Knowledge Intensive Engineering Framework: KIEF (formerly known as SYSFUND) Manual

Tomiyama Lab.
The University of Tokyo
Masaharu YOSHIOKA
National Center for Science Information systems

January 12, 2000

Contents

Chapter 1

Introduction

1.1 What is the KIEF?

KIEF (Knowledge Intensive Engineering Framework) is a tool for supporting various engineering activity related to the whole life cycle of artifacts; e.g. design, maintenance, and so on. In this framework, various kind of engineering tools can be integrated by intensively storing engineering knowledge and representing the relationship among them.

The KIEF has two major roles. It works as a knowledge base, and it manages various computational engineering tools by using pluggable metamodel mechanism that allows to plug in existing design object modelers. In this framework, all design object modelers in SYSFUND (SYStematization tool of FUNctional knowledge for Design) (e.g. Function Behavior State (FBS) modeler which is a CAD for conceptual design, qualitative physics based reasoning system) are already plugged in.

As a knowledge base, the KIEF accumulates *concept dictionary* which is a basic definition of concepts for knowledge bases represented in various engineering tools for management of the relationship among knowledge bases. In addition, since the KIEF includes all design object modelers in SYSFUND (e.g. Physical Reasoner based on Qualitative Process Theory [Forbus84], Function Behavior State (FBS) modeler [Umeda96]), knowledge bases for these modelers accumulates functional knowledge about machines, knowledge about behavior and structure and so on.

As a CAD for conceptual design, the designer uses FBS modeler for functional design. As a result, a function hierarchy and a causal dependency network which depicts the basic mechanism, including a topological structure, of the design object will be constructed. After that, the KIEF system reasons out the possible physical phenomena occurred on the designed object. From this description of the designed object, the designer can make various design object model (e.g., Physical Reasoner base on Qualitative Process Theory, Physical Reasoner based on mathematical model, and so on), and evaluate the designed object for verifying its function.

1.2 Architecture

The architecture of the KIEF is shown as Figure 1.1.

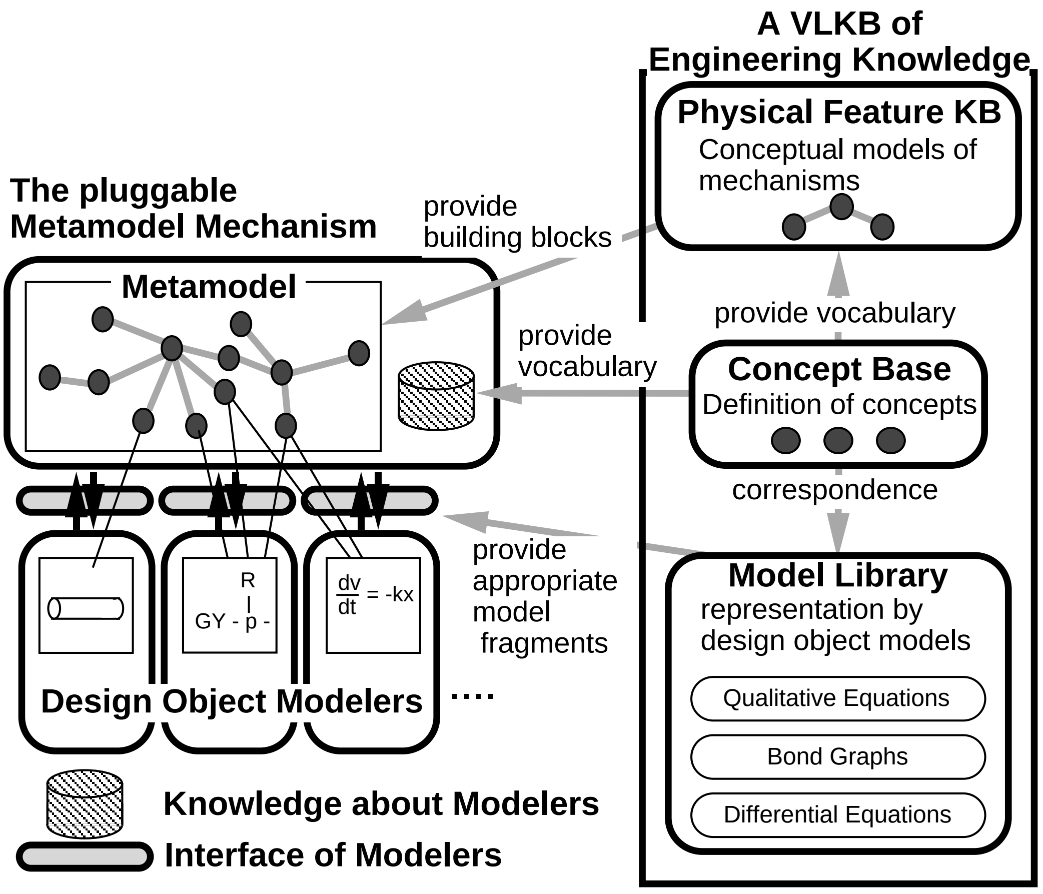


Figure 1.1: System architecture of the KIEF

Right side of Figure 1.1 depicts the three component architecture of the knowledge base system for the KIEF. The middle component, called *concept dictionary*, contains physical concepts.

In the concept dictionary, physical concepts are categorized into the following six types.

- Entity
  An entity represents an atomic physical object. Entities include such as mechanical parts and electric devices, and are organized in an abstract-concrete hierarchy. For example, a “worm gear” is a subclass of a “gear.” The hierarchy allows multiple inheritance.

- Relation
  Relations represent relations among entities to denote static structure. They include relations between physical objects such as “connection” and “on.” They are also organized in an abstract-concrete hierarchy.

- Attribute
  An attribute is a concept attached to an entity and takes a value to indicate the state of the entity, such as “position” and “temperature.” Attributes also have a description about differential relationships with other attributes (e.g. “velocity” is a differential of “position”).

- Physical rule
  A physical rule represents relationships among attributes such as “Kirchhoff low.” A physical rule represents relationships among attributes.

- Physical phenomenon
  A physical phenomenon designates physical laws or rules that govern behaviors. A physical phenomenon is defined by the following slots.

  – Name of the phenomenon.
  – Super (or abstract) physical phenomena as described in Entity.
  – Related physical phenomena, entities, and attributes with respect to the phenomenon.
  – Physical rules govern the phenomenon.

Figure 1.2 is an example description about gear transmission. The upper layer contains a *Physical Feature*, such as a worm gear pair, which represents a combination of a set of entities and relations among the entities, and physical phenomena causally related to the entities. Physical features are used as building blocks for a physical model on this system. Physical concepts in the concept dictionary provide a vocabulary to build the physical feature.

Model fragments for building a model with various model representation are stored in the *model library* with the relationship to the concept stored in the concept dictionary.

Left side of Figure 1.1 depicts the *pluggable metamodel mechanism* which has a capability to use existing design object modelers (including commercial CAE tool such as solid modeler, FEM and so on). The pluggable metamodel system maintains the consistency by using *metamodel* that is a model which represents the relationships among concepts used in various design object models.

In the pluggable metamodel mechanism, a metamodel is constructed by using physical features as building blocks. For plugging-in new design object modelers, *Knowledge about Modelers* should be defined by using the vocabulary in the concept dictionary. Model library is used as model fragments while exporting the information in metamodel to the design object modelers through *Interface of Modelers*.

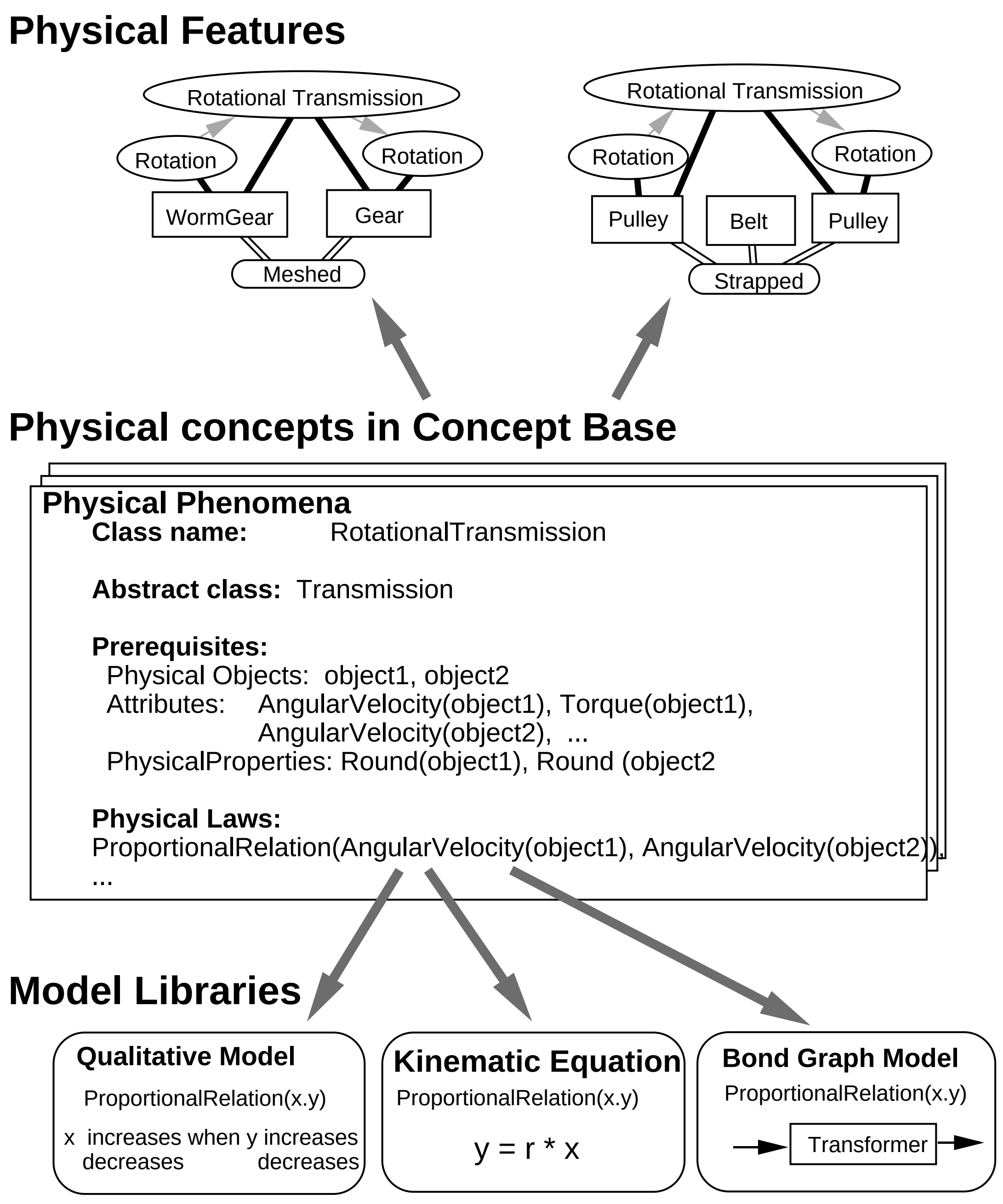

Physical Features
Rotational Transmission
Rotation
Rotation
WormGear
Gear
Meshed
Rotational Transmission
Rotation
Rotation
Pulley
Belt
Pulley
Strapped
Physical concepts in Concept Base
Physical Phenomena
Class name: RotationalTransmission
Abstract class: Transmission
Prerequisites:
Physical Objects: object1, object2
Attributes: AngularVelocity(object1), Torque(object1), AngularVelocity(object2), ...
PhysicalProperties: Round(object1), Round (object2
Physical Laws:
ProportionalRelation(AngularVelocity(object1), AngularVelocity(object2)),
...
Model Libraries
Qualitative Model
ProportionalRelation(x.y)
x increases when y increases
decreases decreases
Kinematic Equation
ProportionalRelation(x.y)
y = r * x
Bond Graph Model
ProportionalRelation(x.y)
Transformer

Figure 1.2: Describing the Knowledge about Gear Transmission in the KIEF

Chapter 2

Knowledge Representation in the KIEF

The KIEF uses different type of knowledge in several ways. There are five types of knowledge in the KIEF system as follows.

1. Concept dictionary
2. Physical feature
3. Knowledge about modelers
4. Model library
5. Knowledge stored in plugged-in design object modelers

At first, we explain the usage of KIEFLauncher that is used for execute knowledge definition tools and reasoning systems. After that, we explain the representation of concept dictionary, physical feature and knowledge about modelers. In addition, we also explain the guideline to make model library. Knowledge stored in plugged-in design object modelers are described in Appendix A.

2.1 KIEFLauncher

KIEFLauncher is a launcher for executing knowledge definition tools and reasoning systems in the KIEF (Figure 2.1). The KIEFLauncher is initially opened when you install a KIEF system. However, if you close this launcher please evaluate "KIEFLauncher open" in Workspace to open new launcher.

The KIEFLauncher consists from two parts. One is buttons for select tool types that are used for changing the buttons at the upper part of the launcher. The other is buttons for start up tools that are used for start up tools at the upper part of the launcher. To start tools in the KIEF, first you select one button from the category buttons and select one button from the tool execution buttons.

In following documents, we describe the usage of the launcher for starting up tools as "Button for select tool type" -> "Button for start up tools."

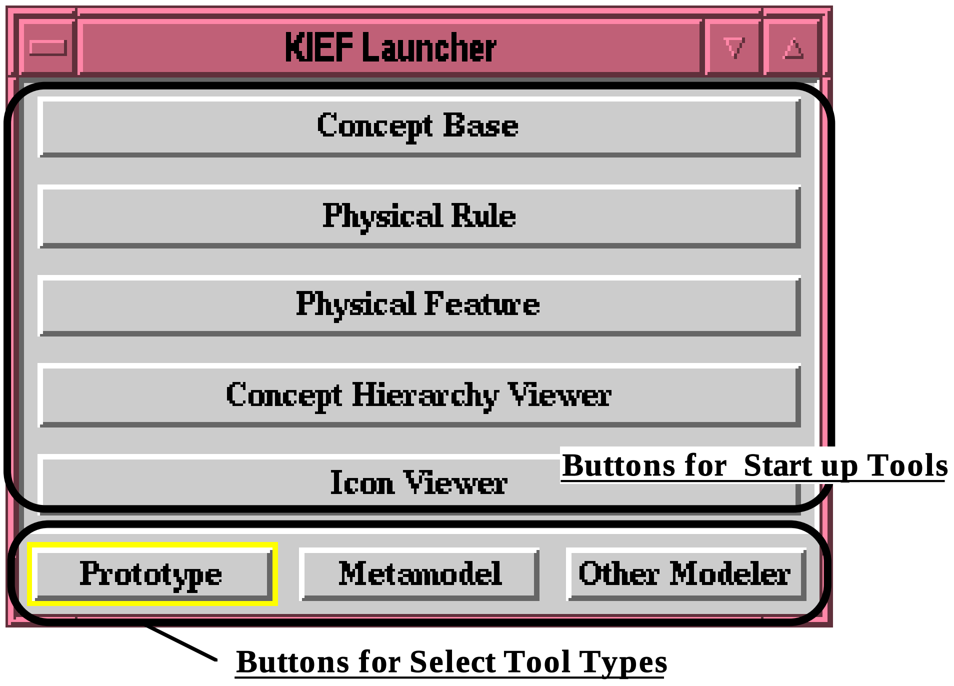


Figure 2.1: The KIEF Launcher

2.2 Concept dictionary

There are following five type of concepts in concept dictionary.

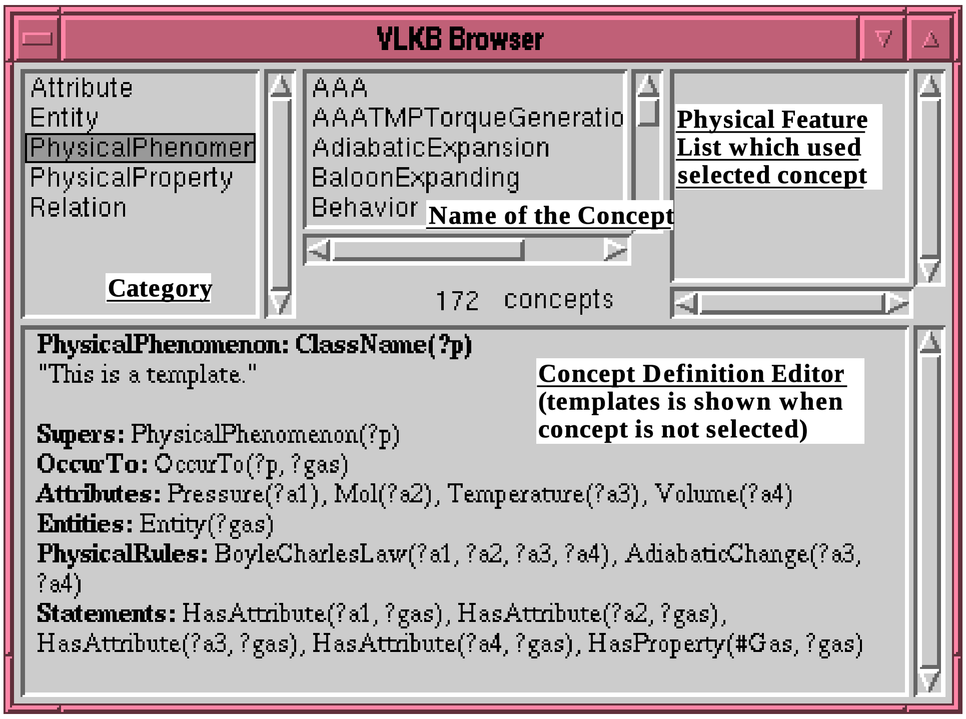
- Entity
- Relation
- Attribute
- Physical phenomenon
- Physical rule

All these concepts except physical rule are defined in *VLKBBrowser* (Figure 2.2) by using predicate logic. To start this VLKBBrowser, use the KIEFLauncher by "Prototype" -> "ConceptBase."

There are three steps to define concepts.

1. Select a category from the "Category" list.
2. Define concepts in "Concept Definition Editor" by using a template. The template is shown by selecting a category and not selecting already defined concept from "Name of the Concept".
3. Accept concepts by using middle button menu "accept."

In this section, we explain the knowledge description about these concepts.



PhysicalPhenomenon: ClassName(?p)
"This is a template."

Supers: PhysicalPhenomenon(?p)
OccurTo: OccurTo(?p, ?gas)
Attributes: Pressure(?a1), Mol(?a2), Temperature(?a3), Volume(?a4)
Entities: Entity(?gas)
PhysicalRules: BoyleCharlesLaw(?a1, ?a2, ?a3, ?a4), AdiabaticChange(?a3, ?a4)
Statements: HasAttribute(?a1, ?gas), HasAttribute(?a2, ?gas), HasAttribute(?a3, ?gas), HasAttribute(?a4, ?gas), HasProperty(#Gas, ?gas)

Figure 2.2: VLKB Browser

2.2.1 Entity

An entity represents an atomic physical object. An entity has 2 slots.

Entity Write the name of this concept of a entity.

Supers Write super entities to make hierarchy.

A frame of an entity (The reserved words and letters are shown by double quotations.)

Entity: **<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

Example

Entity: Gear(?e)

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

2.2.2 Relation

A relation represents relations among entities to denote static structure. A relation has 3 slots.

Relation Write the name of this concept of a relation.

Supers Write super relations to make hierarchy.

HasRelations Define how to relate the entities using a predicate. The order of the entities which is the arguments of the predicate should be considered.

A frame of a relation (The reserved words and letters are shown by double quotations.)

Relation: <**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> ...)

Example

Relation: Meshed(?r)

Supers: Relation(?r)

HasRelations: HasRelation(?r, ?gear1, ?gear2), HasRelation(?r, ?gear2, ?gear1)

2.2.3 Physical Phenomenon

A physical phenomenon designates physical laws or rules that govern behaviors. A physical phenomenon has 3 slots.

PhysicalPhenomenon Write the name of this concept of a physical phenomenon.

Supers Write super physical phenomena to make hierarchy.

Attributes Write related attributes to the phenomenon.

Entities Write related entities to the phenomenon.

PhysicalRules Write physical rules that governs the phenomenon.

Statements Define how to relate the entities and attributes by a predicate. There are 2 types of predicates for describing the relation. The order of the arguments of the predicate should be considered.

1. **OccurTo** relate the phenomenon with the entities.
2. **HasAttribute** relate the attribute with an entity or entities.

A frame of a physical phenomenon (The reserved words and letters are shown by double quotations.)

PhysicalPhenomenon: **<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.

PhysicalRules: **<physicalRulePredicate>(<attributeTerm$_i$ > ...) <physicalRulePredicate>(<attributeTerm$_j$ > ...) ...**
<physicalRulePredicate>::= a symbol define in the frame of PhysicalRule.

Statements:	**OccurTo (?p, <entityTerm$_1$ >) ...,**
	HasAttribute(<attributeTerm$_i$ >, <entityTerm$_j$ >)
	...

<attributeTerm$_{i,j}$ > :: = a symbol for attribute define in this frame.

Example

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

2.2.4 Attribute

An attribute is a concept attached to an entity and takes a value to indicate the state of the entity. Attribute has ? slots.

Attribute Write the name of this concept of an attribute.

Supers Write super attribute to make hierarchy.

ValueType Write data type of the attribute.

AccessingMethod Write name of the method to extract another attribute data from the attribute. These methods are implemented in the interface between metamodel mechanism and external modelers.

A frame of a physical phenomenon (The reserved words and letters are shown by double quotations.)

Attribute:	**<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.

ValueType:

AccessingMethod: **<attributePredicate>(<attributeTerm$_1$ >),**
AccessingMethod(<attributeTerm$_1$ >,<methodNameTerm$_1$ >,)
...

<attributePredicate>::= a symbol define in the frame of Attribute. <attributeTerm$_i$ >::= a symbol which is unique in this frame. <methodNameTerm$_i$ >::= a symbol which is unique in unique in the category.

Example

Attribute:	
Supers:	Motion(?r)
ValueType:	
Entities:	Mass(?object)
PhysicalRules:	SecondLawOfNewtonLaws(?f, ?m, ?acc)
Statements:	OccurTo(?p, ?object), HasAttribute(?f, ?object), HasAttribute(?m, ?object), HasAttribute(?pos, ?object), HasAttribute(?acc, ?object), HasAttribute(?vel, ?object)

2.2.5 Physical Rule

A physical rule represents relationships among attributes. Physical rules are defined in *Physical Rule Browser* (see Figure 2.3). The user can open a Physical Rule Browser with the KIEFLauncher by "Prototype" -> "Physical Rule." A physical rule has 4 slots. The slots "Name" and "Attributes" are necessary.

Name Write the name of this physical rule.

Comments Comments on the physical rule

Attributes Write attributes on which this physical rule constrains.

Expression Write constraint of physical rule by mathematical expression.

A frame of a physical rule

(The reserved words and letters are shown by double quotations.)

Name: **<ruleName>**

<ruleName> ::= 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> ::= "_"
<localEntityName> ::= 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.

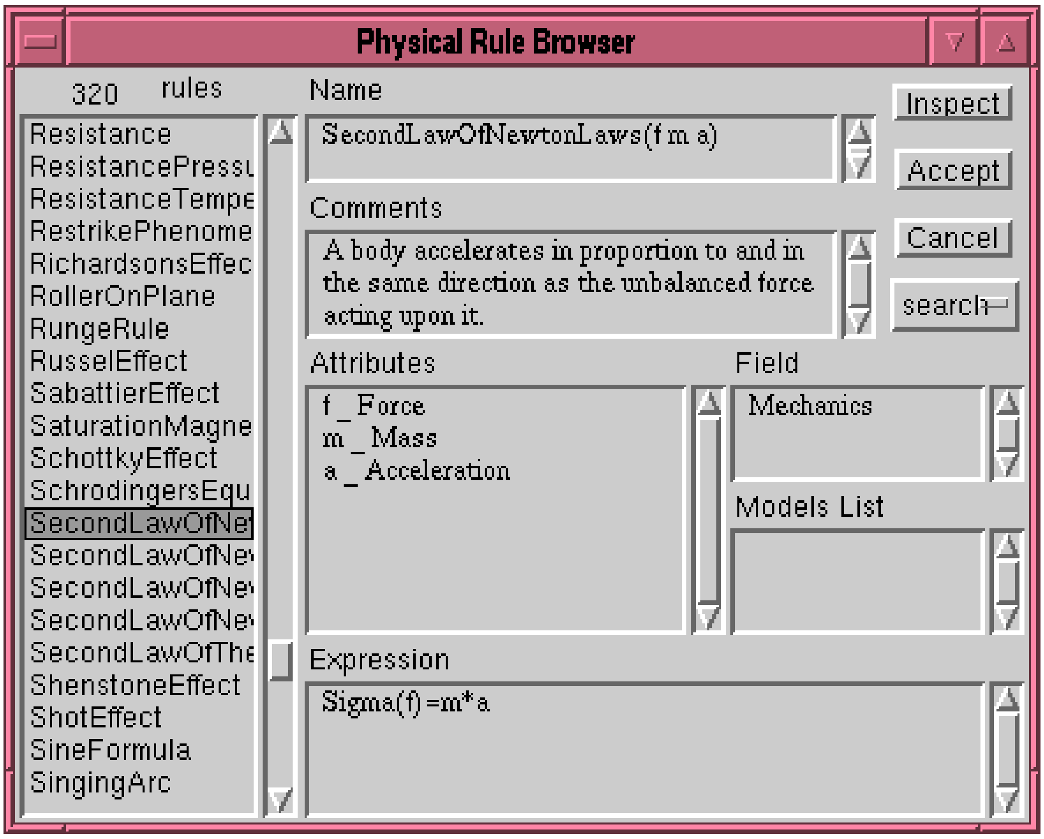
Example

Name: **SecondLawOfNewtonsLaw**

Comments: **KenematicMotion**

Attributes: **f _ Force**
m _ Mass
a _ Acceleration

Expression: Sigma(f)=m*a


Physical Rule Browser

320 rules

Resistance
ResistancePressu
ResistanceTempe
RestrikePhenome
RichardsonsEffec
RollerOnPlane
RungeRule
RusselEffect
SabattierEffect
SaturationMagne
SchottkyEffect
SchrodingersEqu
SecondLawOfNe
SecondLawOfNe
SecondLawOfNe
SecondLawOfNe
SecondLawOfThe
ShenstoneEffect
ShotEffect
SineFormula
SingingArc

Name: SecondLawOfNewtonLaws(f m a)

Comments: A body accelerates in proportion to and in the same direction as the unbalanced force acting upon it.

Attributes:
f _ Force
m _ Mass
a _ Acceleration

Field: Mechanics

Models List

Expression: Sigma(f)=m*a

Figure 2.3: Physical Rule Browser

2.2.6 Construction of Physical Feature

Physical feature is a set of physical phenomena and structures for invoking the phenomena. It is used as a building block of metamodel and as knowledge for checking the occurrence of physical phenomena. Physical features can be constructed using the concepts in a concept dictionary. The KIEF provides the user an editor called *Physical Feature Editor* (Figure 2.4).

1. Instantiate nodes for the physical feature.
   The user instantiate nodes for physical phenomena, entities, relations, attributes for the physical feature by using the command “addPhysicalPhenomenon”, “addEntity” and so on from the “add” submenu.

2. Connect nodes to set the relationship among concepts.
   There are four type of connections.

   - Relation - Entity connection
     Connect relation node and entity node to build up the structure for invoking the phenomena. The user should specify local names from the conceptual definition of relation.
   - Physical Phenomenon - Entity connection
     Connect physical phenomenon node and entity node to specify where the phenomenon occurs. The user should specify local names from the conceptual definition of physical phenomenon.
   - Physical Phenomenon - Physical Phenomenon connection
     Connect physical phenomenon node and physical phenomenon node to represent the causal dependency.
   - Attribute - Physical Phenomenon connection
     Connect attribute node and physical phenomenon node to represent the condition for invoking the phenomenon.

   The user connects the nodes by following manner.

   (a) Select “from node” to connect.
   (b) Select with shift button pressing “to node” to connect.
   (c) Select command “connect nodes.’ from the menu.
   (d) If the user should select local names, the user selects one local name from the selection menu.

3. Set derivation flag for each physical phenomenon.
   Physical Feature is used for checking the occurrence for some of the physical phenomena in the physical feature. Setting a derivation flag of a phenomenon node to true means, the physical feature is used for checking the occurrence of a phenomenon. On the contrary, setting the flag to false means the phenomenon is used as a condition for invoking the physical feature.

Derivation flag can flip by using command “flip derivation flag”. Line width of rectangle represents this flag. Wider line width means true, and narrower line means false.

4. Accept the physical feature.
Push “Accept” button, and set the names by using the dialog.

There are some other commands for constructing the physical feature more flexibly. Please refer to the reference manual.

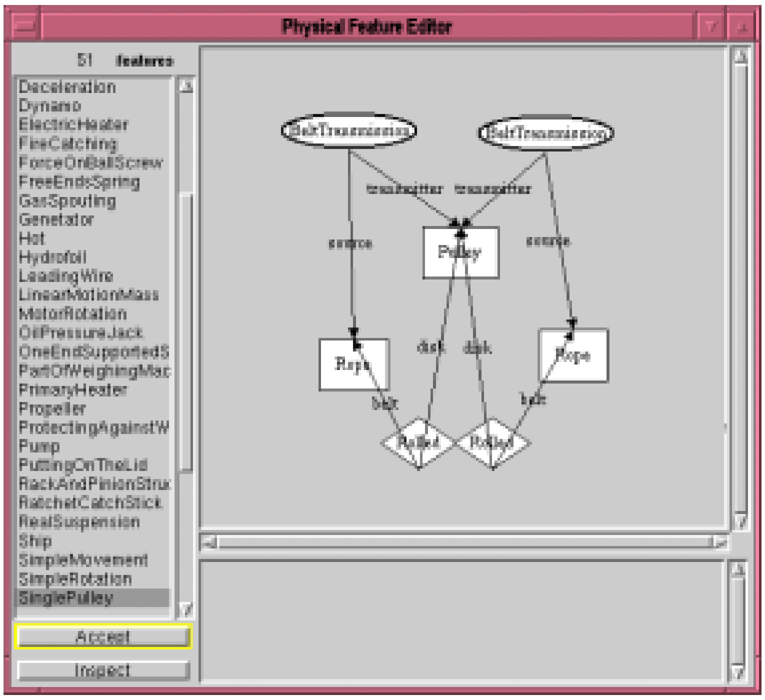


Figure 2.4: Physical Feature Editor

2.3 Knowledge about Modelers

Knowledge about modelers represents characteristic features of the modelers. Knowledge about modelers are defined in **Modeler Definer** (see Figure 2.5). The user can open a Modeler Definer with the KIEFLauncher by “Metamodel” $->$ “Modeler Definer.” Knowledge about modelers has ? slots.

Name Name of the modeler.

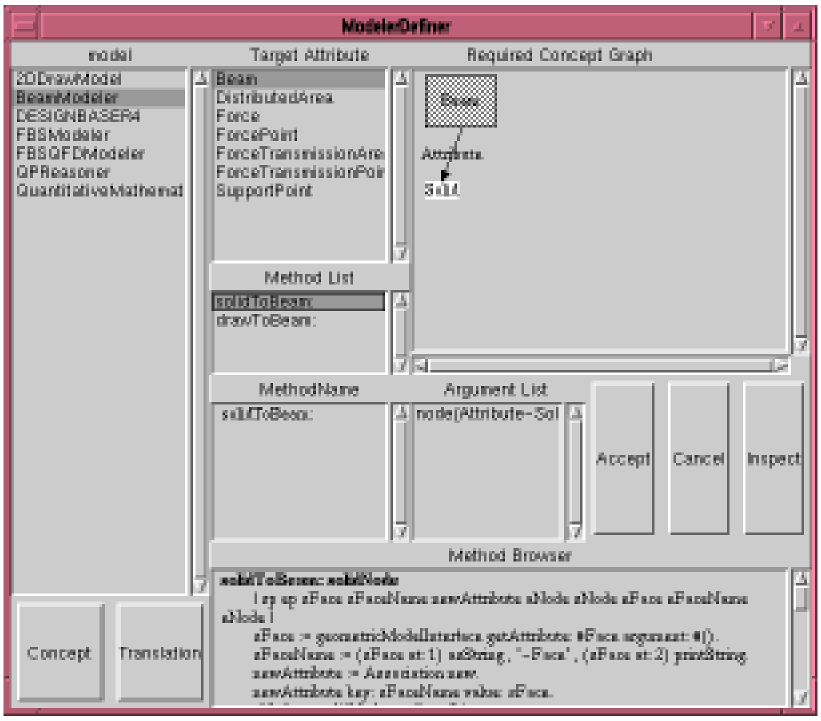
Related Concepts Concepts related to concepts used to build the model in the modeler. These concepts are used to filter out the unrelated concepts from design object model while making initial qualitative model for the modeler.

Usable concepts Concepts used to build the modeler. These concepts are used to abstract the metamodel to a model in the modeler.

Available concepts Concepts computed from the modeler. Available concepts is used to find out the appropriate modeler for computing the value of attribute.

Attribute translation method Method for calculating value of the attributes in the modeler from the value of other attributes in the metamodel mechanism. Required attribute values are represented with the graph that represents relationship between entities, relations, or physical phenomena which the attributes belongs to.

The example of Figure 2.5 shows that attribute solid is required to calculate the attribute of beam.



```
solidToBeam: solidNode
    | sp ap aFace aFaceName newAttribute aNode aNode aFace aFaceName
aNode |
    aFace := geometricModelInterface getAttribute: #Face argument: #().
    aFaceName := (aFace at: 1) asString , '-Face' , (aFace at: 2) printString.
    newAttribute := Association new.
    newAttribute key: aFaceName value: aFace.
```

Figure 2.5: Modeler Knowledge Browser

Chapter 3

Design in the KIEF

Design in the KIEF is conducted by using the pluggable metamodel mechanism. For starting a design from the functional specification, the design process has the following four steps.

1. Constructing FBS model.
   FBS modeler takes required functions as inputs. FBS modeler decomposes the required functions into subfunctions in order to detail it using functional knowledge. As a result, a functional hierarchy is constructed.
   Each function located at the bottom of the function hierarchy should have physical features, which realize the function, or behaviors, which are represented as state transitions, in its function prototype knowledge. The designer selects or creates a physical feature for each bottom functions. For selection of physical features, the designer uses the knowledge in the function prototypes. Instead of selection, the designer can create a new physical feature which can cause the behaviors written in the function prototype using QPAS. After constructing a FBS model, export model to the pluggable metamodel mechanism.

2. Computing a metamodel.
   The metamodel mechanism finds out possible physical phenomena that may occur on the design object with qualitative reasoning.

3. Generating an external model.
   The designer generates an aspect model for evaluation by an external modeler. First, the metamodel mechanism selects related concepts to the aspect and relations among the concepts from the metamodel. By doing so, a conceptual model for generating an external model is constructed. After this, the metamodel mechanism translates the conceptual model to an external model by using model fragments and attribute data represented in the metamodel.

4. Evaluating the external model.
   Finally, the designer evaluates the generated aspect model with the external modeler.

The last 2 steps reiterate for evaluating from various aspects.

3.1 Using Plugged Modelers

First, the designer opens the *Metamodel Interface* by selecting **Metamodel** in the lower part of launcher and selecting **Metamodel** in it. From this interface, plugged modelers can be used. There are following 4 steps to use a plugged modeler.

1. Make aspect modeler.
   The designer selects one modeler name from the available modeler list and selects **make new aspect model** from the menu. Then right part of window changes to aspect modeler for selected modeler.

2. Make aspect model.

   (a) *Filtering related concepts from metamodel*
   The pluggable metamodel mechanism finds out related concepts from a metamodel. For example, consider the design of a robot arm. If the designer want to evaluate the distortion of arm as a beam model, the pluggable metamodel mechanism collects the concept like, force, entity, connection from a metamodel. In other words, non related concepts like electric flow between motor and battery are filtered out.

   (b) *Determine abstraction level for a design object modeler* (Figure 3.1-(a)).
   The metamodel mechanism determines most appropriate abstraction level for the selected modeler based on the knowledge about design object modeler. The metamodel mechanism suggests concepts included in the determined abstraction level and needed for the selected modeler, and the designer has to give abstract descriptions of the design object using these concepts. In the example of making beam model of arm, the metamodel mechanism suggests to describe the arm only using concepts such as beam, load, support, and bending. Then the designer abstracts the arm concept in the domain of geometry as a beam. This abstraction is computationally done by unification that is an operation to create a new instance that delegates the two concepts of shape “arm” and physical feature “beam.”

   (c) *Determine simplification level for the selected modeler* (Figure 3.1-(b)).
   The designer determines appropriate simplification level for the selected modeler. The designer selects physical phenomena that should be considered in the selected aspect model. In other words, some physical phenomena and some entities might be neglected. For instance, the designer can say that any bending is considered while vibration is not of interest in the arm design.

3. Open the plugged modeler.
   By selecting **open** button, the selected plugged modeler opens. After that the designer export information of the metamodel to the plugged modeler

by using **export** menu button. Then data exchange among aspect models are conducted (Figure 3.1-(c)). Suppose in the example of arm design, a bending aspect model is generated. Since aspect models often require numerical information, the metamodel mechanism requests the designer to specify an appropriate aspect model to feed the required numerical information. For instance, since the generated bending aspect model needs dimensions of the beam, the designer specifies the solid modeler as a source to provide geometric information about the arm. Once this information is given, the metamodel mechanism can maintain consistently the relationship between these two models.

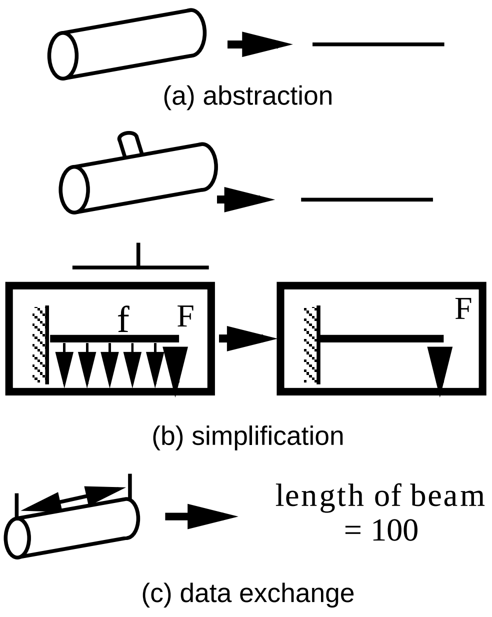


Figure 3.1: Modeling Process

4. Use the plugged modeler.
   After generating the model, the designer operates the model and generates information about design object. After completing the operation on the model, the designer propagate the result to the pluggable metamodel mechanism by using **propagate** menu button in the aspect modeler.

3.2 Functional Design on FBS modeler

At beginning of the design process, designer use the FBS modeler to do functional design. To open the FBS modeler, designer make new aspect model for the FBS modeler by following the step described in Section 3.1.

3.2.1 Function Decomposition

Function decomposition is a process to build a function hierarchy from the top functions given by the designer. The knowledge of developing methods in function prototypes is used by FBS Modeler to assist the designer.

And then, the designer inputs one or more needed function as shown in Figure 3.2. In this figure, the large window which includes **Function Layer** and **Behavior Layer** is the main workspace for constructing an FBS model, the Function Prototypes window depicts a list of the function prototypes, and the **Features for This Function** depicts a list of the physical features connected to a selected function. Here, the designer instantiates the needed function "move (table)" by selecting the function prototype and choosing the menu item **instantiate** in the **Function Prototypes** window.

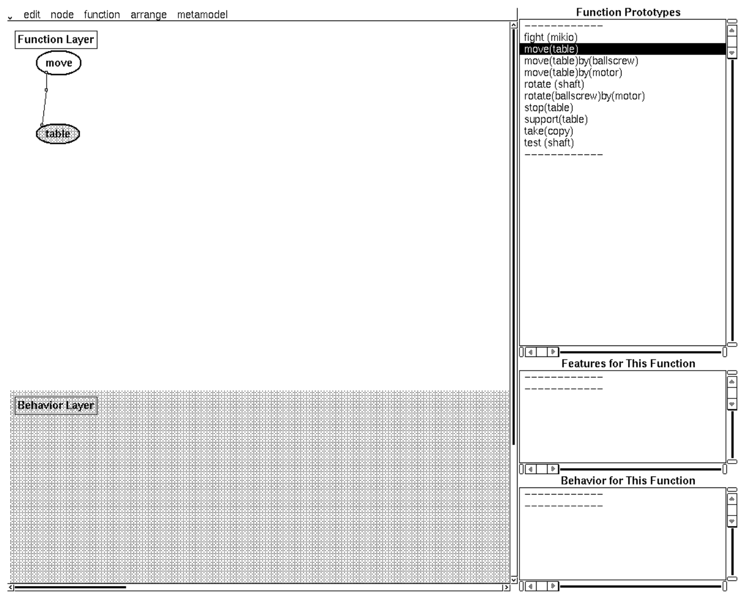
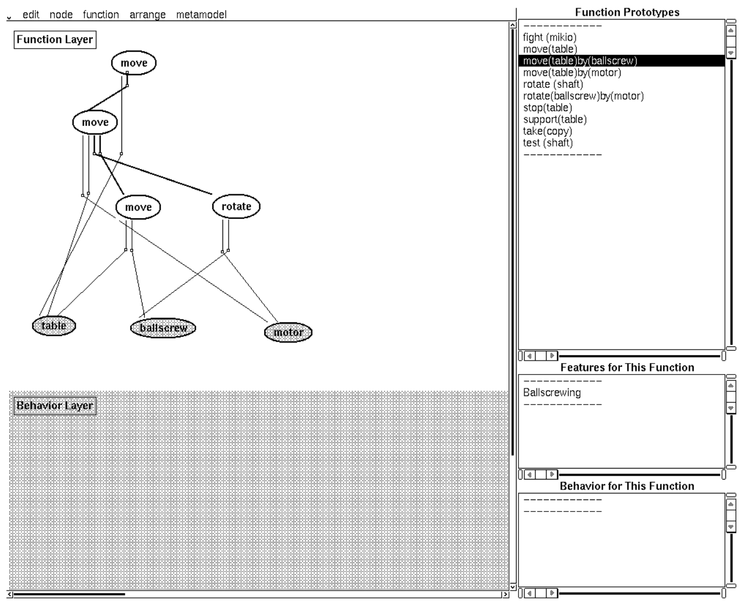


Figure 3.2: Selection of Needed Function

Then, the designer constructs a function hierarchy by decomposing the needed function by using appropriate *developing methods* defined in the function prototypes (see Figure 3.3). This is done by clicking the target function node for developing and choosing the menu item **function/develop [D]**. Here, the black lines, such as the relation between "move" and "rotate," represent super-sub relations in the function hierarchy. Of course, you can construct the function hierarchy without using the developing methods by instantiating some function prototypes and connecting them by choosing the menu item *node/connect [C]*.

As the second step, from the bottom functions of this function hierarchy, physical features will be derived, and by combining these physical features a primary model for the design object that realize the target function will be constructed. There are two ways to get physical features for each bottom functions. One is to



```
------------
fight (mikio)
move(table)
move(table)by(ballscrew)
move(table)by(motor)
rotate (shaft)
rotate(ballscrew)by(motor)
stop(table)
support(table)
take(copy)
test (shaft)
------------
```

```
------------
Ballscrewing
------------
```

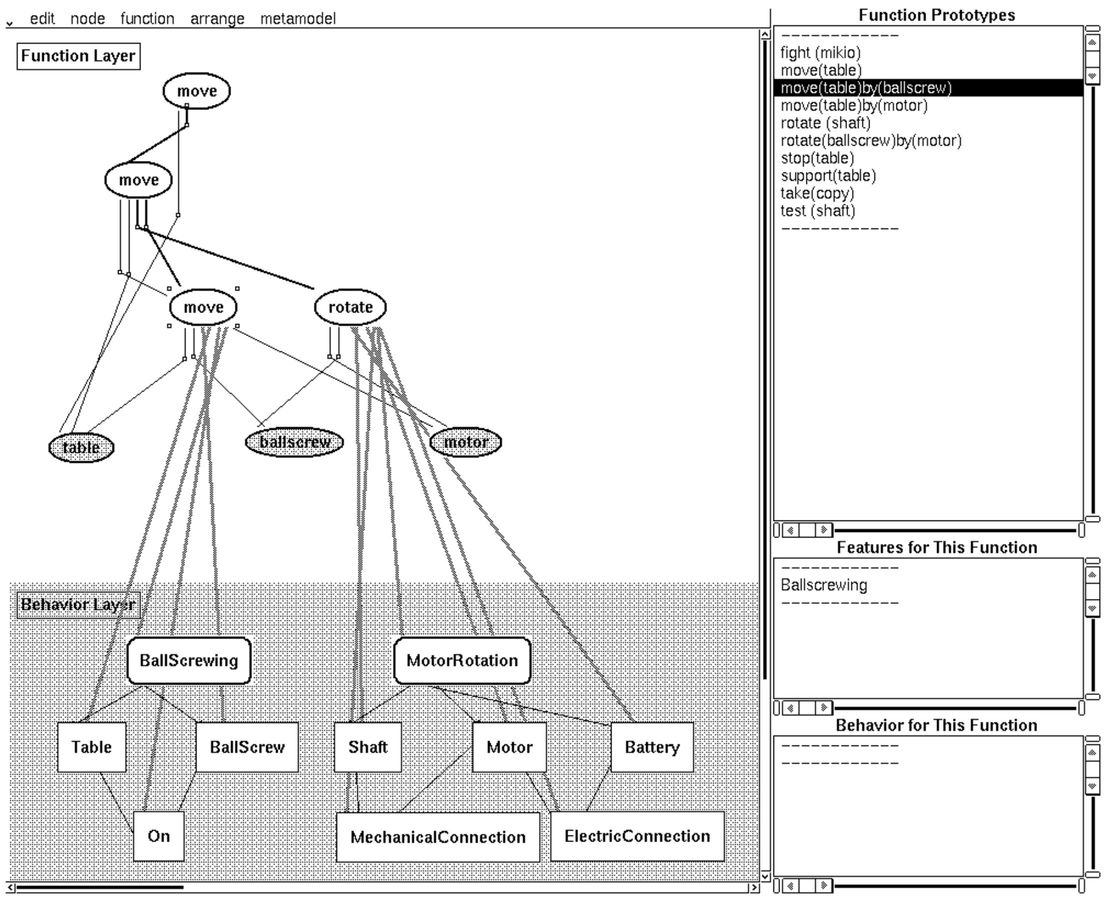
```
------------
------------
```

Figure 3.3: Construction of Function Hierarchy

directly select some physical features written in the function prototypes as knowledge, and the other is to create physical features from the behaviors which may be also written in the function prototypes.

3.2.2 Selection of Physical Features

The designer can select physical features for realizing the function hierarchy. Figure 3.4 shows the result of this selection. Namely, the designer chooses one of the functions in the lowest level of the function hierarchy[1], one of the physical features shown in the **Features for This Function** window, and the menu item *asRealizeFeature* in this window. In Figure 3.4, selected physical features are derived and decomposed into the concept dictionary elements (entities, relations, and physical phenomena) in the Behavior Layer. Thin arrows among the concept dictionary elements depict the physical dependencies, and dotted thick lines depict which concept dictionary element derived from which function.



----
fight (mikio)
move(table)
move(table)by(ballscrew)
move(table)by(motor)
rotate (shaft)
rotate(ballscrew)by(motor)
stop(table)
support(table)
take(copy)
test (shaft)

Ballscrewing

Figure 3.4: Selection of Physical Features

[1]In other words, the functions which do not have any subfunctions can be connected directly to physical features or behaviors.

3.2.3 Creation of Physical Features

Instead of selecting physical features directly, the designer can create some physical features from the behaviors attached to the function prototypes so that the designer can construct the design object model more flexibly.

First, the designer chooses one of the behaviors shown in *Behaviors for This Function* window, and open *QPAS interface* (Figure 3.5). Secondly, the designer selects the suggestions of physical phenomena which entail the desired behavior from “suggestions” window in QPAS interface. The physical feature editor will open and the designer can create new physical features under the assistance of the editor as explained in section 2.2.6.

If this method is used, “modes and conditions” which will be explained afterward are automatically set.

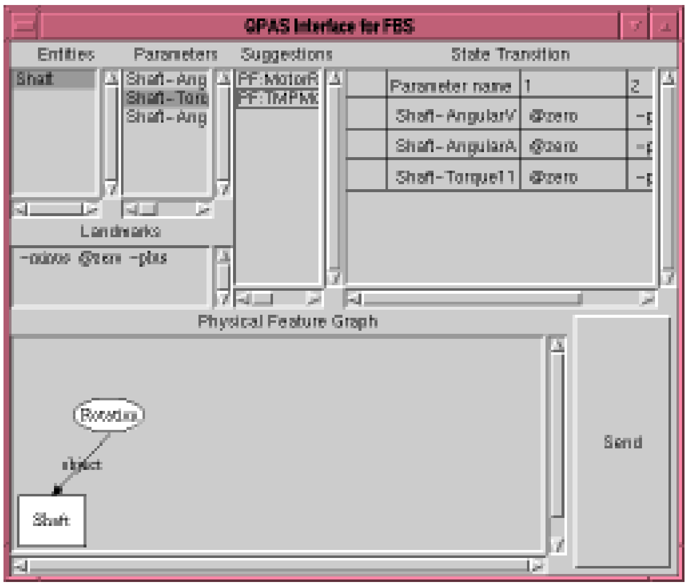
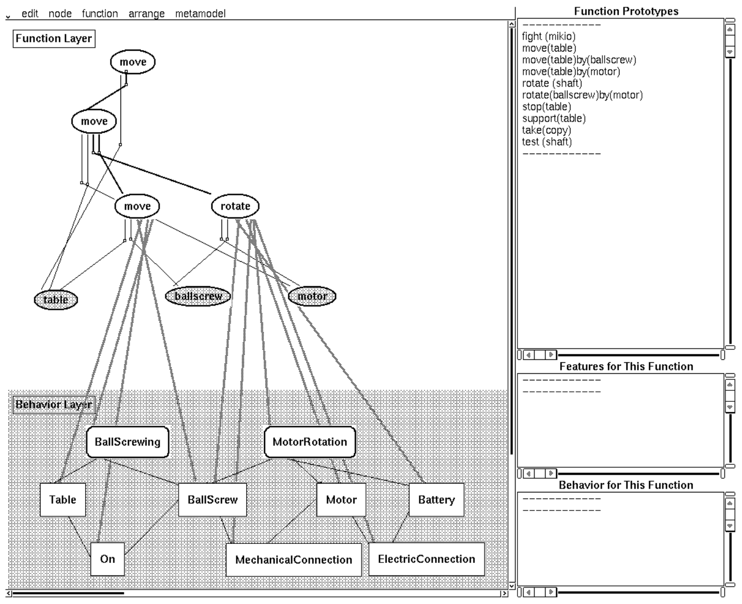


Figure 3.5: QPAS

3.2.4 Delegation

Since the physical features generated in the previous step are independent with each other, the designer should construct a consistent design object model, which is called as a *primary model* in the metamodel system. This is done by identifying same entities included in more than one physical feature. This manipulation is called delegation.

Delegation is the method to make an instance of entity which has more than two views. In a physical feature, each of entities has a certain view which depends on the physical feature. Therefore, when the designer combines several physical features, there may be some entities which are identical but described from different views. In the FBS modeler, this is done by delegation. For example, since "Shaft" and "BallScrew" in Figure 3.4 should be identical, the designer chooses these views, by clicking them with pushing left-shift key, and the menu item node/delegate [G] (see Figure 3.6).



```
------------
fight (mikio)
move(table)
move(table)by(ballscrew)
move(table)by(motor)
rotate (shaft)
rotate(ballscrew)by(motor)
stop(table)
support(table)
take(copy)
test (shaft)
------------
```

Figure 3.6: Example of Delegation

3.3 Evaluation with Other Modelers

As a result of the previous steps, a function hierarchy and a primary model of the design object are constructed. Since it can be physically inconsistent, this must be checked. For checking physical consistency, behavior simulation can be executed by using other modelers. To use other modelers, the designer propagates the FBS model to the pluggable metamodel mechanism by pushing **export** button in the aspect modeler interface.

After propagating the FBS model, the metamodel mechanism reasons out possible physical phenomena occurred on the design object by using physical feature knowledge base by selecting **metamodel** button at lower level and selecting **physical feature reasoning** button.

3.3.1 Evaluation with Qualitative Reasoning System

1. Make new aspect model for Qualitative Reasoning system. Make aspect model by following the step described in Section 3.1.

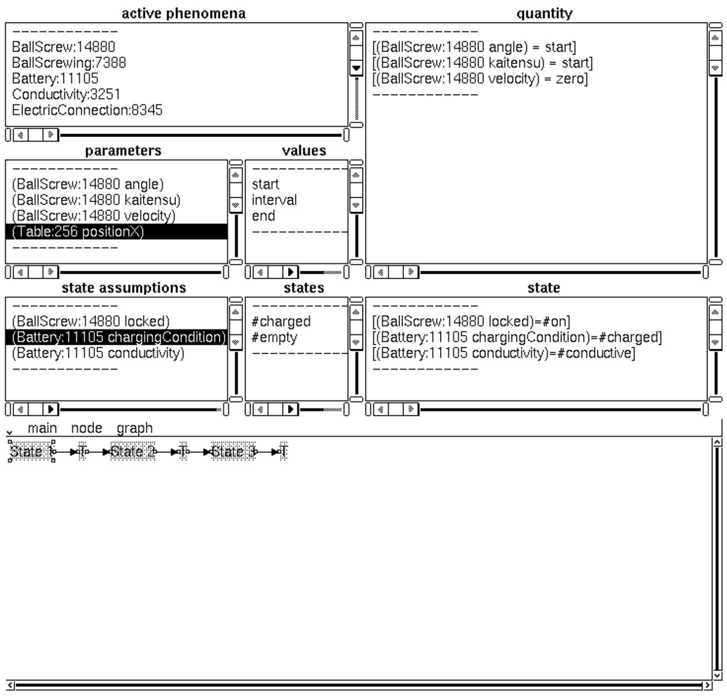
2. Export model from the metamodel mechanism
   Export metamodel to Qualitative Reasoning system by **Export** button. This method creates parameter network model by applying model library for Qualitative Reasoning system.

3. Accept parameter network model
   The designer accepts parameter network model by **Accept** button in Qualitative Reasoning system. At that time, parameters which have no relation with other parameters are neglected. If he/she does not satisfy the exported model, he/she can modify the parameter network manually.

4. Modify landmark of each parameter
   He/she opens the landmark modeler, by pushing **Q space** button and check the setting of parameter space by using this modeler. If he/she does not satisfy the parameter space, he/she can modify the parameter space.

5. Behavior simulation
   To execute behavior simulation, he/she constructs ATMS model for the design object by pushing **Interpretation** button. An envisioner opens and the designer can run behavior reasoning called *envisioning*. Figure 3.7 shows envisioning.

6. Propagate result to the metamodel mechanism



```
------------
BallScrew:14880
BallScrewing:7388
Battery:11105
Conductivity:3251
ElectricConnection:8345
```

```
------------
[(BallScrew:14880 angle) = start]
[(BallScrew:14880 kaitensu) = start]
[(BallScrew:14880 velocity) = zero]
------------
```

```
------------
(BallScrew:14880 angle)
(BallScrew:14880 kaitensu)
(BallScrew:14880 velocity)
(Table:256 positionX)
------------
```

```
------------
start
interval
end
------------
```

```
------------
(BallScrew:14880 locked)
(Battery:11105 chargingCondition)
(Battery:11105 conductivity)
------------
```

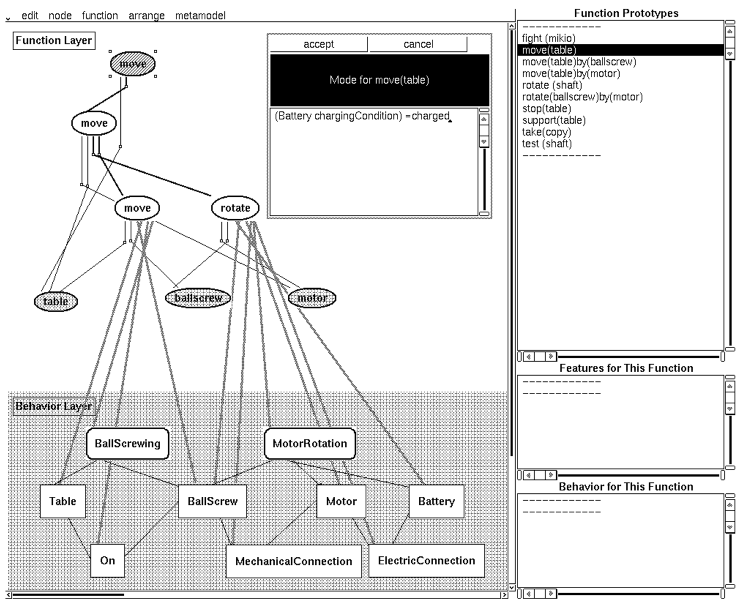
```
------------
#charged
#empty
------------
```

```
------------
[(BallScrew:14880 locked)=#on]
[(Battery:11105 chargingCondition)=#charged]
[(Battery:11105 conductivity)=#conductive]
------------
```

Figure 3.7: Envisioning

3.3.2 Evaluation with FBS Modeler

The designer can evaluate the constructed FBS model by comparing it with the result of simulation by exporting the result from the metamodel by using Aspect Modeler.



(Battery chargingCondition) =charged

Figure 3.8: Conditions

To evaluate the result of the qualitative reasoning system, the designer should describe the conditions which describes a function is considered to be realized. These conditions can be described in any function nodes. In the example, since the needed function "move (table)" can be considered to be realized when the position of the table moves from its initial position, the designer opens the **Condition** window by choosing the target function node "move" and the menu item **function/conditions [C]** and writes "(Table positionX) > start." This expression should be "(*entityName parameterName*) * *parameterValue*)" (* should be >, <, or =) and the user can write more than one such expressions in a condition window (they are parsed as *and* conditions).

By choosing the menu item *behavior reasoning/check consistency*, the system indicates the following information (see Figure 3.9);

Unrealizable physical phenomena: If physical phenomena described by the designer do not occur in the simulated result, some conditions should be inadequate. They are depicted as black rectangular nodes.

Side-Effects: Physical phenomena that are not expected to occur in the simulation may cause side-effects that the designer did not notice. These are **added physical phenomena** shown as hatched rectangular nodes.

Unrealizable functions: If functions have unrealizable views[2] in their F-B relationships, unrealizable subfunctions, or their conditions described above are not satisfied in the simulated network, they will not be realized. They are depicted as black oval nodes. Moreover, by choosing an unrealizable function and the menu item **function/reason**, the designer can know why the function is unrealizable in the **Reason** window.

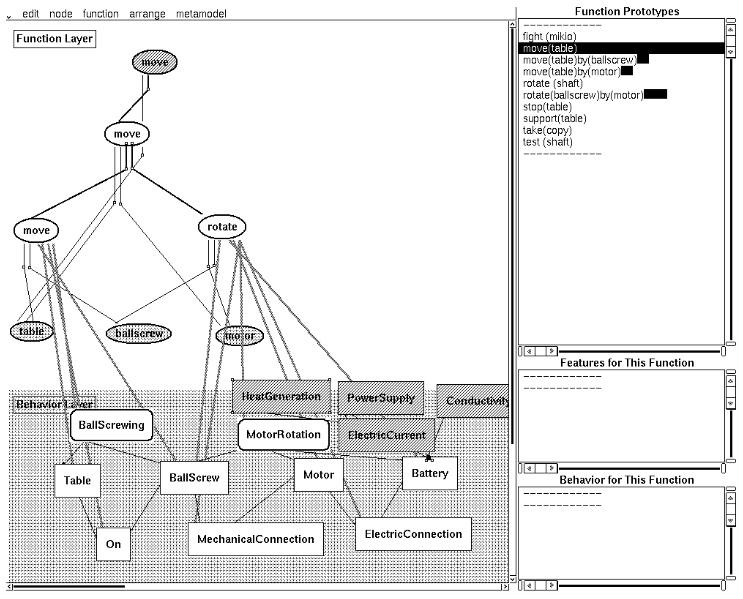


Figure 3.9: Example of Simulation Result

Unless satisfied with the result of evaluation, the designer repeatedly refines the function hierarchy and/or the view network. As a result, the FBS Modeler outputs the basic structure of the design object.

[2]Precisely speaking, even if a function has some unrealizable physical phenomena, a function are considered to be realizable as far as subclasses of unrealizable physical phenomena are occurred.

Bibliography

[Forbus84] K. Forbus: Qualitative Process Theory, *Artificial Intelligence*, Vol. 24, No. 3, pp.85–168 (1984).

[Ishii95] M. Ishii, T. Sekiya, and T. Tomiyama: 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).

[Umeda96] Y. Umeda, M. Ishii, M. Yoshioka, M. Yoshioka, and T. Tomiyama: Supporting Conceptual Design Based on the Function-Behavior-State Modeler, *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AIEDAM)*, Vol. 10, No. 4, pp.275–288(September 1996).

Appendix A

Plugged in Modelers

A.1 Physical Reasoning System based on Qualitative Process Theory

A.1.1 Representation of Physical Knowledge

In the KIEF, structures of design objects and physical phenomena which occur on the structures are described in extended QPT framework and stored in *Behavior Structure Knowledge Base*. This knowledge of behavior and structure is used in two ways. One way is to reason out the behavior of the design object based on its structure by *Qualitative Reasoning System*. The other is to support the designer building the structure, in *QPAS*, in the synthetic phase of design.

Qualitative Process Theory

Qualitative Process Theory("QPT") provides a framework to make models of physical systems and reasons out transitions of their states qualitatively. The notion "qualitative" is often explained by comparing the differences between conventional physics and "qualitative physics."

One difference is the value space of parameters which is the fundamental representation of the state of a physical system. In conventional physics, parameters have numeric values, but in qualitative physics parameters have qualitative values which consists of *landmarks* and *interspaces*. For example, let us think about temperature of water in a kettle and assume that we turn on the gas range. To describe the model of this physical system, QPT regards temperature of water as a ordered sequence of symbols shown in Figure A.1, each of which represents a certain state of water.

Another characteristic of QPT is that it explicitly describes causality between states of the physical system and physical phenomena which can occur on the system and may change its state. These descriptions of changes are used to reason transitions of states. For example, if gas is burning and temperature of the water is under the boiling point, then a phenomenon, "Heating", will occur and increase the temperature. When the temperature reaches to the boiling point, "Boiling" appears and, say, starts to decrease the amount of water. In this way, QPT can

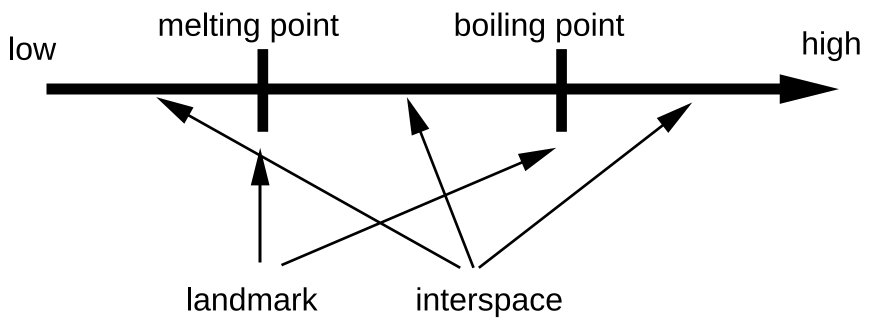


Figure A.1: Temperature of Water

model dynamical changes of phenomena.

Components of QPT models

A model in the framework of QPT consists of three kinds of conceptual components, which are;

- entities,
- relations, and
- physical phenomena.

These components are the elements of the qualitative model of a design object which is represented and reasoned by *Metamodel System*.

• An *entity* represents a physical existence, such as a gear, a spring, and a shaft. In the knowledge base, entities used for engineering design are collected and organized in the general-specialized hierarchy, e.g. gear–worm gear. The hierarchy has multiple general-specialized relationships, so that objects can be categorized in more than one way.

• A *relation* represents structural relationships among entities such as "on", "above", "support", and "connection". Relations are used to distinguish instances of the same entity class. For instance, it can be reasoned out that an electric current can be sent only to the cable "connected" to a battery, but not to the other cable which doesn't have any connections..

• A *physical phenomenon* is a key concept of causality recognized as mechanism of machinery. Therefore, in the extended QPT, a description of a physical phenomenon contains the conditions for the occurrence of this phenomenon and the effects caused by it. Physical phenomena are also organized in multiple general-specialized hierarchy.

In a QPT model, entities are connected by relations, and this network represents the topological structure of the physical system. A physical phenomenon is causally connected to entities and other physical phenomena. This shows that the physical phenomenon occurs on the entities, or it depends on the other physical phenomena connected.

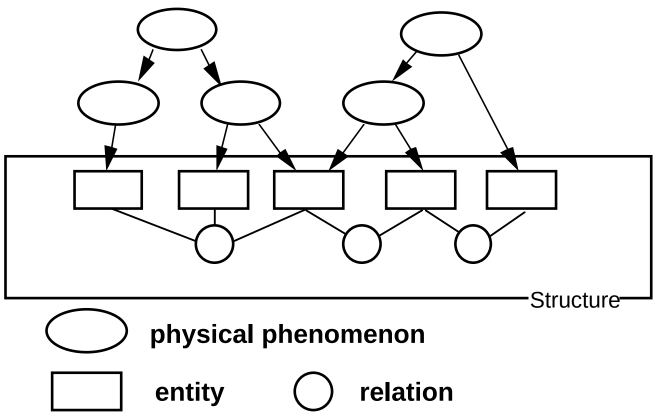


Figure A.2: An Extended QPT Model

An image of an extended QPT model is represented as Figure A.2.

Physical Feature

When knowledge about physical behavior used in conceptual design is collected, *physical features* will be useful basic units. A physical feature can be considered as useful parts of machines including the description of its behavior. Figure A.3 shows some examples.

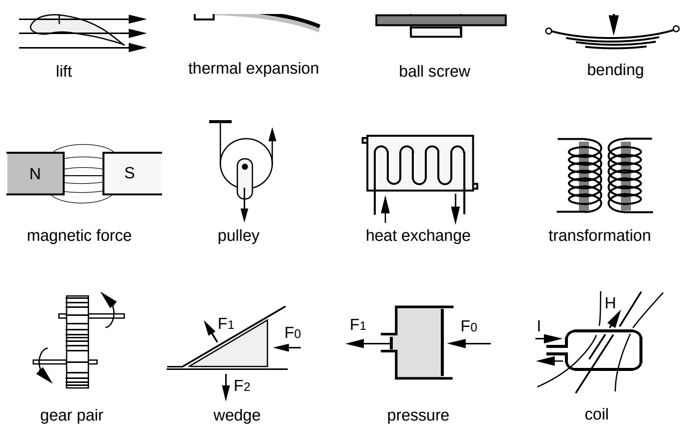


Figure A.3: Examples of Physical Features

A physical feature is represented by an extended QPT, a network of entities, relations, and physical phenomena as shown in Figure A.4. Physical Features will be used as model fragments combined to make a large model of the design object.

Parameters

Parameters are attributes of entities. They play an important role in representation of behavior. A set of values of the parameters represents the *state* of the physical

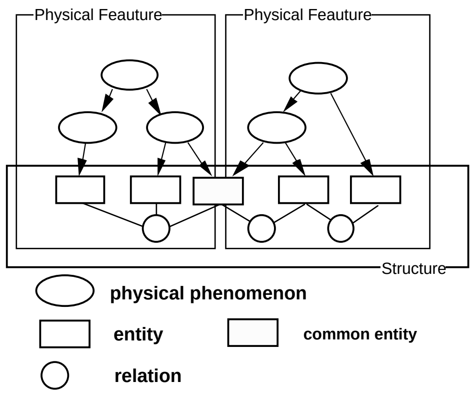


Figure A.4: Representation of Physical Feature

system at a certain point of time. The transitions of *state* show how the physical system works, and the sequences of states are the *behavior* of the physical system.

The parameters used in the physical systems are defined within knowledge frames of physical phenomena accumulated in physical knowledge base. In the frames, pairs of an entity and a parameter are described and the qualitative values that the parameters can take are attached. It is an important point that definitions of parameters are not written in the knowledge frames of entities. Rather, they are written in the frames of physical phenomena. However, in the QPT model, the instances of entities have the parameters.

There are two types of parameters attached to entities. One of the parameters is a quantity parameter, such as angular velocity of a gear. A quantity parameter has a qualitative value space consisting of landmarks and their interspaces. A quantity parameter may have a looped value space, e.g. angular displacement of a shaft. The other is a state parameter that symbolically describes the characteristic conditions of the entity, such as “on” or “off” for a switch. Operations to a device are represented as changes in the state parameters of the device. For example, a commutator of a motor switching electric current has a state parameter that shows the electric current passing through the coil, and by changing this state parameter adequately the motor rotates.

A.2 FBS Modeler

A.2.1 Representation of Functional Knowledge

Functions in FBS Modeler represent how to uses behaviors. Therefore, functional knowledge is collected from the teleological point of views.

FBS Modeler is the tool to collect functional knowledge and to assist the designer in functional design. For constructing the knowledge base of functions, the user should represent each prototype of a function in the form shown in Figure A.5. This is called a *function prototype*.

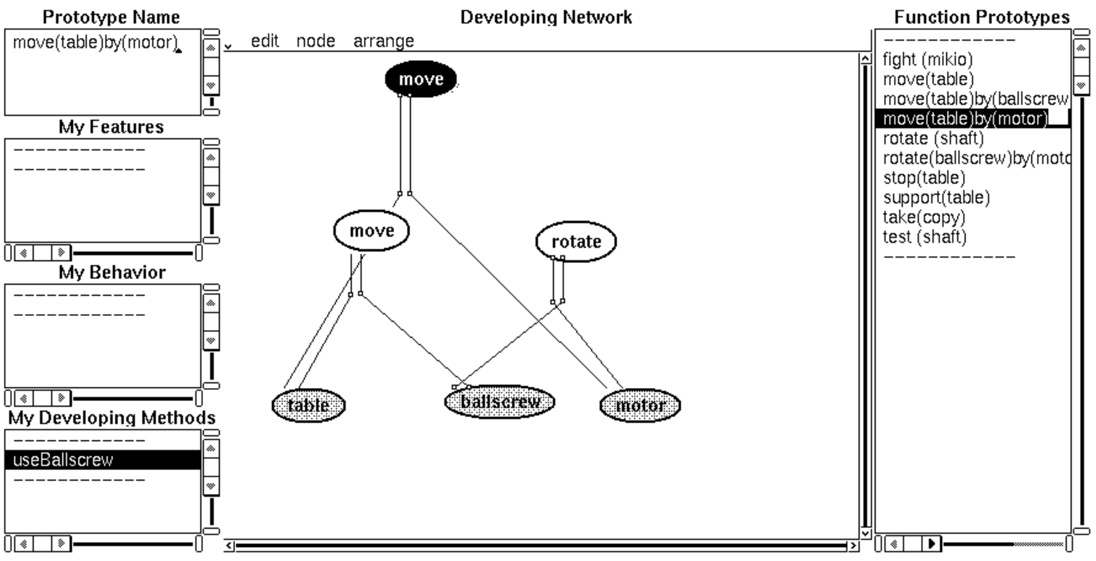


Figure A.5: Example of Function Prototype

A function prototype consists of slots as follows:

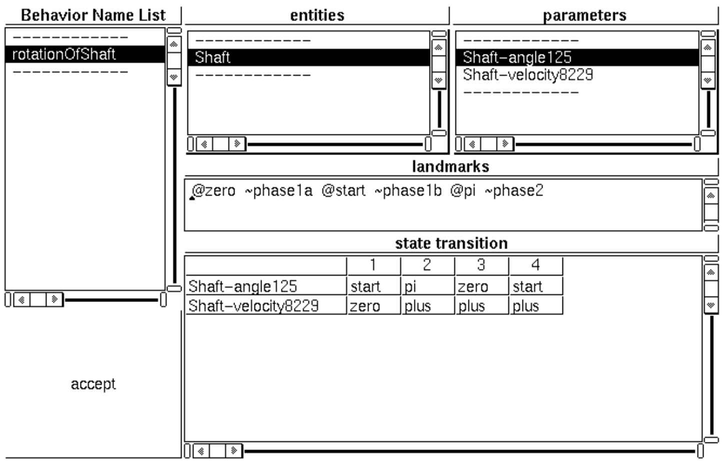
- a prototype name,
- developing methods,
- physical features, and
- behaviors.

A prototype name of a function is in the form of "*verb (objective1) (objective2)*." For example, "rotate(propeller)" are written as a name.

Developing methods describes a list of networks of subfunctions for decomposing this function. The connection among the target function and subfunctions means that the target function is realized if all the subfunctions are realized. Using this knowledge, a function is decomposed repeatedly, and a functional hierarchy is composed as the result of functional design.

Physical features which physically realize the target function are written in the function prototype. The physical features should be constructed in advance based on the extended QPT framework using *Physical Feature Editor*.

Behaviors are the description of state transitions which are the concrete interpretations of functions. These behaviors are represented as sequences of parameter values . For example, a function 'Rotate a shaft' is interpreted as below (Figure A.6. The FBS Modeler provides a table type interface to define the behaviors, called *FBS Behavior Browser*.



@zero ~phase1a @start ~phase1b @pi ~phase2

	1	2	3	4
Shaft-angle125	start	pi	zero	start
Shaft-velocity8229	zero	plus	plus	plus

Figure A.6: Example of State Transitions

A.2.2 How to write Function Prototypes

Function Prototype Editor is used to edit function prototypes. There is a list of function prototypes in the right window.

To create a new function prototype, write the function name at the top-left window and select "accept" from the menu in that window. To edit a function prototype already created, select the function prototypes in the right window and select "setAsTarget" from the menu in that window.

Adding Developing Methods

The developing methods are created in the middle window named "developing network."

First select subfunctions in the right window and select "instantiate" from the menu in the right window. The node of the subfuntion will come up. Although the node is not connected apparently, the system knows that they are connected.

The node created shows only the verb part so that the objective (noun part) should be created by "generate obj." If more than two verbs have a common objective , click the verb node created and then the objective node, then select "connect F-O" from the menu in the middle window to connect the nodes.

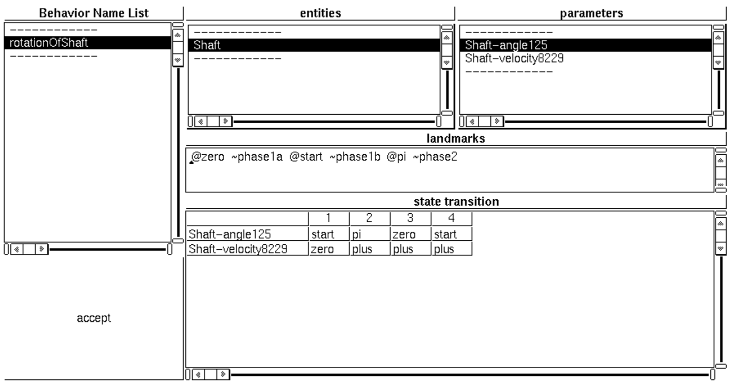
If the user apparently wants to describe the temporal order of the subfunctions, click the verb nodes of the subfunctions and use the command "add TimeDependentArc" from the menu in the middle window.

Adding Physical Features

In *My Features* window of the Function Prototype Editor, select "add" from the menu to add the physical feature that realize this function.

Adding Behaviors

To create a behavior for this function, select "edit behavior" from the menu in the *My Behaviors* window. *FBS Behavior Browser* will open (Figure A.7).



@zero ~phase1a @start ~phase1b @pi ~phase2

	1	2	3	4
Shaft−angle125	start	pi	zero	start
Shaft−velocity8229	zero	plus	plus	plus

Figure A.7: Behavior Browser

In this browser, the user should first select the entities, and then the parameters in order to describe state transitions. Selecting "define entity" from the menu in *entities* window, entities are derived from the knowledge base. After clicking one of the entities, select "define parameters" from the menu in *parameters* window. A list of parameters which can be applied to the entity comes up.

In the next step, the user should make state transitions represented as a matrix table in *state transitions* window. A row of the parameter is created through the previous step. The user creates columns by selecting "add column" from the menu of the window. These columns correspond to points of time. To put the value to each elements of the matrix, click the elements and select "landmark" or "interspace" from the menu. A list of landmarks or interspaces comes up. Select one of the values.

After all the elements are filled and the table is completed, click the button labeled "accept", then the table will be accepted. If there are parameters which have "cyclic" values, the behavior browser asks the user about the direction of changes in some time intervals.