in Perspective: Their Characteristics and Integration in Multiple Model-Based Diagnosis," *Applied Artificial Intelligence*, Vol. 8, No. 2, Apr.-June 1994, pp. 219-237.
12. A. Abu-Hanna, *Multiple Domain Models in Diagnostic Reasoning*, PhD thesis, Univ. of Amsterdam, Amsterdam, 1994.
13. J. Top and J.M. Akkermans, "Bond-Graph-Based Reasoning about Physical Systems," *Proc. Second AAAI Workshop on Model-Based Reasoning*, MIT Press, Cambridge, Mass., 1990, pp. 39-49.



Ameen Abu-Hanna is performing post-doctoral research at the Department of Social Science Informatics at the University of Amsterdam. He is currently working on the design of medical ontologies in the European Games-II project (General Architecture for Medical Knowl-

edge-Based Systems) within the Advanced Informatics in Medicine framework. His research interests include the engineering of problem solvers and domains, multimodel-based diagnosis, functional representation, the design and use of ontologies, and knowledge acquisition. He earned his PhD in 1994 from the University of Amsterdam, and his MSc in computer science in 1988 and his BSc with honors in computer engineering in 1985, both from the Technion in Haifa. He can be reached at SWI, Univ. of Amsterdam, Roetersstraat 15, 1018 WB Amsterdam; Internet: ameen@swi.psy.uva.nl.



Wouter Jansweijer is a member of the research staff in the Department of Social Science Informatics at the University of Amsterdam. He is currently working in the CEE Esprit project Kactus on knowledge sharing and reuse. His research interests include knowledge

structuring, the nature of problem solving, and knowledge acquisition. He earned his PhD in cognitive science from the University of Amsterdam in 1988. He can be reached at SWI, Fac. of Psychology, Univ. of Amsterdam, Roetersstraat 15, 1018 WB Amsterdam, The Netherlands; Internet: jansweij@swi.psy.uva.nl.

IEEE EXPERT