Available online at www.sciencedirect.com

SCIENCE DIRECT®

Advanced Engineering Informatics 18 (2004) 95–113

ADVANCED ENGINEERING INFORMATICS

www.elsevier.com/locate/aei

Physical concept ontology for the knowledge intensive engineering framework

Masaharu Yoshioka[a,*], Yasushi Umeda[b], Hideaki Takeda[c], Yoshiki Shimomura[d], Yutaka Nomaguchi[e], Tetsuo Tomiyama[f]

[a]*Hokkaido University, N14 W9, Kita-ku, Sapporo-shi, Hokkaido 060-0814, Japan*
[b]*Tokyo Metropolitan University, Tokyo 192-0397, Japan*
[c]*National Institute of Informatics, Tokyo 101-8430, Japan*
[d]*The University of Tokyo, Tokyo 113-8654, Japan*
[e]*Osaka University, Osaka 565-0871, Japan*
[f]*Delft University of Technology, AA Delft 2600 The Netherlands*

Abstract

Knowledge intensive engineering aims at flexible applications of a variety of product life cycle knowledge, such as design, manufacturing, operations, maintenance, and recycling. Many engineering domain theories are organized and embedded within CAD and CAE tools and engineering activities can be formalized as modeling operations to them. Since most of domain theories deal with the physical world and can be associated with physical concepts, a physical concept ontology can form a common ontology to integrate engineering models that are formed based on domain theories. This paper reports a physical ontology-based support system for knowledge intensive engineering called Knowledge Intensive Engineering Framework (KIEF) to integrate multiple engineering models and to allow more flexible use of them. First, the paper describes the physical ontology as the core of KIEF and an ontology-based reasoning system, called a pluggable metamodel mechanism, to integrate and maintain relationships among these models. The pluggable metamodel mechanism uses a metamodel that represents the designer's mental model about a design object as a concept network model. The designer builds and decomposes a functional hierarchy from functional specifications with an FBS (Function-Behavior-State) modeler. He/She then maps the functional hierarchy into a metamodel using physical features that are building blocks for conceptual design. Then, the pluggable metamodel mechanism enriches the information contained in the metamodel by using causal dependency knowledge about the physical world and by building and analyzing various engineering models. We demonstrate the power of KIEF by illustrating a design case performed on KIEF.
© 2004 Published by Elsevier Ltd.

Keywords: Ontology; Engineering knowledge; Theory integration; Model integration; Design object modeling

1. Introduction

In modern engineering processes, design models play crucial roles for designing and evaluating design solutions Designers now actually use various design modeling systems such as 3D-CAD and Finite Element Analysis (FEA) systems in a repetitive loop. These models represent the same design object and they are not independent of each other; for example, mesh data for FEA is closely related to shape data of a solid model. To manage consistency among these models, to easily transfer data from one system to another, and to propagate changes in one system to another, we need an integrated design support environment.

A number of research efforts have been made to integrate multiple design object models, beginning with support for data exchange. Initial Graphics Exchange Specification (IGES) [1] is a typical example of this standard mainly for 2D drawing layout and supports only format level data exchange. The next level was the concept of product model [2] that is now commonly used for model management across a variety of applications. Standard for the Exchange of Product model data (STEP) [3] defines a standard of product models. In the 1990s, Computer Aided Engineering (CAE) tools were integrated as extension modules into

* Corresponding author. Tel./fax: +81 11 706 7107.
E-mail address: yoshioka@ist.hokudai.ac.jp (M. Yoshioka).

1474-0346/$ - see front matter © 2004 Published by Elsevier Ltd.
doi:10.1016/j.aei.2004.09.004

the geometry-based CAD system, for example, CATIA [4] and Pro/Engineer [5]. However, this was rather one-to-one connection between a specific pair of CAD and CAE and was by no means a universal integration mechanism. One reason for this problem is the fact there is no truly multipurpose universal product model.

Integration of multiple engineering object models has two levels, viz. integration at data level and integration at knowledge level. The former is correspondence of data stored in various engineering object models, which comes from the fact that these models represent the same design object. With this, we can achieve consistency management and propagation of changes in data values among multiple engineering object models.

The latter results from deeper consideration about what those models are based on. An engineering object model focuses on a particular class of physical phenomenon and it does represent any other phenomena. Because there is a particular branch of physics (a background theory or a domain theory) behind the phenomenon of interest, the knowledge level description should be concerned with terminology and concepts that this domain theory contains, including such concepts as physical causal relationships and conceptual dependencies. A design modeling system embeds such a background theory and a design object model described in this system can be associated with concepts.

Handling and integrating engineering object models at this knowledge level have advantages over the data level integration. First, we can achieve true inter-operability of different engineering object models. Even if there is no direct correspondence between two models, the system might be able to reason out correspondence or conversion from one model to another using knowledge about domain theories. Second, it becomes possible to support designers in building, modifying, and using models at high intelligent level. These advantages are maximized by having an ontology-based reasoning. Understanding relationships among these models at the conceptual level will result from a clear ontological structure of engineering knowledge.

In this paper, we propose a physical concept ontology to build a large-scale engineering knowledge base as a kernel of the Knowledge Intensive Engineering Framework (KIEF) to support knowledge-intensive engineering [6]. Knowledge-intensive engineering is a new way to conduct a variety of engineering activities, including design, manufacturing, operation, maintenance, and recycling, in which knowledge accumulated in different engineering modeling systems is used in a flexible and integrated manner. KIEF can represent and manage relationships among knowledge for particular design cases. This information is useful to find out relationships among knowledge from multiple domains, such as dynamics and electricity in a mechatronics design case and to achieve multiple model integration at knowledge level. While our earlier reports [7–9] cover many other aspects of KIEF, the basis of the engineering ontology used in the framework has not been previously presented. Explicating the physical concept ontology is thus the main theme of this paper.

The rest of this paper is divided into five sections. Section 2 proposes a layered knowledge structure to integrate multiple domain theories, which was established in our past continuous effort to develop such a physical concept ontology aiming at knowledge level integration. In Section 3, the physical concept ontology and methodology to integrate domain theories by using this ontology are described. Section 4 illustrates a prototype system of KIEF with an example. Section 5 compares our approach with related work and Section 6 concludes the paper.

2. Engineering knowledge

2.1. Formalization of engineering activities

In engineering activities, engineers are faced with various problems for which they need to access to a wide variety of engineering knowledge To support this activity, in KIEF we organize knowledge in the form of domain theories (such as ‘electronics’ and ‘dynamics’) Based on domain theories, we have also developed computational support tools, such as CAD and CAE (we call these tools as *design modeling systems* in this paper) to be used in the engineering process.

However, since each domain theory has its own terminology and concepts, communication among experts in different domains is often difficult if not impossible. There is also a tendency to describe a theory precisely with a specific representation scheme for this particular theory, avoiding the use of a general scheme, which makes communication makes even more difficult. For example, structural analysis of a mechanical component with a complicated shape is carried out with Finite Element Method (FEM) that is based on domain theory ‘strength of material’. Similarly, the control model of a mechanical system is based on domain theory ‘control engineering’. Unfortunately, between these two, there is hardly any common concept except for force and stiffness. The necessary transportation of data, from a geometric modeling system to FEM and to the control simulation system, is difficult if we do not have deep understanding about these two theories and we only have data level integration.

Our aim is to build a general framework to integrate multiple design object models not only at the data integration level but also at the knowledge level of domain theories. The knowledge level integration will allow integrated use of multiple design modeling systems. Since each design modeling system is associated with a domain theory that defines terminology and concepts, relationships among systems are maintained by using these terminology and concepts.

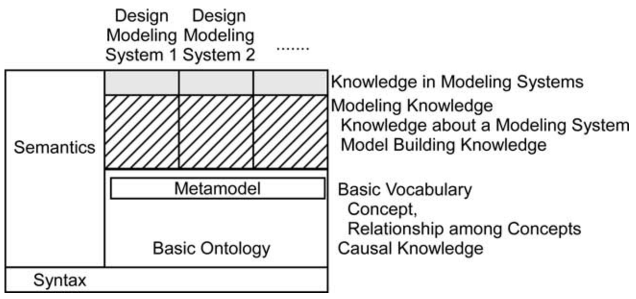


Fig. 1. The model of engineering knowledge structure.

From the viewpoint of information processing, engineering activities can be formalized as model operations within design modeling systems. As long as the designer is interested only in computation results, usually it is not required to have deep understanding about the domain theory embedded inside the modeling system. However, for instance, once there is a trade-off problem over results of different CAE systems, a designer should understand relationships among the models represented by these CAE systems. For this purpose, integration at the data level is insufficient, because this will boil down to understanding of relationships among their domain theories, such as physical causality relationships.

2.2. Structure of engineering knowledge

Our model of engineering knowledge structure was established through our past continuous research efforts. Interested readers should refer to [7,9,10,11]. This model assumes a layered structure depicted in Fig. 1.

The lowest layer describes the syntax to represent the knowledge. The upper layers describe semantics of the knowledge and both of the knowledge level integration and the data level integration take place here. These layers are divided into three sub-layers.

The first basic ontology layer provides the common vocabulary (or terminology) to represent fundamental physical concepts. These concepts have relationships among them such as causality and super-sub hierarchy. The concepts included in this basic ontology layer are used to represent relationships among design modeling systems.

The data level integration is based on data correspondence of the same data in different design modeling systems. To do this, a conceptual model of the design object is built at this basic ontology layer. This conceptual model is called a '*metamodel*' [10,12] and represents the design object as a network of concepts.

The second middle layer contains knowledge about modeling systems and model building knowledge to describe, e.g. how to use each of these modeling systems, including input and output conditions of these systems. We call this sort of knowledge 'modeling knowledge.' The third upper layer describes modeling system specific knowledge that is embedded in each modeling system [7].

The basic principle of the metamodel concept is explained here. Fig. 2 shows the relationship among different design models for the analysis of the deformation of a 'Beam' that is used for the head design of a magnetic hard disk. The top level, domain theory level, is a set of design modeling systems that embed background level knowledge. The bottom half of Fig. 2 shows the metamodel of this design object with the vocabulary defined in the basic ontology. Then, modeling knowledge is used to define relationships between concepts in the metamodel and those in a design modeling system. For instance, the metamodel

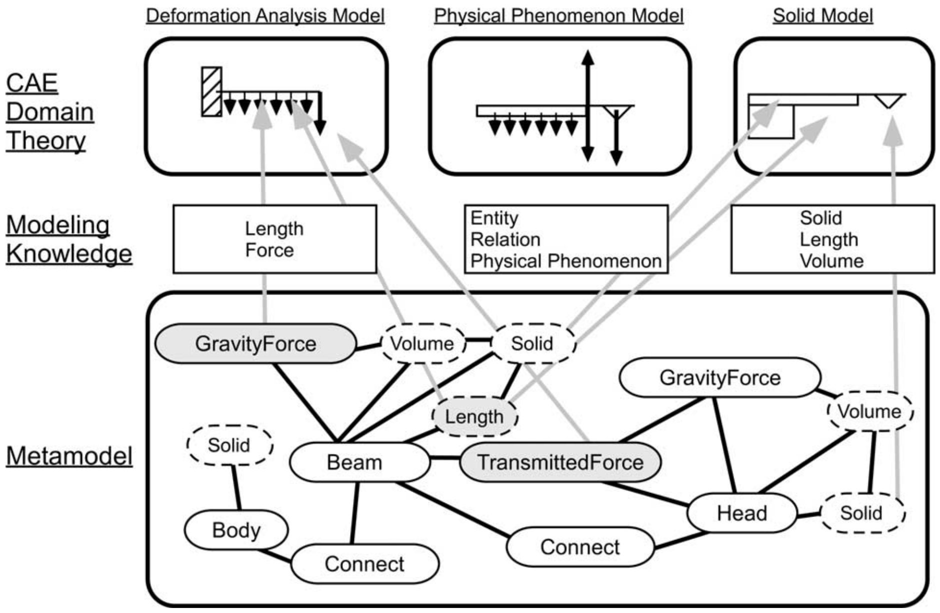


Fig. 2. Relationships between concepts used in different models.

represents such a statement as ‘Beam’ is ‘connected’ to ‘head.’

An example of modeling knowledge is a list of concepts that can be handled with a design modeling system, together with how to translate data in the metamodel into data for the design modeling system. In this case, a solid model deals with concepts ‘Solid’, ‘Length’, and ‘Volume’, shown by dotted line nodes in the metamodel. A deformation analysis model deals with concepts ‘Length’ and ‘Force’, depicted by gray-colored nodes.

Through the metamodel, we can find out relationships among attributes in the solid model and those in the deformation analysis model. The relationships among attributes in these two models include direct ones, such as relationships between ‘Length’ in the solid model and ‘Length’ in the deformation analysis, and indirect ones, such as relationships between ‘Volume’ of ‘Head’ in the solid model and ‘TransmittedForce’ in the deformation analysis model. The latter requires additional knowledge such as ‘equilibrium of force’ and ‘mass equals volume multiplied by density’. This indirect correspondence is more flexible and useful than direct ones. To do this, we need a physical phenomenon model that deals with the causal relationships about force. In addition, since the value of ‘TransmittedForce’ may change when the designer modifies the shape of ‘Head,’ it is necessary to build a dependency network among these data.

The metamodel realizes both direct and indirect data correspondences for data level integration through this mechanism. It also realizes knowledge level integration using the basic ontology and modeling knowledge, resulting in more flexible model management. For example, through direct correspondences among concepts, the metamodel tells us that data for ‘Length’ in the solid model and data for ‘Length’ in the deformation analysis model should be same. It also reasons out that data for ‘TransmittedForce’ in the deformation analysis is calculated from data for ‘Volume’ in the solid model by using knowledge about force. In this way, knowledge about physical causal dependency allows a more flexible use of different domain theories.

3. Layered knowledge structure for the knowledge intensive engineering framework (KIEF)

3.1. Engineering activities supported by KIEF

KIEF is a framework to integrate design modeling systems that has embedded knowledge about domain theories. KIEF also maintains consistency among integrated design object models. As we discussed in the previous section, KIEF uses physical concept ontology, modeling knowledge, and the metamodel for this integration.

This section illustrates engineering activities with KIEF and discusses types of knowledge that are required for KIEF. The following is an outline of a design process with KIEF.

(1) Construction of an initial metamodel
First, an initial metamodel must be built with, for example, a function modeling system such as FBS (Function-Behavior-State) modeling [13]. This initial metamodel should have basic information about the design object, such as its conceptual structure and preliminary layout (topological connection), but not necessarily attribute information, such as shape, at this stage.
(2) Enrich metamodel information
The initial metamodel does not necessarily contain sufficient information about the design object for further processes. We need enrich information of the metamodel with such knowledge as causality knowledge to reason out feasible physical phenomena that may occur to the design object.
(3) Synthesis and analysis based on design modeling systems
We evaluate design solutions using different design modeling systems. In so doing, KIEF assists the modeling process by using input and output information about design modeling systems and maintains the consistency among different design object models.

The first two activities are operations in the basic ontology layer in Fig. 1. The third step uses the knowledge in modeling systems in Fig. 1 through modeling operations (operations to modeling systems). To control modeling processes (such as model building, model evaluation, etc.), the modeling knowledge in Fig. 1 is used. Modeling knowledge corresponding to a modeling system typically contains concepts necessary to build a model used as an input for this system and concepts to be obtained as a result of modeling (or computation).

The following is the summary of the knowledge of KIEF and will be explained in detail in the subsequent sections.

- Basic vocabulary
  Physical concept ontology: This ontology is the basic ontology for representing a design object model (Section 3.2).
  Physical feature: Physical feature library serves as a mechanism library containing building blocks for building a metamodel and to find a causal relationship among entities to check whether or not some physical phenomena may occur to the design object (Section 3.3).
- Modeling knowledge
  Knowledge about a modeling system: This knowledge describes input and output information of the modeling system (Section 3.4.1).
  Model building knowledge: This knowledge supports the construction of a design object model in the modeling system. In this paper, we introduce

model library as an example of this kind of knowledge (Section 3.4.4).

3.2. Physical concept ontology

Integrating multiple design object models means knowledge level integration of domain theories, which requires explicit representation of physical phenomena. To do so, the physical concept ontology is introduced. This is the most fundamental knowledge component in the basic ontology layer and is stored in a knowledge base called '*concept base*'.

The concept base includes the following five conceptual categories.

Entity:	An entity represents an atomic physical object.
Relation:	A relation represents a relationship among entities to denote a static structure.
Attribute:	An attribute is a concept attached to an entity and takes a value to indicate the state of the entity.
Physical phenomenon:	A physical phenomenon designates physical laws or rules that govern behaviors.
Physical law:	A physical law represents a simple relationship among attributes.

Each domain theory has its own concept system and the details of its concept definitions may be different. For example, the dynamics theory requires a simple description of entity; i.e. an entity in terms of mass and inertia. The theory for gear motion analysis requires specific concept categories to represent different types of gears. The theory for electric circuit requires totally different concept categories, such as conductor, resistance, capacitance, and inductance. The concept base organizes these concepts in an abstract-concrete class hierarchy with multiple inheritances; i.e. we can define multiple abstract super class concepts for one concept. This ontological structure allows various domain theories to share these concepts, while specialized concepts can be incorporated without difficulties into the concept base.

In the following, each conceptual category is explained. Formal definitions of these concepts are given in Appendix A.

3.2.1. Entity

An entity represents an atomic physical object In order to describe the abstractconcrete relationship among concepts, we define an entity concept with its "*name*" and "*supers*" that is for defining abstract concepts of a defined one. The concept "*Entity*" is the most abstract concept in this category. An example of an entity is as follows.

Name:	Gear(?e)
Supers:	MechanicalParts(?e),Mass(?e)

3.2.2. Relation

A relation represents a relationship among entities to denote static structure. We define a relation concept with its "*name*" and "*supers*" as the entity concept. The concept "*Relation*" is the most abstract concept in this category.

In addition to these two slots, we need to define a relation with references to entities. We use "*HasRelations*" slot and "*HasRelation*" predicate for this purpose. The first term of the HasRelation atom is the name of the defined relation and other terms are the reference names of the entities. Therefore, HasRelation term that is used to define the relationships among n entities has $n+1$ terms. This definition is also used for explaining whether this concept is directional (e.g. "on") or not (e.g. "connect"). If a relation concept is directional, we describe one predicate logic atom (e.g. HasRelation(on, upper, lower)). If it is not directional, we describe two or more predicate logic atoms to represent equivalent relations (e.g. HasRelation(connect, object1, object2), HasRelation(connect, object2, object1)).

Name:	Meshed(?meshed)
Supers:	Relation(?meshed)
HasRelations:	HasRelation(?meshed, ?gear1, ?gear2), HasRelation(?meshed, ?gear2, ?gear1)

3.2.3. Attribute

An attribute is a concept attached to an entity and takes a value to indicate the state of the entity We define an attribute concept with its "*name*" and "*supers.*" The concept "*Attribute*" is the most abstract concept in this category.

In addition to these two definitions, we use "*Statements*" to describe additional information. Statements includes dimension of the attribute and definitional relation with other attributes (e.g. acceleration is a time differential of velocity).

Name:	Acceleration(?a)
Supers:	Attribute(?a)
Statements:	DifferentialOf(?a, Velocity, Time), UnitOf(?a, ?, 'm/s$\hat{2}$'), DimensionOf(?a, (L M T I Temp), (1 0–2 0 0))

3.2.4. Physical phenomenon

A physical phenomenon designates physical laws that govern behaviors We define a physical phenomenon concept with its "*name*" and "*supers.*" The concept "*PhysicalPhenomenon*" is the most abstracted concept in this category and all concepts should be a concrete concept of it.

In many cases, a physical phenomenon represents relationships among attributes of entities. "*Entities*" and “*Attributes*” define related entities and attributes of the defined phenomenon. For a complicated physical phenomenon that is difficult to represent as a relationship among attributes, we can define it by using only “*Entities*” information. “*Statements*” describe the relationships among entities, attributes, and the phenomenon. In this slot, “*OccurTo*” predicate describes the relationship between the phenomenon and related entities. “*HasAttribute*” predicate describes the relationship among related attributes and related entities.

In order to represent complicated phenomena affected by multiple physical laws, we divide the definition of physical phenomenon and physical law, such as Newton’s law. “*PhysicalLaws*” is used to describe related physical laws.

Name:	LinearMotion(?p)
Supers:	Motion(?r)
Attributes:	Force(?f), Mass(?m), Position(?pos), Acceleration(?acc), Velocity(?vel)
Entities:	Mass(?object)
PhysicalLaws:	SecondLawOfNewtonLaws(?f, ?m, ?acc)
Statements:	OccurTo(?p, ?object), HasAttribute(?f, ?object), HasAttribute(?m, ?object), HasAttribute(?pos, ?object), HasAttribute(?acc, ?object), HasAttribute(?vel, ?object)

3.2.5. Physical law

A physical law represents a simple relationship among attributes “*Name*” and “*Attributes*” define the name and related attributes of the defined physical law, respectively. “*Expression*” defines the relationship among attributes by using mathematical equation.

Name:	SecondLawOfNewtonsLaw
Attributes:	f_Force, m_Mass, a_Acceleration
Expression:	Sigma(f) = m*a

3.3. Building a metamodel

In KIEF, a metamodel represents a design object as a network of concepts At the beginning of an engineering process (such as design), an initial metamodel will be built. This is done in the following manner. First, a designer picks up physical concepts relevant to the design object from the concept base and instantiates them. A metamodel is built as a network of these instantiated physical concepts. However, building a metamodel from scratch is not an easy task, as this metamodel quickly gets complicated. It is better to prepare building blocks for physical entities and physical causality knowledge that finds causal relationships among entities.

For this purpose, the “*physical feature*” library serves as a mechanism library containing building blocks to build a metamodel. Just like geometric features [14] used as building blocks to define mechanical components in a geometric CAD system, physical features make it easier to define complicated design objects that involve physical phenomena. A physical feature is a network of concepts that includes a set of physical phenomena and conditions (such as existence of entities and related physical phenomena) for invoking the phenomena.

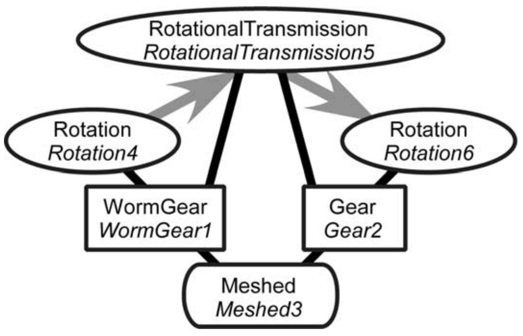


Fig. 3. Example of a physical feature.

Fig. 3 shows an example of a physical feature that represents a mechanical configuration consisting of a gear and worm gear pair for rotational power transmission, together with related physical phenomena. All nodes are instances of concepts in the concept base (the class name and *instance name* are described in each node in Fig. 3). Rectangular nodes represent entities and rounded rectangular nodes correspond to relations. Oval nodes represent physical phenomena. Non-directed links from physical phenomena and relations correspond to reference from each concept (e.g. relation ‘Meshed’ requires two entities and ‘WormGear’ and ‘Gear’ are reference objects). Directed gray links between physical phenomena are causal relationship links.

The designer can built an initial metamodel by explicitly combining physical features. However, this initial model may not contain sufficient information for subsequent processes to, for example, evaluate performance with a design modeling system, because this design object may invoke physical phenomena besides those that can be derived from the information included in the physical features explicitly selected by the designer. In other words, there could be unintended physical phenomena. Therefore, a reasoning mechanism must detect these unintended physical phenomena by checking hidden or implicit causal relationships.

The metamodel system reported in [12] had a mechanism to do so based on Qualitative Process Theory (QPT) [15]. However, in this old metamodel system, the description of physical phenomena lacked generality. For example, a physical phenomenon ‘RotationalTransmission’ that may occur to different types of mechanisms had to be defined individually for these different mechanism such as ‘RotationalTransmissionByGearPair,’ and ‘RotationalTransmissionByPulley,’ and they were all recognized differently.

KIEF solved this problem by improving the standard QPT based reasoning in the following two points. First, the occurrence of a physical phenomenon is checked by the prerequisites of physical features, rather than the prerequisites of the physical phenomenon. KIEF uses the conditional information of a physical feature (instead of a physical phenomenon) as a prerequisite to check the occurrences of a physical phenomena described in the feature.

Second, KIEF makes maximum use of the abstract-concrete hierarchy of concepts stored in the concept base in checking the occurrence of physical phenomena. This is conducted by graph matching between the metamodel and the prerequisites part of a physical feature. Every node in these two networks belongs to an abstract-concrete hierarchy, so for instance a physical phenomenon that can happen to a class of entities may happen to entities belonging to its subclass.

A physical phenomenon sometimes requires another physical phenomena to get invoked. For instance, for ‘RotationalTransmission5’ and ‘Rotation6’ of the gear in Fig. 3) to happen, the worm gear must be rotated (by an external means) and ‘Rotation4’ should take place. In other words, ‘Rotation6’ occurs, if and only if ‘Rotaion4’ happens, while within this physical feature itself there is nothing actually invoke ‘Rotaion4.’ This means we need to classify related physical phenomenon into two types. One is physical phenomena that are invoked as a consequence of this physical feature (e.g. ‘RotationalTransmission5’ and ‘Rotation6’ in Fig. 3). The other type is phenomena that are necessary to check their conditions to see if this physical feature can happen (e.g. ‘Rotation4’ in Fig. 3). By using classification, we can treat a physical feature as knowledge to check whether physical phenomena may occur or not.

Prerequisites: Conditions about entities and relations, and physical phenomena that are used to check their conditions. Needed to see if this physical feature happens or not.

Consequence: Physical phenomena that are invoked by this physical feature.

3.4. Handling domain theories

3.4.1. Knowledge about a modeling system

As discussed in Section 2.2, we use design modeling systems as sources for domain theories To utilize domain theories that are embedded in design modeling systems such as CAE, we need modeling knowledge that is used for filling the gap between the concepts used in the metamodel and knowledge in each modeling system (hatched area in Fig. 1). We have already proposed the concept of “*pluggable metamodel mechanism* [9] to connect a design modeling system to KIEF.

The pluggable metamodel mechanism uses “*Knowledge about a Modeling system*” to integrate design modeling systems and to support building a design object model. To connect a design modeling system, information about its inputs and outputs is necessary. To support modeling processes, knowledge for selecting related concepts from a metamodel and knowledge about data exchange methods are necessary. Therefore, knowledge about a modeling system has the following four slots using concepts from the concept base.

- Available concepts
  This list contains concepts that are used for constructing a model for this design modeling system (input concepts of the design modeling system).
- Computable concepts
  This list contains concepts that can be computed by using the knowledge embedded in this design modeling system (output concepts of the design modeling system).
- Related concepts
  Concepts in this list are used to select related concepts from a metamodel. Each concept in this list should be an abstract concept of one or more concepts in the available concepts.
- Data exchange method
  This description is used to obtain data in a specific format from this design modeling systems.

Table 1 is an example of a description about a beam modeling system. ‘Force’ is an abstract (super) concept of ‘ConcentratedForce’, and ‘DistributedForce.’ By using the related concept information, KIEF selects any force that is an instance of concepts whose abstract class is ‘Force’; e.g. ‘GravityForce’ and ‘TransmittedForce.’ Since a model in the beam modeling system should be described with available concepts, any forces that are used to make a model should be categorized into ‘ConcentratedForce’ or ‘DistributedForce.’

3.4.2. Integrating a design modeling systems into KIEF

The pluggable metamodel mechanism supports building a design object model to be used for a particular design modeling system, obtaining information from the metamodel of the design object In addition, the pluggable metamodel mechanism stores and manages the computation result to share it with other design modeling systems.

Table 1
Knowledge about a beam modeling system (Example)

Name of the slot	Contents
Available concepts	Beam, HingedSupport, ConcentratedForce, DistributedForce, etc
Computable concepts	ShearingForceDiagram, BendingMoment-Diagram
Related concepts	Entity, Relation, Force method
Data exchange	Length of beam is calculated by using 2D shape or solid model

(1) Support building a design object model (i.e. input to a design modeling system)
   (a) Build an "*aspect model*."
   An *aspect model* is a subset of a metamodel, which represents a design object model for a design modeling system in the concept network level. To build an aspect model, related concepts are used to select candidate concepts. Then, available concepts are used to actually build an aspect model.
   (b) Build a design object model for a design modeling system.
   After building an aspect model, the pluggable metamodel mechanism collects information to build a design object model for the design modeling system, which is more concrete than the aspect model. A data exchange method is used to obtain information from other design modeling systems. If there is no information about the required concepts, the pluggable metamodel mechanism tries to search other design modeling systems that can compute required information by looking up information about computable concepts. In addition, KIEF also allows a more flexible method to build a design object model in a design modeling system by combining model fragments, when a domain theory is well organized enough to represent each knowledge fragment as a model fragment that corresponds to available concepts. For example, to use a behavior simulation system based on analytical equations, a model fragment corresponds to one or more equations that represent a particular physical law or phenomenon. KIEF collects these equations and generates a set of equations to be solved by the design modeling system.
(2) Store and manage the results (i.e. output from a design modeling system)
   (a) Use knowledge in a design modeling system.
   The design modeling system generates (computes) new information about a design object by using its embedded knowledge. Computational results are exported to the pluggable metamodel mechanism. In order to maintain relationships between the input information and the computed output information, the pluggable metamodel mechanism creates dependency relationships between the input concepts and computed concepts.

By using these dependency relationships, the pluggable metamodel mechanism maintains consistency of the metamodel; e.g. when information in the metamodel is modified, the pluggable metamodel mechanism detects the information that has to be recomputed.

Fig. 4 shows an example of a modeling process with the pluggable metamodel mechanism. Nodes and links used in Fig. 3 (e.g. rectangular nodes and non-directed links) have the same meaning. Dotted oval nodes represent attributes and directed gray links between attributes are derivation relationship links.

First, the concepts described in the related concepts are selected as candidates to build an aspect model. For a beam model case, all entities, all relations, and 'GravityForce7' are selected as candidates. After that, the designer builds an aspect model by using the concepts described in the available concepts; i.e. the designer should choose related concepts in the metamodel to be included in the beam model and relate each chosen concept to an available concept. For example, in order to analyze deformation of the shaft, 'Shaft2' is mapped to 'Beam' concept. We use tentative concept category 'Shaft&Beam' that is a concrete concept of 'Shaft' and 'Beam' for representing this mapping.

Second, the pluggable metamodel mechanism collects information to build a design object model for the beam analysis system. In this case, 'Solid8' is used for calculating the information of 'Beam.' Therefore, the pluggable metamodel mechanism creates a dependency relationship between 'Solid8' and 'ShearingForceDiagram9' that is calculated by the beam analysis system.

Based on this modeling process, we can use each design modeling system as a compartmentalized knowledge base system and maintain the consistency based on derivation relationships by using the pluggable metamodel mechanism.

3.4.3. Exchanging data among design modeling systems

To define a data exchange method for design modeling system requires to define a standard method to access data in design modeling systems. Since there can be different kinds of representation methods (data structure) for each attribute (e.g. solid, face, force, etc.), KIEF should be able to handle the differences in representation.

To absorb different data structures, we should define a standard data structure for each attribute. However, some attributes, which have a complex data structure, are difficult to have these standard definitions (e.g. solid and free form surface). Therefore, we define the standard in two ways.

- Standard access methods to obtain data
  For some attributes that have complex data structure, instead of defining the data structure itself, we define standard access methods to extract related attributes (e.g. obtaining attribute information of a vertex from the information of a solid). This task can be made easier if we use a product data model such as STEP.
- Standard data structure
  For other attributes that have simple data structures, such as vertex, we define standard data structures.

By using these standard access methods, the pluggable metamodel mechanism can obtain and exchange data between plugged-in design modeling systems.

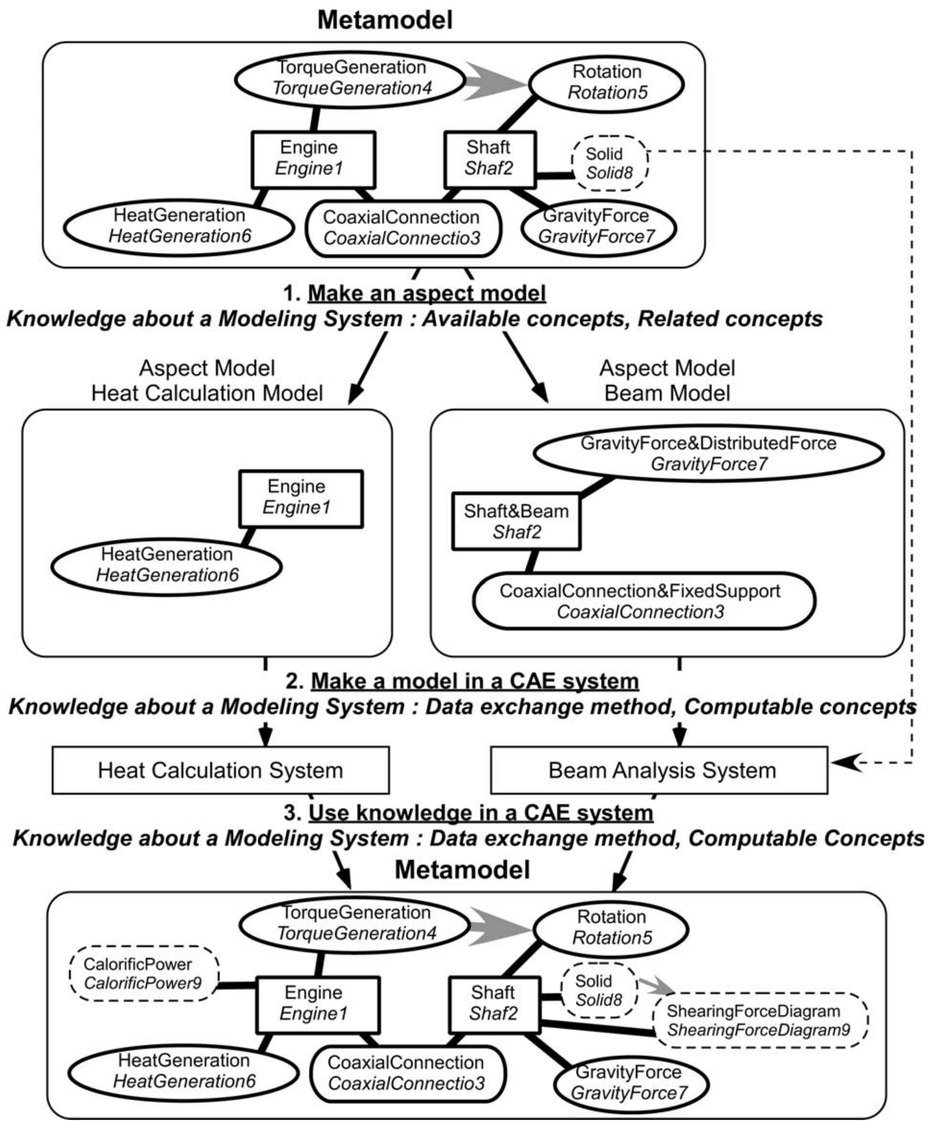

Metamodel
TorqueGeneration TorqueGeneration4
Rotation Rotation5
Engine Engine1
Shaft Shaf2
Solid Solid8
HeatGeneration HeatGeneration6
CoaxialConnection CoaxialConnectio3
GravityForce GravityForce7
1. Make an aspect model
Knowledge about a Modeling System : Available concepts, Related concepts
Aspect Model Heat Calculation Model
Aspect Model Beam Model
GravityForce&DistributedForce GravityForce7
Shaft&Beam Shaf2
CoaxialConnection&FixedSupport CoaxialConnection3
2. Make a model in a CAE system
Knowledge about a Modeling System : Data exchange method, Computable concepts
Heat Calculation System
Beam Analysis System
3. Use knowledge in a CAE system
Knowledge about a Modeling System : Data exchange method, Computable Concepts
CalorificPower CalorificPower9
ShearingForceDiagram ShearingForceDiagram9

Fig. 4. Example of modeling process by using the pluggable metamodel mechanism.

3.4.4. Model construction support with a model library

When a domain theory is well organized enough, we can support modeling processes more flexibly by defining a model fragment that can serve as a building block to build a design object model for the concerned design modeling system. *Model library* stores model fragments for different design modeling systems. These model fragments correspond to concepts defined in the concept base. Model library directly corresponds to 'Knowledge in modeling systems' in Fig. 1.

When the user tries to build a model in a design modeling system that has model library, the pluggable metamodel mechanism collects corresponding model fragments from the model library and assembles these fragments into a model for the system.

Below, we describe some examples of the model library.

- Numerical equation model library
  We can define a symbolic equation for each physical law. For instance, to simulate behavior of a design object, the pluggable metamodel mechanism collects related physical laws by using physical phenomenon information in the metamodel. Then, the system collects attributes and builds equations by using these numerical equation model fragments.
  However, when there are many physical phenomena to take into account, a large number of equations are collected. In such a case, it is difficult to solve all collected numerical equations and a user may need to simplify the model to obtain approximate results. In order to support such modeling operations, more information and knowledge would be necessary to generate an appropriate engineering model for particular situation [7].

- Qualitative process theory model library
  We also define qualitative parameter relationships between attributes of physical phenomena and physical

Table 2
Knowledge in the concept base

Physical concepts	Number
Entities	800
Relations	150
Physical phenomena	500
Attributes	600
Physical laws	300

laws. To simulate behavior of a design object qualitatively, the pluggable metamodel mechanism collects related physical phenomena and physical laws from the metamodel. Then, the system generates a qualitative parameter network model. Behavior simulation is conducted by a qualitative physics reasoning system [12] based on QPT [15].

4. Applications based on physical concept ontology

Based on the discussion in previous section, we collected concepts in the concept base and Table 2 shows numbers of concepts already stored in a concept base. We employ two approaches to construct this knowledge base.

Library based approach: We collect concepts based on a textbook; e.g. we use [16] for physical law definition.

Example based approach: We also collect concepts base on the analysis on particular design examples.

These numbers of concepts are not enough to handle wide varieties of engineering problems at this moment. However, we think this knowledge is enough to evaluate KIEF through example design problems.

In this section, we illustrate applications of KIEF to demonstrate the power of the proposed ontology. We first review the Qualitative Process based Abduction System (QPAS)[1] [20,21], which generates design solution candidates from state transition information. We discuss that the proposed layered structure ontological knowledge is indeed useful to develop such a system. Then, we describe an implementation of a design support system on KIEF.

4.1. Qualitative process based abduction system

A Qualitative Process based Abduction System (QPAS) is a reasoning system to propose candidate design solutions from a series of qualitative state transitions [21]. QPAS originally used the knowledge base of a metamodel system [12] based on Qualitative Process Theory (QPT) [15]. QPT is a process centered qualitative reasoning theory and is more universal than confluence based qualitative reasoning theories [22]. As we discussed in 3.3, this system had a flexibility problem in knowledge representation. Therefore, we modified QPAS to fit the concept base. Since QPAS uses QPT Model Library, we first briefly review this library.

4.1.1. Qualitative process theory model library

The QPT model library has model fragments that are defined below. Each model fragment in QPT corresponds to a physical law or physical phenomenon. This fragment has the following three slots.

Name: Name represents a corresponding physical phenomenon or physical law.
Attributes: Attributes that are used to represent state transitions. Since QPT only deals with attributes that may change during simulation, a subset of attributes is selected from the definition of a corresponding physical law or physical phenomenon.
Expression: "Expression" represents parameter relationships between two attributes with the following two qualitative relationships.

- "Quantity relation"(Q+, Q−) represents a proportional relationship between two attributes. Q+ means that, if either of the parameters increases, the other parameter will also increase. Q− means that, if either of the parameters increases, the other parameter will decrease.
- "Influence relation"(I+, I−) represents a qualitative differential relationship. The first parameter is the target and the second parameter is the condition that decides whether the target parameter will increase or decrease. I+ means that the target parameter increases if the second parameter is positive and that target one decreases if the second one is negative. I− is vice versa.

Below is an example of a QPT model library that corresponds to a physical law described in Section 3.2.5.

Name:	SecondLawOfNewtonsLaw
Attributes:	f_Force, a_Acceleration
Expression:	f Q+a

4.1.2. Algorithm of QPAS

QPAS uses the QPT library, attribute and physical phenomenon of the concept base and the physical feature library to generate candidate design solutions from a given state transition We here illustrate the algorithm of QPAS through an example of designing shaft rotation mechanism (Fig. 5).

First, the designer inputs a state transition table as an initial requirement. We use one or more entities that play an important role to define its behavior and their parameters to define this state transition. We call these parameters '*functional parameters*'. In QPT, a parameter is defined with qualitative values which consist of '*landmarks*'

[1] In design research community, many researchers regard a candidate solution generation process as abduction (hypothetical reasoning) [17–19].

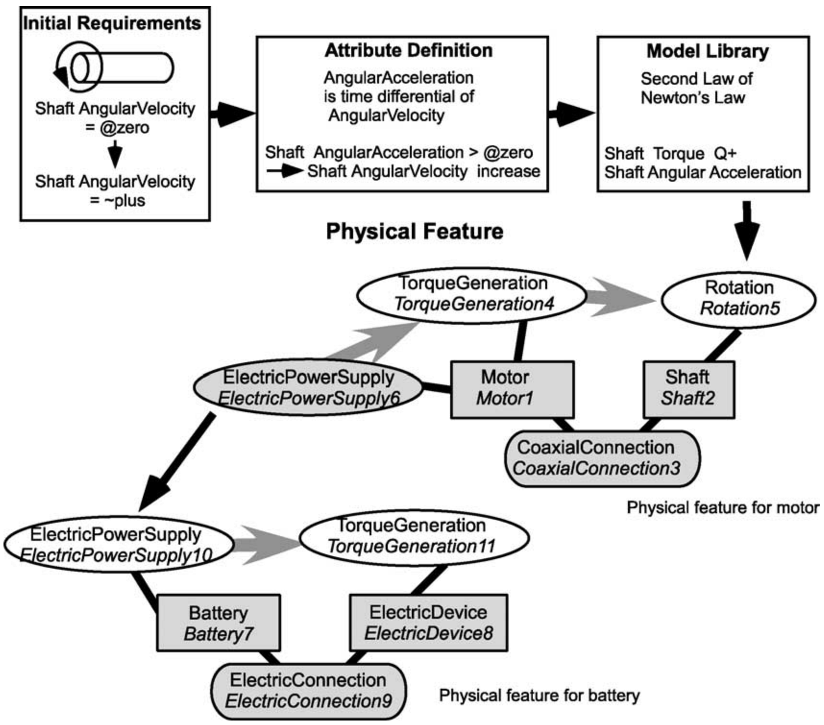

Initial Requirements
Shaft AngularVelocity = @zero
Shaft AngularVelocity = ~plus
Attribute Definition
AngularAcceleration is time differential of AngularVelocity
Shaft AngularAcceleration > @zero → Shaft AngularVelocity increase
Model Library
Second Law of Newton's Law
Shaft Torque Q+ Shaft Angular Acceleration
Physical Feature
TorqueGeneration TorqueGeneration4
Rotation Rotation5
ElectricPowerSupply ElectricPowerSupply6
Motor Motor1
Shaft Shaft2
CoaxialConnection CoaxialConnection3
Physical feature for motor
ElectricPowerSupply ElectricPowerSupply10
TorqueGeneration TorqueGeneration11
Battery Battery7
ElectricDevice ElectricDevice8
ElectricConnection ElectricConnection9
Physical feature for battery

Fig. 5. Example of QPAS reasoning process.

and '*interspaces*'. A '*landmarks*' represents characteristic value of given parameters. For example, let us think about temperature of water, 'boiling point' and 'melting point' are landmark of temperature, and temperature below 'melting point', temperature between 'melting point' and 'boiling point', and temperature over 'boiling point' are interspaces of it.

In this example, we use entity 'Shaft' and parameter 'AngularVelocity' is selected as its functional parameter. The first state "Shaft AngularVelocity = @zero" ("@zero" represents angular velocity(ω) is at a landmark $\omega = 0$) and the second state "Shaft AngularVelocity = ∼plus"̂(∼plus¤ " represents angular velocity (ω) is in an interspace $\omega > 0$). The top left part of Fig. 5 shows an example of state transition.

From this transition table, QPAS finds out candidate physical phenomena that may affect the functional parameters to achieve the designated state transition from the QPT model library. In addition, attributes that have a time differential relationship of function parameters are also candidate functional parameters to find physical phenomena. We use the concept base to search candidate functional parameters. In this example, since 'AngularAcceleration' has time differential of 'AngularVelocity,' 'AngularAcce-AngularAcceleration' is selected as a candidate functional parameter and physical law 'Second law of Newton's law' is selected to construct a candidate solution (top right part of Fig. 5).

After finding out a candidate physical phenomenon and/or physical law, QPAS searches a possible candidate design structure by selecting physical features that include the candidate physical phenomenon and/or physical law as intended physical phenomena. Then, the designer selects one physical feature from the candidates, and the system constructs a solution candidate model by merging the selected physical feature with entity information used in the state transition table. In this example, since physical feature 'motor' has physical phenomenon 'Rotation' that associates with 'Second law of Newton's law' (middle part of Fig. 5), "electrical motor" is selected as the candidate physical feature. In this merging process, an entity that is used to define a functional parameter is merged with an entity that the selected physical phenomenon affects. In this example, 'Shaft2' are merged with entity 'Shaft' that is used to define the functional parameter.

Since some physical phenomena require state changes of other attributes to achieve the desired state transition, the state transition table is updated by using attribute relationships. In this example, 'Torque' of shaft should be '∼plus' to achieve the desired state transition and this attribute becomes a functional parameter. This physical feature selection process reiterates until all functional parameters have appropriate effects to achieve state transitions. In each selection process, a selected physical feature is merged into the solution candidate model. In this example, since physical feature 'motor' has physical phenomenon 'TorqueGeneration' to achieve the state transition of 'Torque,' no other physical feature is selected for this process.

Finally, QPAS checks whether all physical features fulfill the condition to invoke intended physical phenomena.

When there is one or more physical feature(s) that do not fulfill their condition, QPAS also searches physical features that will fulfill their conditions and merge them into the solution candidate model (bottom part of Fig. 5).

4.1.3. Discussion

In the improved QPAS that uses physical features instead of physical phenomena, we integrate knowledge defined in different knowledge structure layers (i.e. QPT model library, attribute definitions in the concept base, and physical features) for solving a design problem. This means that the knowledge level integration actually is done at these three different layers; i.e. the concept base and physical features in the basic ontology layer, the knowledge about a modeling system in the modeling knowledge layer, and the model library and knowledge embedded in design modeling systems in the knowledge in modeling systems layer.

In contrast, the previous QPAS used these types of knowledge defined in one format. Since we have a less number of attribute concepts and physical phenomena and laws compared with the number of mechanism libraries, the former two types of knowledge have more reusability than the latter. In this layered knowledge structure, the new QPAS is better than the previous one in terms of knowledge reusability.

4.2. A Design support system based on KIEF

4.2.1. Architecture

To demonstrate the usage of proposed physical concept ontology, we implemented a design support system based on KIEF. KIEF and the design support system were implemented in VisualWorks[2] \Smalltalk

Fig. 6 shows the architecture of KIEF. The concept base provides ontological knowledge to represent a metamodel. The pluggable metamodel mechanism integrates design modeling systems by using the knowledge about modeling systems described in Section 3.4. The interface of a modeling system is used for exchanging data from a design modeling system to the pluggable metamodel mechanism.

The current implementation of KIEF is connected to seven external design modeling systems; i.e. a qualitative physics reasoning system [12], ProEngineer[3] (a solid modeler and its FEM preprocessor), a 2D draw modeling system, an FBS modeler, QPAS, a catalog-retrieving system, and Mathematica[4]–based engineering analysis systems (such as a surface tension analyzer described in the example below).

4.2.2. Design process on KIEF

The design process on KIEF is explained as follows First, the designer selects physical features and combines them to build an initial metamodel. We can use an FBS (Function-Behavior-State) modeler [13] and QPAS to support this process. In this implementation of KIEF, we do not deal with aggregation relationships between entities (e.g. a 'Gear' is a part of a 'GearBox'). Second, KIEF reasons out physical phenomena that can possibly happen to this design object, including both explicitly intended ones described in the initial metamodel and unintended ones not assumed in the metamodel. This is done based on the knowledge about physical features that represent causal relationships between the conditions of the design object and physical phenomena. Finally, the designer analyzes the design object by evaluating it with different design modeling systems. KIEF assists the modeling process for these modeling systems and maintains the consistency among different design object models.

Fig. 7 shows operations of this design process. In this figure, 'Input' and 'Output' represent the input and output of each step. 'Support' shows which knowledge and reasoning facilities are used in each step. 'Operation' is a list of operations that should be done by the designer.

4.2.3. Example

Let us illustrate an example of design performed on KIEF. The example, the development of a laser stereo-lithography method with an improved surface accuracy, is based on a real design case conducted at a research laboratory of the University of Tokyo. They tried several approaches to improving the surface accuracy of laser stereo-lithography. They could succeed to improve its surface accuracy to ± 1.5 μm. The final design solution is described in [23].

4.2.3.1 Construction of an initial metamodel. The designer creates an initial metamodel with the FBS modeler by decomposing functional specifications into detailed subfunctions and by selecting corresponding physical features to the decomposed subfunctions. Physical features can also be suggested by QPAS. Fig. 8 depicts the initial metamodel in this way.

4.2.3.2 Detection of unintended physical phenomena. The upper part of Fig. 8 shows the functional structure, and the lower part represents the structure of the design object and physical phenomena occurring to it. The gray-colored physical phenomenon nodes 'Turbulence', 'Pressure-PressureChange' and 'TemperatureChange' are detected as unintended physical phenomena by KIEF. The designer may find these unintended phenomena disturbing to achieve desired behavior. For example, 'Turbulence' may affect the shape of the hollow and becomes noise for the level of 'Resin' that is a liquid to solidify to form a 'ProductOfRapidPrototyping.'

4.2.3.3 Synthesis and analysis based on design modeling systems. KIEF can help the designer choose an appropriate design modeling system to analyze the design object. For instance, the designer wanted to analyze a phenomenon 'SurfaceTension.' According to knowledge

[2] VisualWorks is a registered trademark of Cincom Systems.

[3] ProEngineer is a trademark of Parametric Technology Corporation.

[4] Mathematica is a trademark of Wolfram Research Inc. and is a symbolic processing mathematics system.

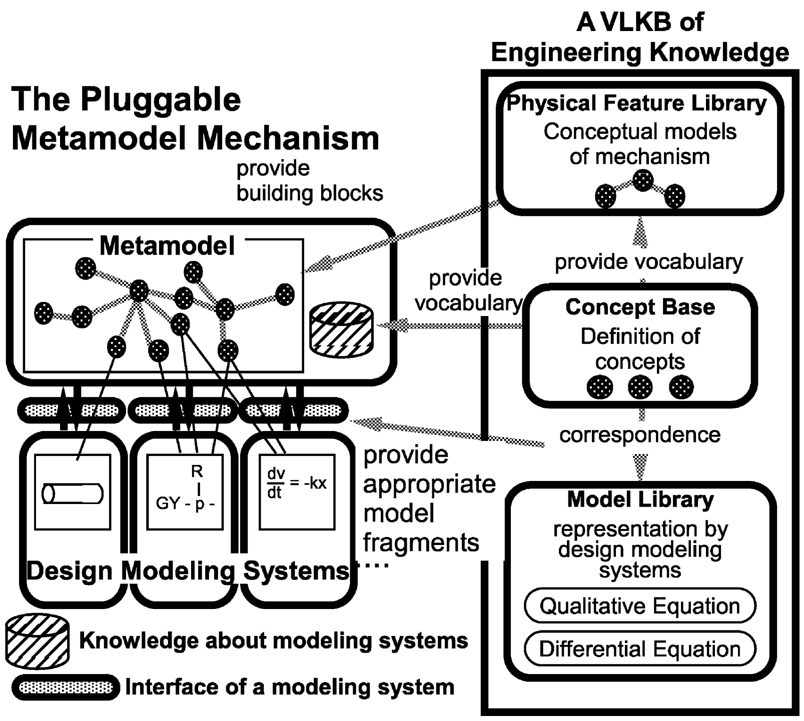
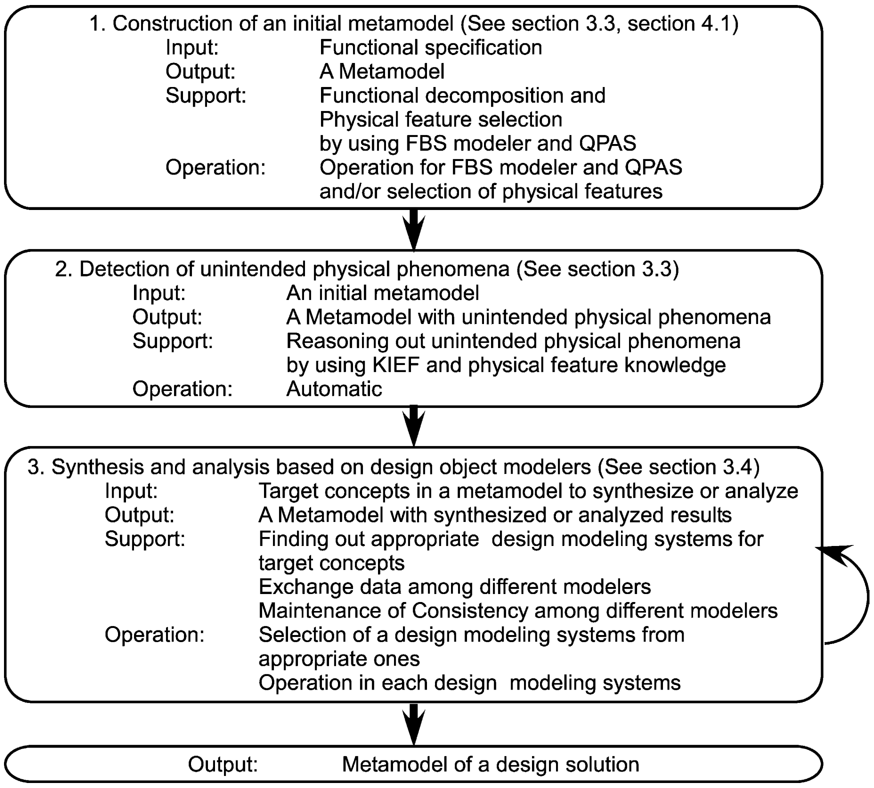

A VLKB of Engineering Knowledge
The Pluggable Metamodel Mechanism
Physical Feature Library
Conceptual models of mechanism
provide building blocks
Metamodel
provide vocabulary
provide vocabulary
Concept Base
Definition of concepts
correspondence
provide appropriate model fragments
Model Library
representation by design modeling systems
Qualitative Equation
Differential Equation
R
GY - p -
dv/dt = -kx
Design Modeling Systems ...
Knowledge about modeling systems
Interface of a modeling system

Fig. 6. Knowledge intensive engineering framework.

about modeling systems, KIEF suggested that a surface tension analyzer could handle 'SurfaceTension.' The model in Fig. 9 shows the aspect model that only uses concepts described in available concepts of the surface tension analyzer. The pluggable metamodel mechanism uses this aspect model to collect attribute information that is necessary to construct a model for the surface tension analyzer. KIEF selects a solid modeler to obtain shape


1. Construction of an initial metamodel (See section 3.3, section 4.1)
Input: Functional specification
Output: A Metamodel
Support: Functional decomposition and Physical feature selection by using FBS modeler and QPAS
Operation: Operation for FBS modeler and QPAS and/or selection of physical features

2. Detection of unintended physical phenomena (See section 3.3)
Input: An initial metamodel
Output: A Metamodel with unintended physical phenomena
Support: Reasoning out unintended physical phenomena by using KIEF and physical feature knowledge
Operation: Automatic

3. Synthesis and analysis based on design object modelers (See section 3.4)
Input: Target concepts in a metamodel to synthesize or analyze
Output: A Metamodel with synthesized or analyzed results
Support: Finding out appropriate design modeling systems for target concepts
Exchange data among different modelers
Maintenance of Consistency among different modelers
Operation: Selection of a design modeling systems from appropriate ones
Operation in each design modeling systems

Output: Metamodel of a design solution

Fig. 7. Design Process in KIEF.

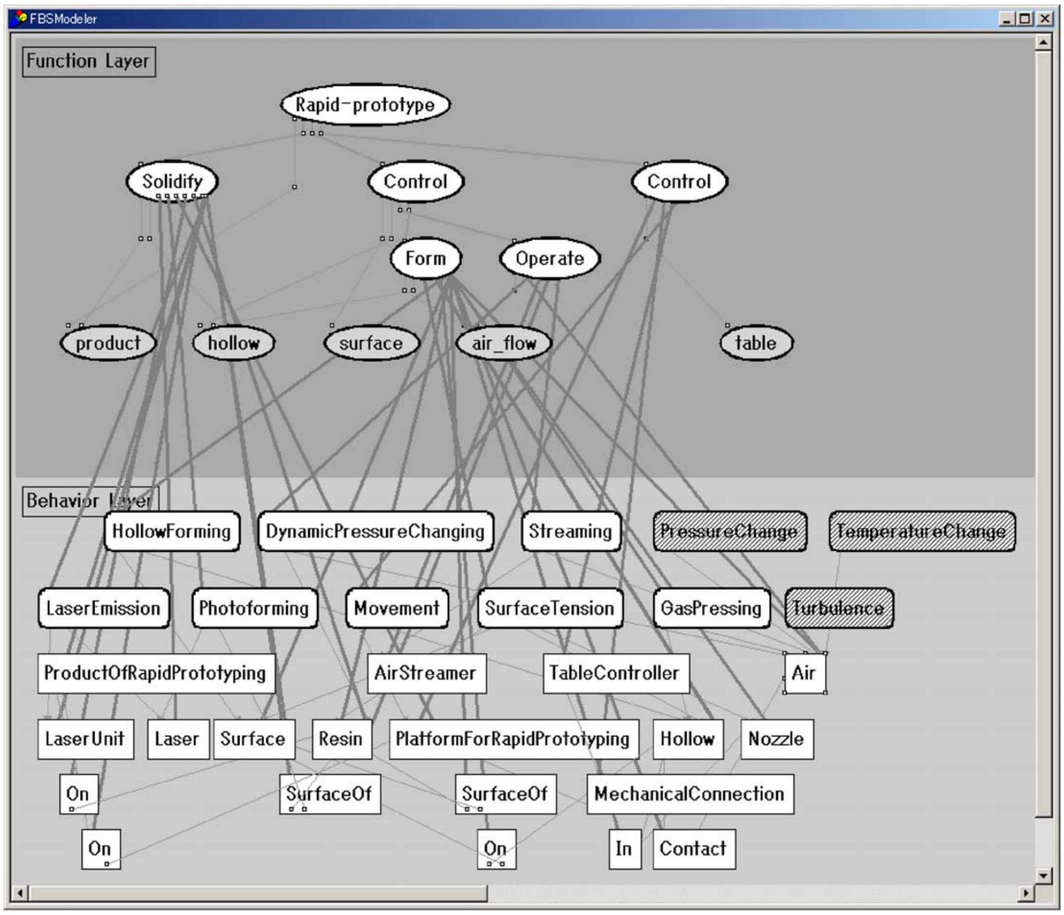


Fig. 8. Result of the FBS modeller.

information. Fig. 10 shows the result of surface tension analysis. Based on this analysis, the designer modifies the 'Radius' of the 'Nozzle'. Since 'Radius' is originally derived from the 'Solid' information in the solid modeler, the pluggable metamodel mechanism finds out that it is necessary to modify 'Solid' information of 'Nozzle' for consistency maintenance. Based on this information, 'Solid' information is modified in the solid modeler

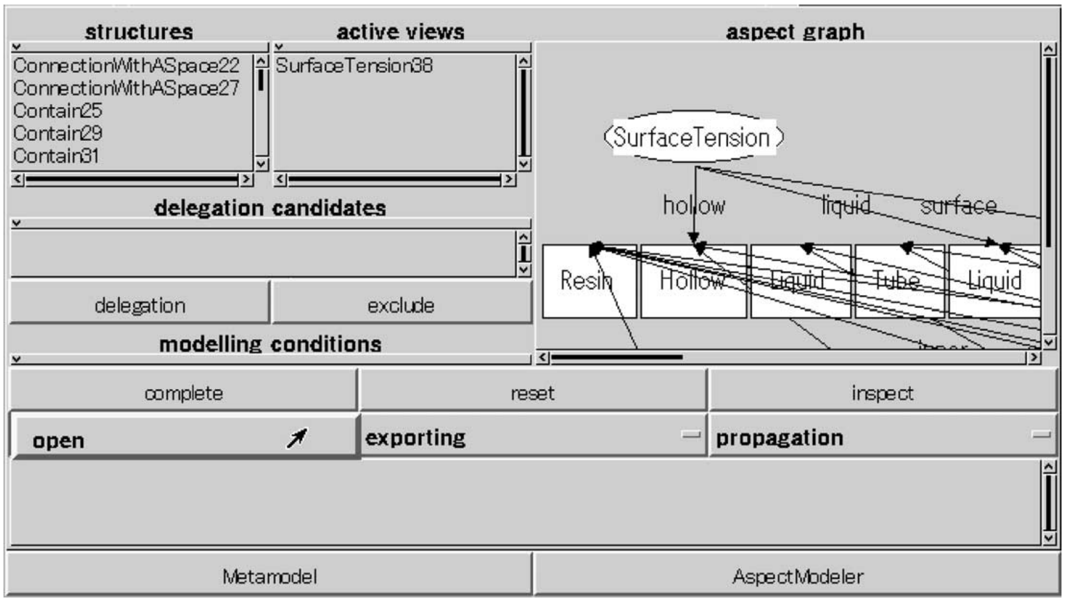


Fig. 9. An aspect model for surface tension analysis.

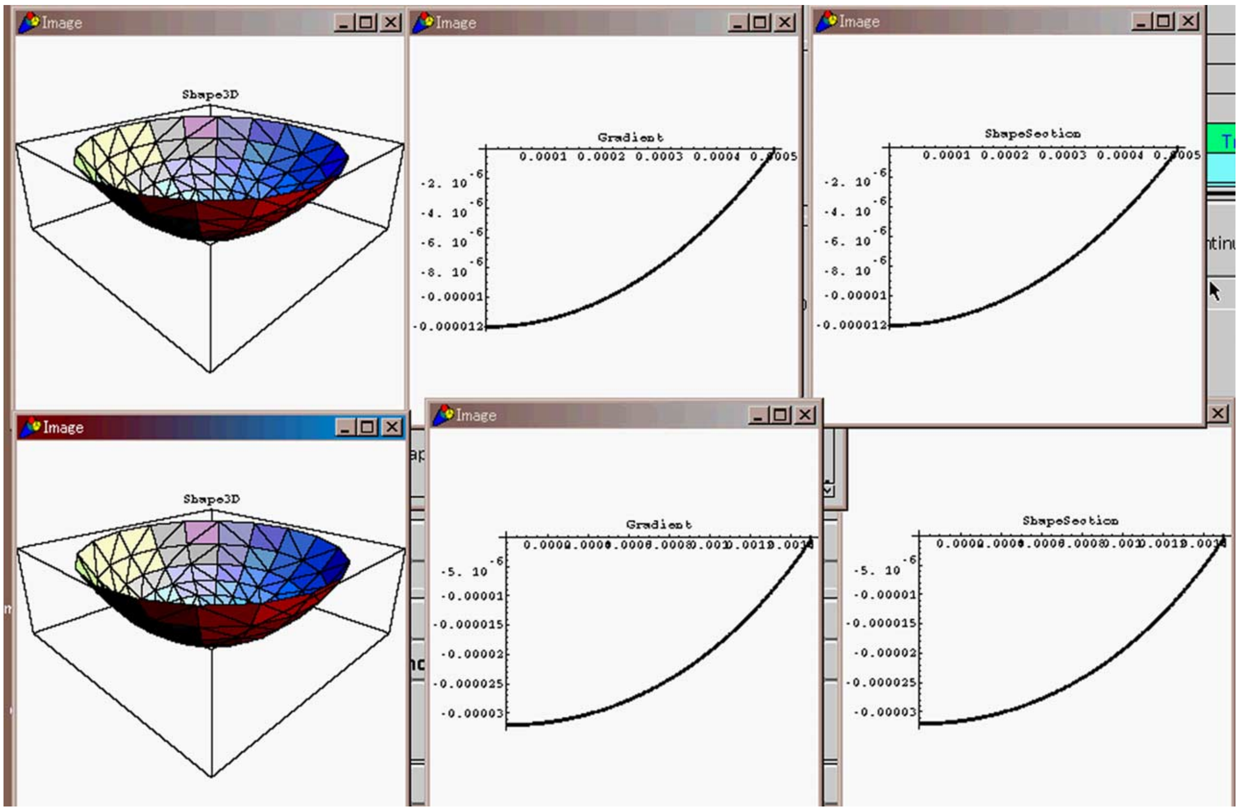


Fig. 10. Results of the surface tension analyzer.

(Fig. 11). This example shows that concept dependency network represented in a metamodel is useful to maintain consistency related to the design object.

We can add a new design modeling system to KIEF and integrate it with other design modeling systems by increasing knowledge about modeling systems. For example, knowledge about the catalog retrieval system (Fig. 12) is described as a reasoning system that can compute detailed attribute information (shape, material, and so on) from an entity concept. The pluggable metamodel mechanism can easily attach this system to KIEF and supports the designer to use it appropriately in an integrated manner.

4.3. Discussion

Through this example, we demonstrated the advantages of having ontological knowledge structure, which are listed below.

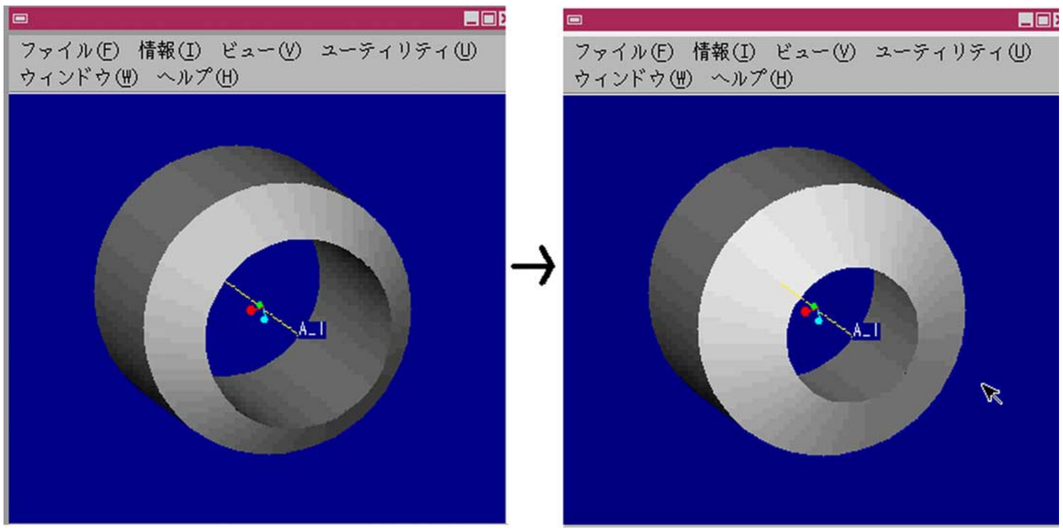
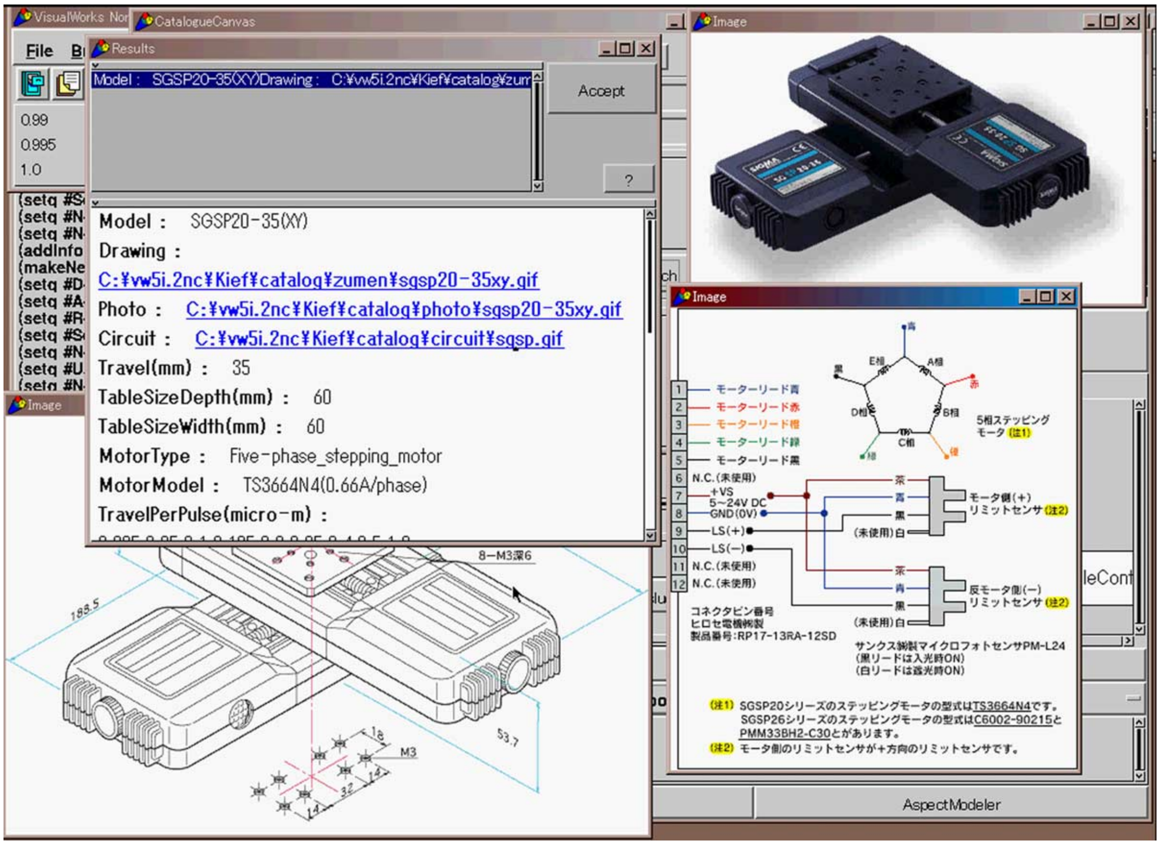


Fig. 11. Modification of “solid” information based on surface tension analysis.


Model : SGSP20-35(XY)Drawing : C:¥vw5i.2nc¥Kief¥catalog¥zum

Model : SGSP20-35(XY)
Drawing :
C:¥vw5i.2nc¥Kief¥catalog¥zumen¥sgsp20-35xy.gif
Photo : C:¥vw5i.2nc¥Kief¥catalog¥photo¥sgsp20-35xy.gif
Circuit : C:¥vw5i.2nc¥Kief¥catalog¥circuit¥sgsp.gif
Travel(mm) : 35
TableSizeDepth(mm) : 60
TableSizeWidth(mm) : 60
MotorType : Five-phase_stepping_motor
MotorModel : TS3664N4(0.66A/phase)
TravelPerPulse(micro-m) :

Fig. 12. Results of the catalog retrieval system.

(1) *Multiple ontology*: Based on the layered engineering knowledge structure, we can flexibly and appropriately define different types of knowledge in an integrated manner. For example, physical feature knowledge represents general causality knowledge, and design modeling systems embeds domain specific knowledge.

(2) *Multiple model management*: Through the pluggable metamodel mechanism based on multiple ontology, we can integrate different design object models only by defining relationships between the basic ontology and each modeling system. Therefore, we do not need to describe one to one correspondence between these modeling systems.

(3) *Conceptual knowledge level integration*: KIEF uses a metamodel that represents the relationships among concepts used in different design modeling systems, to integrate these design modeling systems at conceptual knowledge level. The conceptual knowledge level integration facilitates, for example, flexible use of design modeling systems connected to the pluggable metamodel mechanism through qualitative physics based reasoning about physical features physical phenomena. This facilitates flexible data level integration. KIEF can also maintain the consistency among various design object models.

Construction of the concept base is not an easy task, though. The quality of knowledge in KIEF depends on how much we spend to collect it. Without preparation of good quality knowledge for a particular design problem, the user of KIEF may face the following problems.

(1) KIEF fails to find out unintended physical phenomena when there is no physical feature to check them.
(2) Since KIEF only uses qualitative information (topological structure) to check occurrence of physical phenomena, it may generate many minor or even unfeasible phenomena that can be neglected according to task specific knowledge. For example, physical phenomenon ‘HeatGeneration’ occurs on all ‘ElectorElectoricalWire’ with physical phenomenon ‘ElectriElectricalFlow,’ and physical phenomenon ‘HeatTransfer’ also occur with other entities that connect with ‘ElectoricalWire’. In such cases, it becomes difficult to filter out unintended physical phenomena that seriously affect the desired behavior. This could be solved by introducing other types of knowledge. For example, we need knowledge about

task domains; e.g. a list of attributes that are required to be focused on.

(3) KIEF does not support mapping between different design modeling systems automatically (yet), but this can be solved by implementing knowledge and interfaces for the design modeling systems.

5. Related works

Several attempts have been previously made to represent a physical system with a single ontology. Chandrasekaran et al., [24] proposed a device-oriented ontology to represent the function and behavior of a device. Sasajima et al. [25] proposed Function and Behavior Representation Language (FBRL), which is another device ontology for plant diagnosis. These ontologies are good for representing knowledge in particular domains, but they are not enough to represent all types of engineering knowledge.

PACT [26] and Collaborative Device Modeling Environment (CDME) [27] proposed another framework for integrating existing external modeling systems by using Knowledge Interchange Format (KIF) [28], Ontolingua [29] and Compositional Modeling Language (CML) [30]. They aim to integrate models through translating their knowledge into single format.

Our ontology, in contrast, employs multiple ontology approaches for representing different domain theories represented in different design modeling systems. Since each domain theory is represented in a suitable ontology, KIEF has better expressiveness of knowledge. However, for instance, computational results from each design modeling system can only be compared and integrated within a common ontology that is similar to the one used in a single ontology approach. A single ontology just to represent design objects with a unified vocabulary is insufficient, though, because this common ontology additionally needs to allow us to reason out and maintain the relationship among different domain theories, including such operations as reasoning out unintended physical phenomena, consistency management, and selection of appropriate design modeling systems. These reasoning capabilities are necessary to combine different domain theories flexibly.

Also, PHYSSYS ontology [31] is a formal ontology based upon system dynamics theory and it also aims to construct engineering ontology for integrating reusable engineering model components. They propose to relate different domain theories using an ‘ontology projection’. The ‘ontology projection’ approach is similar to ours that collects a model library based on the physical concept ontology. It works well for a well-structured domain area. However, as we discussed in Section 3.4, it is very difficult to represent all engineering knowledge to concepts in such a formal ontology based on system dynamics theory. In addition, they only discuss translation of a general ontology model to a detailed ontology model and they do not discuss consistency management among models from different domains.

There are several research efforts to implement an integrated design environment. An example of single ontology based systems can be found in [32], which is based on Bond-graph and later resulted in an integrated systems development environment called 20-sim[5]. The system is well functioning, as long as Bond-graph is a good fundamental theory for the design domain. Designer’s workbench [33] and DIICAD-Entwurf [34] are another examples of this type of researches. They try to integrate design models from functional model to its structure. These systems can describe relationship between functional description and geometrical elements, but they do not have explicit physical concept ontology or offer knowledge level integration.

6. Conclusions

In this paper, we proposed a physical concept ontology to integrate various types of engineering knowledge. First, our approach allows multiple ontology with layered ontology structure. At the core of these multiple ontology, there is a common ontology based on physical concepts that facilitates conceptual knowledge level integration and provides vocabulary of engineering knowledge. The knowledge covered are knowledge on design objects themselves, physical phenomena and physical laws, concepts that can be dealt with by various design modeling systems, and knowledge about how to use these systems.

With this ontological representation, we can represent and define a design object as a metamodel. Combined with knowledge about design modeling systems, we can generate an aspect model at physical concept level from the metamodel and further convert it into a model for an individual design modeling system through the pluggable metamodel mechanism. Data is transferred from one design modeling system to another via conceptual knowledge level conversion. This means that the pluggable metamodel mechanism achieves multiple model management and conceptual knowledge level integration as well as data level integration. We can represent knowledge seamlessly from abstract and common level to domainand modeling system-specific level. Ontology facilitates clearer representations and modeling of knowledge.

However, the cost of collecting knowledge could still be too expensive in comparison with benefits obtained from such sophisticated integration. Therefore on one hand, it is important to generate the maximum added value from accumulated knowledge. In our approach, knowledge is used not only for representation but also for reasoning at

[5] http://www.20-sim.com

conceptual level. On the other hand, knowledge accumulation efforts should be made easier with computational supports. We have already proposed the concept of knowledge acquisition by design [7] and developed a system for automatic design documentation and knowledge acquisition [35].

This paper also illustrated Knowledge Intensive Engineering Framework (KIEF) that uses the proposed physical concept ontology and embeds the concept base and the pluggable metamodel mechanisms with a number of design modeling systems. We also demonstrated the power of our approach by illustrating an example design on KIEF.

Acknowledgements

We would like to thank the staff and students of our research group, especially Mr. Masaki Ishii for his contribution to the physical concept ontology design and QPAS development. The example used in Section 4.2 was a result of the 'Modeling of Synthesis' project supported by the Japan Society for Promotion of Sciences (RFTF 9600701) conducted at the University of Tokyo, Osaka University, Nara Institute of Science and Technology, and National Institute of Informatics, Japan, between 1996 and 2000. In particular, we would like to thank Prof Tamotsu Murakami, Dr Takayuki Sekiya, Dr Akira Tsumaya and Dr Souichi Wakuri for their help.

Appendix A. Definitions of concepts in the concept base

Reserved words and letters are shown by double quotations. Examples of these concepts are illustrated in Section 3.2.

Entity

Name: <**entityNamePredicate**>(**?e**)
<entityNamePredicate>::=a symbol which is unique in the category
Supers: <**superEntityPredicate**>(**?e**) <**superEntity**>(**?e**)...
<superEntity>::=a symbol define in the other frame of Entity

Relation

Name: <**relationNamePredicate**>(**?r**)
<relationNamePredicate>::=a symbol which is unique in the category
Supers: <**superRelationPredicate**>(**?r**) <**superRelationPredicate**>(**?r**)...
<superRelationPredicate>::=a symbol define in the other frame of Relation
HasRelations: <**relation**>, <**relation**>...
<relation>::=HasRelation(?r, <localEntityName> <localEntityName>...)

Attribute

Name: <**attributeNamePredicate**>(**?a**)
<attributeNamePredicate>::=a symbol which is unique in the category
Supers: <**superAttributePredicate**>(**?a**), <**superAttributePredicate**>(**?a**)...
<superAttributePredicate>::=a symbol define in the other frame of Attribute.
Statements: **DifferentialOf** (**?a**, **#<attributeNameDifferentialOfPredicate>**, *attributeNameDifferentialByPredicate*)...,
DimensionOf(?a, #(#L #M #T #I #Temp), #(<integer> <integer> <integer> <integer> <integer>))

Physical phenomenon

Name: <**phenomenonNamePredicate**>(?p)
<phenomenonNamePredicate>:=a symbol which is unique in the category
Supers: <**superPhenomenonPredicate**>(**?p**), <**superPhenomenonPredicate**>(**?r**)...
<**superPhenomenonPredicate**>::=a symbol define in the other frame of PhysicalPhenomenon.
Attributes: <**attributePredicate**>(<**attributeTerm**$_1$>), <**attributePredicate**>(<**attributeTerm$_2$**>)...
<attributePredicate>::=a symbol define in the frame of Attribute. <attributeTerm$_i$>::=a symbol which is unique in this frame.
Entities: <**entityPredicate**>(<**entityTerm**$_1$>), <entityPredicate>(<entityTerm$_2$>)...
<entityPredicate>::=a symbol define in the frame of Entity. <entityTerm$_i$>::=a symbol which is unique in this frame.
PhysicalLaws: <**physicalRulePredicate**$_i$> (<**attributeTerm**$_i$>...) <**physicalRulePredicate**> (<**attributeTerm**$_j$>...)...
<physicalRulePredicate>::=a symbol define in the frame of PhysicalLaw.
Statements: **OccurTo** (**?p**, <**entityTerm**$_1$>)..., **HasAttribute**(<**attributeTerm**$_i$>, <**entityTerm**$_j$>)...
<attributeTerm$_{i,j}$>::=a symbol for attribute define in this frame.

Physical law

Name: <**lawName**>
<lawName>::=a symbol which is unique in the knowledge base (usually the first letter is in capitals)
Comments: comment about the knowledge.
Attributes: <**attributeSet**> <**attributeSet**>
...
<attributeSet>::=<localAttributeName> <arrow><AttributeName>

<arrow> : : = "-"

<localAttributeName> : : = a symbol for an instance of an attribute used locally in this frame.

Expression: Mathematical expression of the phenomenon. <localAttributeName> is used for describe expression.

References

[1] ANSI/US PRO/IPO 100. Initial graphics exchange specification IGES 5.3; 1996.

[2] Kimura F. Product and process modelling as a kernel for virtual manufacturing environment. Annals CIRP 1993;42(1):147–50.

[3] ISO TC184/SC4, ISO 10303-1. Industrial automation systems and integration—product data representation and exchange. Part1: overview and fundamental principles; 1994.

[4] IBM corporation, CATIA solutions homepage. http://www-3.ibm.com/solutions/engineering/escatia.nsf/Public/catiaoverview.

[5] Parametric technology corporation, Pro/engineer solution. http://www.ptc.com/products/proe/index.htm.

[6] Tomiyama T. From general design theory to knowledge-intensive engineering. Artif Intell Eng Des, Anal Manufact (AIEDAM) 1994; 8(4):319–33.

[7] Sekiya T, Tsumaya A, Tomiyama T. In: Finger S, Tomiyama T, Mäntylä M, editors. Classification of knowledge for generating engineering models. Knowledge Intens Comput Aided Des. Dordretch: Kluwer; 1999. p. 73–90.

[8] Nomaguchi Y, Tomiyama T, Yoshioka M. Document-based design process knowledge management for knowledge intensive engineering. In: Cugini U, Wozny M, editors. Proceedings of the fourth IFIP working group 5.2 workshop on knowledge intensive CAD, 2000. p. 163–85.

[9] Yoshioka M, Sekiya T, Tomiyama T. An integrated design object modelling environment-pluggable metamodel mechanism-. Turkish J Elect Eng Comput 2001;9(1):43–62.

[10] Tomiyama T, Kiriyama T, Takeda H, Xue D. Metamodel: a key to intelligent CAD systems. Res Eng Des 1989;1(1):19–34.

[11] Ishii M, Sekiya T, Tomiyama T. A very large-scale knowledge base for the knowledge intensive engineering framework, in: KB & KS'95, The second international conference on building and sharing of very large-scale knowledge bases; 1995, p. 123–31.

[12] Kiriyama T, Tomiyama T, Yoshikawa H. The use of qualitative physics for integrated design object modeling. Design theory and methodology (DTM'91). New York: The American Society of Mechanical Engineers (ASME); 1991 [p. 53–60].

[13] Umeda Y, Ishii M, Yoshioka M, Tomiyama T. Supporting conceptual design based on the function-behavior-state modeler. Artif Intell Eng Des, Anal Manufact (AIEDAM) 1996;10(4):275–88.

[14] Shah JJ, Mäntylä M. Parametric and feature-based CAD/CAM. NY, USA: Wiley; 1995.

[15] Forbus K. Qualitative process theory. Artif Intell 1984;24(3):85–168.

[16] Hix C, Alley R. Physical laws and effects. London: Wiley; 1958.

[17] Yoshikawa H, Gossard D (Eds.). Intelligent CAD, I, North-Holland, Amsterdam; 1989.

[18] Coyne R. Logic models of design. London: Pitman Publishing; 1988.

[19] Takeda H, Veerkamp PJ, Tomiyama T, Yoshikawa H. Modeling design processes. AI Mag 1990;11(4):37–48.

[20] Ishii M, Tomiyama T, Yoshikawa H. A synthetic reasoning method for conceptual design. In: Wozny M, Olling G, editors. Towards world class manufacturing, IFIP transactions B-17, North-Holland, Amsterdam, 1994. p. 3–16.

[21] Ishii M, Tomiyama T. A synthetic reasoning method based on a physical phenomenon knowledge base. In: Sharpe J, editor. Proceedings of the 1995 Lancaster international workshop on engineering design, 1996. p. 109–23.

[22] de Kleer J, Brown JS. A qualitative physics based on confluences. Artif Intell 1984;24(1):7–83.

[23] Xie T, Murakami T, Nakajima N. Micro photoforming fabrication using a liquid hollow shaped by pressure difference and surface tension. Int J Jpn Soc Precision Eng 1999;33(3):253–8.

[24] Chandrasekaran B, Josephson JR. Function in device representation. Eng Comput 2000;16:162–77.

[25] Sasajima M, Kitamura Y, Ikeda M, Mizoguchi R. A representation language for behavior and function: Fbrl. J Expert Syst Appl 1996; 10(3/4):471–9.

[26] Cutkosky MR, Engelmore RS, Fikes RE, Genesereth MR, Gruber TR, Mark WS, Tenenbaum JM, Weber JC. PACT: An experiment in integrating concurrent engineering systems. IEEE Comput 1993; 26(1):28–37.

[27] Iwasaki Y, Farquhar A, Fikes R, Rice J. A web-based compositional modelng system for sharing of physical knowledge. In: Fifteenth international joint conference on aritficial intelligence, Nagoya, Japan, 1997; p. 450–94.

[28] Genesereth M. . In: Allen J, Fikes R, Sandwall E, editors. Knowledge interchange format. Proceedings of the conference of the principles of knowledge representation and reasoning. Los Altos: Morgan Kaufmann Publishers; 1992. p. 599–600.

[29] Gruber TR. Ontolingua: a mechanism to support portable ontlogies. Technical report KSL91-66, Knowledge Systems Laboratory, Stanford University, Stanford; 1992.

[30] Falkenhainer B, Farquhar A, Bobrow D, Fikes R, Forbus K, Gruber T, Iwasaki Y, Kuipers B. CML: A compositional modeling language, Technical report KSL94-16, Knowledge Systems Laboratory, Stanford University, Stanford; 1994.

[31] Borst P, Akkermans H. Engineering ontologies. Int J Human–Comput Stud 1997;46(2/3):365–406.

[32] Broenink JF, Kleijn C. In: Roberts CTGN, editor. Computer-aided design of mechatronic systems using 20-sim 3.0. WESIC 99, Second workshop on European scientific and industrial collaboration, 1999. p. 27–34.

[33] Andreasen MM. The role of artefact theories in design. In: Universal Design Theory, Shaker, Aachen, 1998; p. 57–70.

[34] Grabowski H, Lossack R. In: Finger S, Mäntylä M, Tomiyama T, editors. Knowledge based design of complex products by the concept of design working spaces. IFIP WG 5.2 workshop knowledge intensive CAD-2 proceedings, 1996. p. 79–98.

[35] Nomaguchi Y, Tomiyama T, Yoshioka M. In: Cugini U, Wozny M, editors. Document-based design process knowledge management for knowledge intensive engineering. From knowledge intensive CAD to knowledge intensive engineering. Dordrecht: Kluwer; 2001. p. 131–44.