An Investigation on Domain Ontology to Represent Functional Models

Munehiko SASAJIMA[†],**Yoshinobu KITAMURA**[†], **Mitsuru IKEDA**[†]
Shinji YOSHIKAWA[‡],**Akira ENDOU**[‡] and **Riichiro MIZOGUCHI**[†]

[†]The Institute of Scientific and Industrial Research, Osaka University,
8-1,Mihogaoka,Ibaraki,567 Japan
sasajima,kita,ikeda,miz@ei.sanken.osaka-u.ac.jp

[‡]Power Reactor and Nuclear Fuel Development Corporation,
4002 Narita-Cho,Oarai-Machi,
Ibaraki Pref, 311-13 Japan

Abstract

Although a lot of researchers have pointed out the significance of functional representation, the general relations between function and behavior is not fully understood yet. We consider the knowledge of each component in a system as consisting of two elements. One is a necessary and sufficient concept for simulation of the component which we call behavior. The other is the interpretation of the behavior under a desirable state which the component is expected to achieve, which we call function. By classification of a primitives necessary for the interpretation of the behavior in various domains, which we call "domain ontology", we can capture and represent the function by selection and combination of the primitives. This paper proposes the primitives we identified and the method to use them for representing function. Also we investigate the relation between function and behavior based on the primitives. As the primitives can represent concepts at various levels of abstraction, they will contribute to those tasks which rely on the simulations on the model of the target object, such as diagnosis, design, explanation, and so on.

Introduction

Model-based simulation has been utilized for solving various problems, such as diagnosis(Kitamura *et al.* 1994)(Hirai *et al.* 1991) (Sembugamoorthy & Chandrasekaran 1986)(Abu-Hanna, Benjamins, & Jansweijer 1991), design(Vescovi *et al.* 1993)(Iwasaki *et al.* 1993), explanation(Swartout, Paris, & Moore 1991)(Gruber & Gautier 1993) and so on. In order to promote such model-based problem solving, the concepts of behavior and function have to be understood in depth, since they provide us with a firm foundation of the methodology.

The following two reasons support the significance of the research about the concept of function.
1)Investigation on the concept of function contributes to fault diagnosis. The task deals with the concept of trouble of a component which can be regarded as a loss of its function. Capturing the concept of trouble and hence function is necessary to achieve efficient diagnosis.
2)Investigation of function also contributes to explanation of those tasks performed by expert systems. Using the concept of function, an expert system can reason how a component's behavior is significant in the system. Furthermore, the system can explain the system's behavior in terms of those concepts familiar to the users at an appropriate level of abstraction, which helps them understand the explanation.

Another issue to discuss is clear understanding of the difference between behavior and function. When we interpret behavior of a component, we employ certain viewpoints from which more than one interpretation is made for a behavior of a component. Necessity of a specific output of the component is an example of the viewpoint. For example, suppose a component whose behavior is to divide an input saline solution into pure salt and a saline solution. The behavior is interpreted as producing salt in the system which requires salt. In the system which requires fresh water, however, the same behavior is interpreted as desalinization.

A lot of research has been done to date aiming at a deep understanding of both behavior and function. J.de Kleer(de Kleer 1984) defines function of a component as a combination of behavior and selection of it from the set of possible behaviors of the component. Although the information required for the selection can be one of the candidate factors necessary to interprete behaviors, it is not enough. Two different interpretations, for example, are possible for one behavior of a heat exchanger, "shift thermal energy from fluid of higher temperature side to lower temperature side". One is that "the component gets the fluid of the lower side warmer", and the other is "the component gets the fluid of the higher side colder". de Kleer's definition of function does not explain this example.

V.Sembugamoorthy and B.Chandrasekaran(Sembugamoorthy & Chandrasekaran 1986) propose a framework to represent the function of a system by combination of function and behavior of each component of the system. In their work, declaration of a desired state and the environment in which the function of the system appears can be regarded as attached information. As relations between the attached information and the concept of the function are not discussed so much, proposed framework seems to have much room to refine.

Anne M.Keuneke(Keuneke 1991) classifies function into four concepts, such as To Make,To Maintain,To Prevent,and To Control. We regard them as concepts at the top level of function hierarchy which should be refined further from various view points.

The discussion we have made thus far shows there is no satisfactory theory about behavior and function in spite of its importance. We classify the knowledge about each component of a model into three elements. The first element is required to have necessary and sufficient information enabling simulation of how the the system works, by combining all the elements in the system without referring to the other components or the whole system. We use the term "behavior" to refer to this element.

Next, we recognize intended desirable states which each component is expected to achieve. Such a state is the second element, and we use term "goal"to refer to it.

Lastly, we interpret the behavior of each component under a related goal. We use the term "function" to refer to the interpretation result (the third element). Although function of a component cannot be defined without referring to the whole system, it can be described for each component.

According to the above discussion, we come up with the following definition of function:
function = behavior + attached information
in which attached information is some information which make function different from behavior.

Identification of primitives to represent the attached information plays a critical role in capturing the concept of function. When the identification is accomplished, functional representation of a component is easily made by selection and combination of the primitives. Furthermore, to make the primitives domain-independent enhances re-usability of them which constitute a portion of "Domain Ontology".

One of the bottlenecks in model building is the existence of a gap between model builder's concepts about a component and behavior of the component to be represented. Identification of the primitives to represent function helps decrease the gap.

Our long term goal includes to organize the concepts of function and behavior as reusable building blocks for qualitative models which facilitates model-based problem solving. As the first step toward this goal, we investigate the primitives for describing attached information and organize them as a portion of domain ontology. This paper proposes a new method to describe models of behavior and function of components. Also we show potential of the method by describing models on different domains at various levels of abstraction.

Primitives to represent the function

To establish a framework to describe a functional model of components, we investigate several viewpoints to capture the behavior and the additional information to interpret it. In this section, we discuss the viewpoints and primitives to represent them in order.

Overview of the components description

There are two ways of behavior representation. One is process-centered way which views how the substances under consideration are processed. The other is device-centered way which concentrates on input-output relations of a component. We employ the latter way and represent each component as a black box which has ports for input and output. Some components, e.g. a tank, have only ports for input and other components, e.g. a battery, have only ports for output. We call an input substance "In-Obj" and an output substance "Out-Obj". Except the cases in which clear distinction is necessary, we use the term "Compo-Obj" to refer to one of those substances treated by a component. We treat substances (such as water) and energy (such as heat) which exhibit functionality of the component as Compo-Objs. Description of them is based on object-oriented paradigm.

Description of the behavior

The behavior of a component is represented by its Compo-Objs and relations among them. This section describes how to describe them in order.

Description of Compo-Objs Two types of Compo-Obj are discussed: one consists of only one kind of substances or energy referred to as basic Compo-Obj and the other consists of more than one kind of substances or energy referred to as mixed Compo-Obj.

Basic Compo-Objs. The following six viewpoints are employed to represent basic Compo-Obj.

- The Compo-Obj is energy or substance
- The location at which the Compo-Obj exists
- Super class of the Compo-Obj
- Parameters which represent the Compo-Obj
- Relations among the parameters
- Phase of the Compo-Obj

Mixed Compo-Objs. There are Compo-Objs which consist of more than one kind of Compo-Objs.

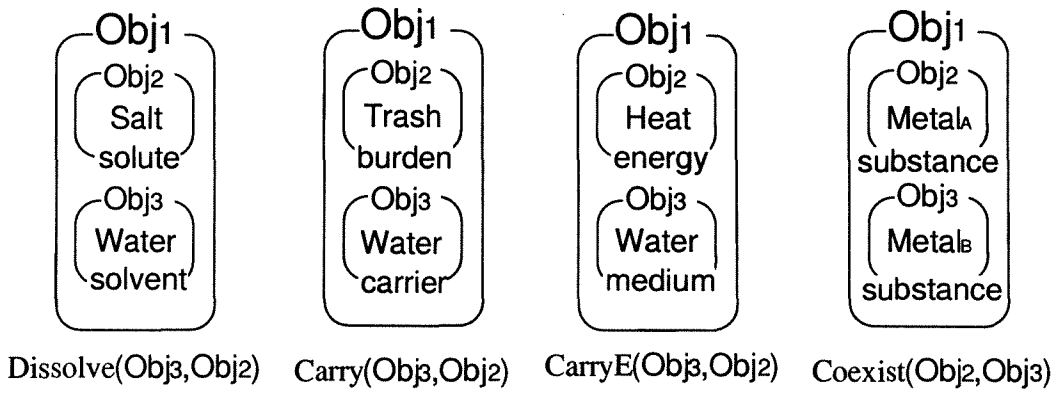


Figure 1: Mixed Compo-Objs

For example, a saline solution consists of salt and water and an alloy consists of $metal_A$ and $metal_B$. We call such a Compo-Obj as a Mixed Compo-Obj and each of those Compo-Objs which together compose a Mixed Compo-Obj as a Compo-Obj-Composer. As an example of saline solution shown in Fig.1, we capture Mixed Compo-Obj as a Compo-Obj which includes Compo-Obj-Composers with their special relations. We employed four kinds of such relations, and represent them by predicates as follows.

Solute and solvent

One of the two Compo-Obj-Composers dissolves the other Composer like a saline solution.

Predicate:**Dissolve**(Obj_x,Obj_y)
Obj_x:solvent,Obj_y:solute

Carrier and Burden

Although not being in "dissolve" relation, one of the two Compo-Obj-Composers relies its mobility on the other Composer like trash in water flow.

Predicate:**Carry**(Obj_x,Obj_y)
Obj_x:carrier,Obj_y:burden

Medium and Energy

One of the two Compo-Obj-Composers carries the energy class of Composer as a medium. Example here is water and heat energy that together compose boiling water.

Predicate:**CarryE**(Obj_x,Obj_y)
Obj_x:medium,Obj_y:energy

$Substance_A$ and $Substance_B$

Two Compo-Obj-Composers coexist in one Compo-Obj and give no explicit effect to each other like two kinds of metals in an alloy.

Predicate:**Coexist**(Obj_x,Obj_y)
Obj_x:$substance_A$,Obj_y:$substance_B$

We represent the relation between water and salt in a saline solution, for example, as

Dissolve(water,salt)

In this paper, we use the term Compo-Obj to refer to both a Compo-Obj consisting of one kind of substances or energy and a Mixed Compo-Obj unless there is a necessity to distinguish them.

Relations among parameters. The amount of energy which a Compo-Obj possesses has a close relation to parameters representing the Compo-Obj itself.

For example, consider boiling water (water brings heat energy). The amount of heat energy is in proportion to the temperature and volume of the water which brings the heat. Explicit description of this relation allows us to reason the transition of the amount of heat energy by transition of parameters of water.

Format for describing Compo-Objs. According to the above discussions, we come up with the following template for Compo-Obj representation (Fig.2):

Class Name: Compo-Obj
Attributes:
- Name: ID of the Compo-Obj.
- ISA: Super class of the Compo-Obj:Sodium,Water,Heat,etc.
- Params: Parameters to represent physical feature of the Compo-Obj.
- E-Flag: If the Compo-Obj is energy then T.
- Location: The location where the Compo-Obj exists:In,Out,etc.
- Para-Relations: Relations among the Params of the Compo-Obj.
- Phase: Phase of the Compo-Obj:Solid,Liquid,Gas.
- Sub-Objs: The Compo-Obj-Composers of the Compo-Obj.
- Sub-Relations: Relations among the Compo-Obj-Composers.

Figure 2: Template for describing a Compo-Obj

Description of relations among Compo-Objs To represent relations among Compo-Objs in various levels of abstraction, we employ three viewpoints discussed below. Fig.3 represents an abstract model of an ideal turbine without loss of energy. With this example, this section describes the viewpoints in order.

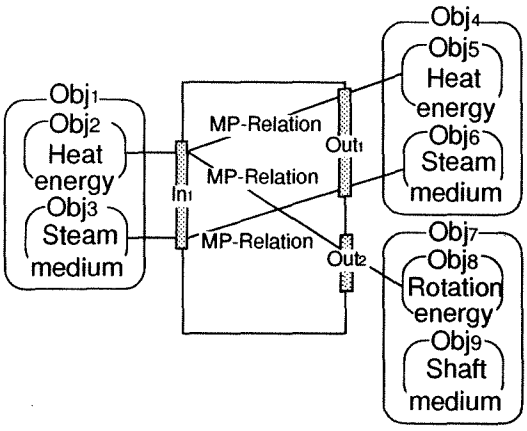
MP-Relations:Relations among In-Objs and Out-Objs. Some Out-Objs are made from specific

In-Objs. For example, we can recognize water produced by a desalination plant is made from saline solution. We call a set of such relations of a component as MP-Relation, and the following predicate represents them.

MP([Out-Obj$_1$,...,Out-Obj$_N$],[In-Obj$_1$,...,In-Obj$_M$])

In Fig.3, part of heat energy(Obj$_2$) is converted to rotation energy(Obj$_8$) of the shaft and rest of the heat energy is out without conversion. Thus the following predicates together represent MP-Relation of this turbine.

MP([Obj$_5$,Obj$_8$],[Obj$_2$])
MP([Obj$_6$],[Obj$_3$])



MP-Relations:
MP([Obj5,Obj8],[Obj2]),MP([Obj6],[Obj3])

SameClass:
Obj2.ISA=Obj5.ISA, Obj2.ISA≠Obj8.ISA ,Obj3.ISA=Obj6.ISA

QN-Relations:
Obj8.Amount=k* Obj2.Amount+C
Obj2.Amount=Obj5.Amount+Obj8.Amount
Obj6.Amount=Obj3.Amount
Obj6.Velocity=Obj3.Velocity
... ...

Figure 3: Model of a turbine

SameClass:the sameness of the class. The behavior of the turbine in Fig.3 is viewed as to convert input energy to different kind of energy at an abstract level. This kind of relation is partially characterized from the viewpoint of the sameness of the class between the two Compo-Objs. We describe the sameness by two operators, = and ≠. In Fig.3, the following relations exist.

Obj$_2$.**ISA** = Obj$_5$.**ISA**
Obj$_2$.**ISA** ≠ Obj$_8$.**ISA**
Obj$_3$.**ISA** = Obj$_6$.**ISA**

QN-Relations:Quantitative relations. Relations among the **Params** of Compo-Objs are represented by a set of equations, called **QN-Relations**.

In Fig.3, Obj$_8$.**Amount**, the amount of rotation energy to be output, is in proportion to Obj$_2$.**Amount**, the amount of heat energy. At the same time, the summation of the amount of output heat energy and rotation energy equals to the amount of the input heat energy according to the assumption of no loss of energy. The following two equations reflects these relations, respectively.

Obj$_8$.**Amount** = K * Obj$_2$.**Amount** + C
Obj$_2$.**Amount** = Obj$_5$.**Amount**+Obj$_8$.**Amount**

Description of the attached information

In order to describe the function of components, we investigated the attached information and obtained the primitives from four viewpoints: (1)goal of the component (2)function type (3)focus on Compo-Objs (4)necessity of Compo-Objs. These four items are explained in this subsection.

Goal of the component We recognize a desirable state to achieve for each component in a system and the term "goal" refers to the concept. There are two types of such states. One is described by the combination of parameters and desirable values and represents its desirable state absolutely, that is, independently of input. The other is described by input-output relations and represents the desirable state relatively to the input of the component. Example here is the heat exchanger in Fig.4. A goal description "temperature of the output coolant(Obj$_{10}$) does not exceed five hundred degree centigrade" is an example of the former type of the goal and "the heat energy of the coolant (Obj$_1$) is decreased into one tenth by the other input coolant (Obj$_4$)" is an example of the latter type. We represent such a desirable state by the predicate **G**(*state*).

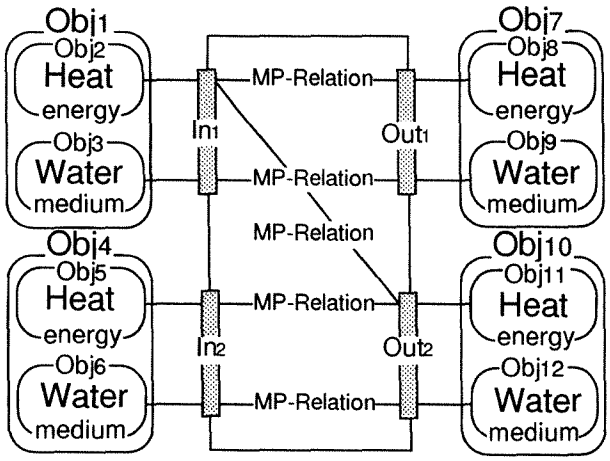


Figure 4: Heat exchanger

Function type Anne M.Keuneke classifies concept of the function into four types, such as To Make,To Control, To Maintain, and To Prevent(Keuneke 1991). For our representation, we give informal definition to them.

To Make: To set a parameter at a desirable value.

To Control: To shift a parameter of desirable value to another desirable one.

To Maintain: To keep the value of parameters desirable for certain period.

To Prevent: Not to make parameters to take special values which represents no good states of the system.

To Make type function is the basis of all other functions which makes parameters to be in a desirable range. Function of a cooking stove, for example, is considered as To Make temperature of a thing put on it to be more than a certain degree. If the aim of the stove is to boil a kettle of water, then the function To Make is achieved when the temperature inside the kettle exceeds one hundred degree centigrade. On the other hand, the function of an electric water heater with a sensing device which accurately makes the temperature of the water in it to be ninety five degree centigrade is To Control.

Function of a component which keeps desired state by To Control function is To Maintain.

A component whose function is To Prevent watches and controls parameters not to go undesirable states for the system. A relief valve attached to a tank To Prevent explosion of the tank watches pressure caused by substances inside the tank. The valve switches its behaviors: not to let the substances pass the valve out of the tank and to release the substances through the valve according to the pressure inside the tank.

Combining the description of the goal and the function type, we obtain an abstract explanation pattern of the function of a component, such as "function type" + "goal state" whose instance is "To Prevent Temperature of Compo-Obj$_N$ becomes ninety five degree centigrade".

Using the goal state and function type concept of trouble of a component is classified into four types:

ToMake: Trouble if the component does not achieve any goal.

ToControl: Trouble if the component in a goal does not shift to another goal.

ToMaintain: Trouble if the component does not keep on the achieved goal for certain period.

ToPrevent: Trouble if the goal parameter takes specific value which represents no good state.

Focus on Compo-Objs When we capture the main function of a component, we focus on a specific Compo-Obj's class. We represent such a concept by the predicate **Focus**(class of Compo-Obj). When we interpret the behavior of the component in Fig.5 as that of a resister which lowers potential of input direct current

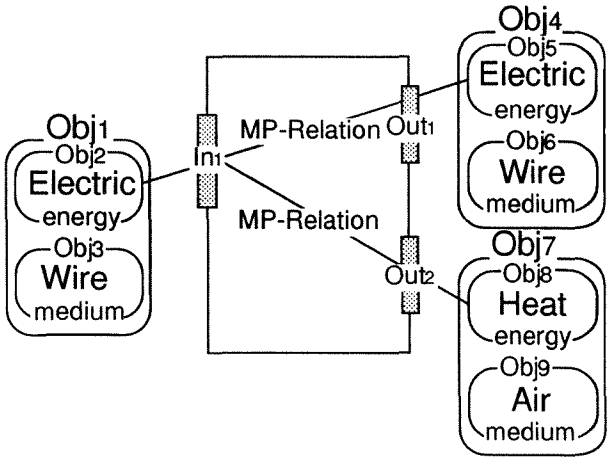


Figure 5: Resister

electricity, focus is given to the electric energy and **Focus**(Electric energy) represents it. On the other hand, the same behavior is interpreted as that of an electric heater when we focus on the heat energy.

Necessity of Compo-Objs Compo-Objs which belong to the focused class often exists at different ports of a component from each other. For example, consider a behavior of the heat exchanger in Fig.4 which focuses on heat energy existing at the four ports. The heat exchanger can be interpreted not only as a heater giving the heat energy to the colder Out-Obj(Obj$_{10}$), but also as a cooler taking the heat energy of the hotter In-Obj(Obj$_2$) away.

Difference between the two interpretations is caused by difference of the necessity of each heat energy at different port, according to the goal of the component. We represent a focused class of Compo-Obj is necessary at a port by the predicate **Need**(name of the port,focused class) and not necessary by the predicate **NoNeed**(name of the port,focused class). When we interpret the behavior of the component in Fig.4 as that of heater, Obj$_{11}$ is necessary at the port Out$_2$, thus **Need**(Out$_2$,Heat)is suitable. Also when we interpret the same behavior as that of cooler, Obj$_8$ is not necessary at the port Out$_1$, thus **NoNeed**(Out$_1$,Heat)is suitable.

In some cases ports which do not deal with focused class of Compo-Objs play important roles. Consider Fig.5 as an abstract model of a resister. Used in an ordinary circuit, we do not have to care the heat energy output from the resister. Used in a precise circuit, however, the heat energy gives a harm to the circuit. Taking such port and harmful Compo-Obj's class as arguments of **NoNeed(/2)**, the side effect of a component can be explicitly represented. For example, the

resister's side effect is represented by
NoNeed(Out_2,Heat).

Format to describe components

According to the above discussion, we propose a template for describing the behavior and the function of a component(Fig.6).

Class Name: Component
Attributes:
Ports:
Compo-Objs:
MP-Relations:
SameClass:
InherentParams:
QN-Relations:
Goal:
FuncType:
Focus:
Needs:

Figure 6: Template for describing a component

Evaluation of the representation method

One of the causes of difficulty in model building is existence of a gap between conceptual level of model builder and that of a vocabulary supplied by a system for model building. A vocabulary to fill the gap is required to represent components at various levels of abstraction. This section evaluates our representation method from this viewpoint.

Hierarchical classification of the function and the behavior

In the last section, we defined a format to represent a component captured from various viewpoints. Using these viewpoints, we came up with a hierarchical classification of the concept of the behavior and the function as shown in Fig.7.

A concept of a component which deals with one In-Obj and two Out-Objs is represented as the root node. The concept is classified into five concepts at the first level of the tree according to the condition of the composition of the In-Obj. According to the condition of composition of the two Out-Objs and their relation to the In-Obj, each node made by the first division is further classified at the second level of the tree.

Classifications at the first and the second level are based on those viewpoints used for behavior description and is domain- and context-independent.

According to the viewpoint of focus on the class and necessity of each focused Compo-Obj, concepts represented at the second level are classified into the third level concepts. Some of the concepts represented at the third level can be connected to those verbs we often use to represent the concept. As is discussed below, we see there still remain several viewpoints to be investigated other than we have already done. Identification of those primitives to represent function from the viewpoints will enable us to promote further classification beyond the third level.

Representation at various levels of abstraction

Here we show some examples to evaluate the proposed representation method. The concept of a behavior "an In-Obj is a solution, one of the Out-Objs is pure solute derived from the In-Obj, and the other Out-Obj is rest of In-Obj", (represented as the node with thick circle at the second level in Fig.7) which is represented in Fig.8, is described as follows (Fig.9).

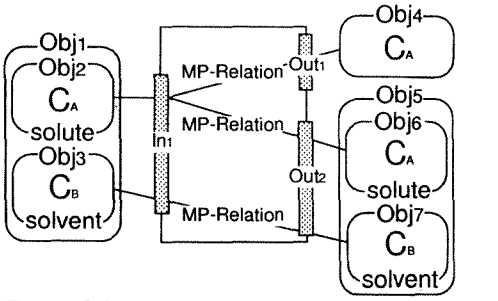


Figure 8: divide a solution

Now let C_A be a class identifier to which Obj_2,Obj_4 and Obj_6 belong (Fig.8). Attaching **Focus**(C_A) and **Need**(Out_1,C_A) to the above description of the behavior changes it into the description of the function corresponding to "Extract solute from In-Obj". Replacing **Need**(Out_1,C_A) by **NoNeed**(Out_2,C_A) the last description of the function changes into different function, "Dilute In-Obj". Furthermore, attaching **G**(Obj_6.Mass = 0) to the description of "Dilute In-Obj", the function changes into another function, "Purify In-Obj".

Attaching the following descriptions(Fig.10) to the function,"Purify In-Obj", the function of a kidney as a component of the human to maintain purifying blood by filtering waste dissolved in blood is represented.(Fig.11)

Replacing some parts of the description of the function of a kidney by the following descriptions(Fig.12), desalinization system as a component of the plant to make saline solution desalt is represented.(Fig.13)

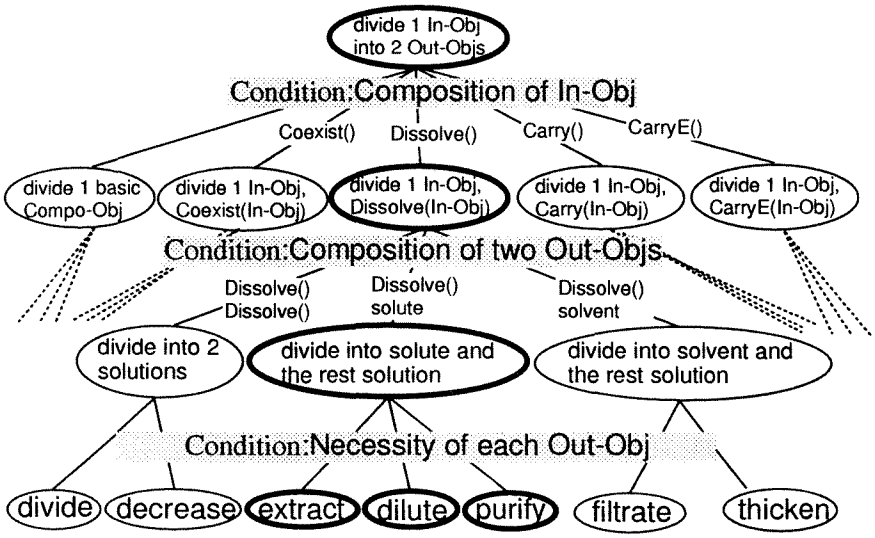


Figure 7: Classification of the behavior and function(part)

Objects:

Name:Obj1;**Location**:In1;**Phase**:Liquid;
Params:[Mass,Volume,Calory,Velocity,...];
Sub-Objects:Obj2,Obj3;
Sub-Relations:**Dissolve**(Obj3,Obj2);]

[**Name**:Obj2;**Location**:In1; **Phase**:Solid;
Params:[Mass,Volume,Calory,Hardness...] ;]

[**Name**:Obj3;**Location**:In1;**Phase**:Liquid;
Params:[Mass,Volume,Calory,Velocity,...];]

[**Name**:Obj4; **Location**:Out1; **Phase**:Solid;
Params:[Mass,Volume,Calory,Hardness,...];]

[**Name**:Obj5; **Location**:Out2; **Phase**:Liquid;
Params:[Mass,Volume,Calory,Velocity,...];
Sub-Objects:Obj6, Obj7;
Sub-Relations:**Dissolve**(Obj7,Obj6);]

[**Name**:Obj6; **Location**:Out2; **Phase**:Solid;
Params:[Mass,Volume,Calory,Hardness,...];]

[**Name**:Obj7; **Location**:Out2; **Phase**:Liquid;
Params:[Mass,Volume,Calory,Velocity,...];]

MP-Relations:

MP([Obj4,Obj6],[Obj2]),
MP([Obj7],[Obj3])

SameClass:

Obj4.ISA = Obj2.ISA,
Obj6.ISA = Obj2.ISA,
Obj7.ISA = Obj3.ISA

Figure 9: description of the concept,"divide a solution (part)"

Obj2.ISA:	Waste
Obj3.ISA:	Blood
Obj4.ISA:	Waste
Obj6.ISA:	Waste
Obj7.ISA:	Blood
QN-Relations:	Obj4.Amount $\leqq$ Obj2.Amount
	Obj6.Amount $\leqq$ Obj2.Amount
	Obj7.Amount = Obj3.Amount
Focus:	Focus(Waste)
FuncType:	To Maintain

Figure 10: Additional values for description of Kidney

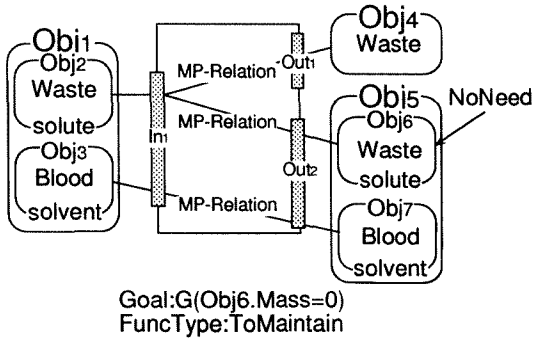


Figure 11: The function of a kidney

Obj_2.ISA:	Salt
Obj_3.ISA:	Water
Obj_4.ISA:	Salt
Obj_6.ISA:	Water
Obj_7.ISA:	Salt
QN-Relations:	$Obj_4.Amount \leqq Obj_2.Amount$
	$Obj_6.Amount \leqq Obj_2.Amount$
	$Obj_7.Amount = Obj_3.Amount$
Focus:	Focus(Salt)
FuncType:	To Make

Figure 12: Additional values for description of a desalinization system

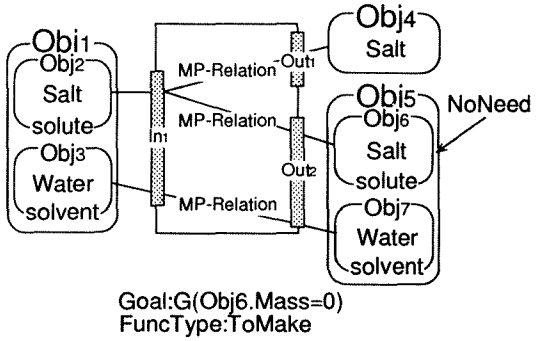


Figure 13: The function of a desalinization plant

We can describe a model of a component for simulation by attaching detailed relations among its Compo-Objs and parameters inherent to the component.

As demonstrated in this section, the representation method shows its potential to describe components of various domains at various levels of abstraction.

Example of functional modeling(2)

Here we show another example of the representation of different interpretations of one behavior.

Figure 14 shows a model of behavior of a component which converts direct current electricity input(Obj_2) to increases force(Obj_5) and reaction(Obj_6) of a fluid(Obj_7) from another input port.

Suppose Obj_7 is air. Focusing the force, represented by the predicate **Focus**(Force), behavior of the component is interpreted as that of an electric fan which raises wind. Next, suppose Obj_7 is water. Fucusing the reaction from the water, the same behavior is interpreted as that of a screw to gain driving force of the component itself. The predicate

Focus(Reaction)

represents it.

Example of functional modeling(3)

Figure 15 is a behavior model of a heat exchanger(HX). Suppose the HX is used in two ways, for cooling system

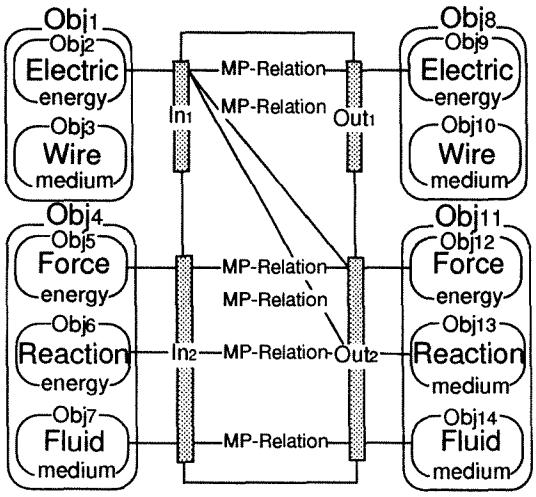


Figure 14: Behavior model of an electric fan and a screw

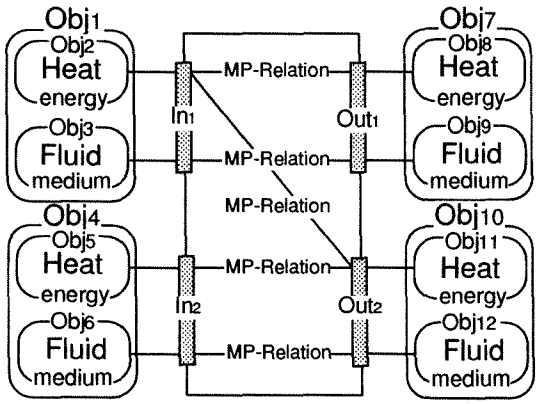


Figure 15: Behavior model of a heat exchanger

of an engine and for heating system of a room.

When the HX is used as a cooling system of an engine, its behavior is interpreted as taking away the heat of the coolant(Obj_1) of engine. This interpretation is based on the idea that the heat energy of the output coolant, Obj_7, is not necessary for the engine, connected to the port Out_1. This interpretation is represented by the predicate,

NoNeed(Out_1,Heat Energy).

On the other hand, when the HX is used as a heater of a room, its behavior is interpreted as heating the air of the room(Obj_4). This interpretation is based on the idea that the heat energy of the output air, Obj_{10} is necessary for the room, connected to the port Out_2. This interpretation is represented by the predicate,

Need(Out_2,Heat Energy).

Some coolers achieve their function as air condition-

ers, for example, by maintaining the temperature of a room, and other coolers achieve their function as coolant systems which,for example, prevent overheat of an integrated circuit. Functional model of the former type of cooler can be represented by attaching the following two predicates to the functional model of the cooler.

Goal:G(Obj_7.Temp = k)
FuncType: ToMaintain

In the same manner, the latter type of coolers are represented by attaching the following predicates.

Goal:G(Obj_7.Temp < k)
FuncType: ToPrevent

Discussion

In his work(de Kleer 1984), J.de Kleer proposes a method to use function of a system, which is derived from function of each component, to decrease ambiguous results of simulation. The method represents a component having more than one causal relation between input and output, where function of the component is decided by selecting one of the relations.

His work enables us to represent the function to achieve desirable state, which is also achieved by our representation method as discussed above. Furthermore, our method can explicitly represent the function which prevents system from falling into no good state and side effects which may damage the whole system.

V.Sembugamoorthy and B.Chandrasekaran(Sembugamoorthy & Chandrasekaran 1986) capture behavior as a series of states of the system and function as to achieve an intended desirable state. The function is achieved by the behavior and the function of each component whose role is defined by the function of the system. Such a framework to represent the function of large systems hierarchically is important.

In the above functional representation (Sembugamoorthy & Chandrasekaran 1986), the function is described as achieving a desirable state. Anne M.Keuneke(Keuneke 1991) classifies the concept of function into four types according to some conditions, for example, the method to achieve the function, the initial condition which raise the function, the length of the term in which effect of the function is hold, and so on. She captures function as attachment of implicit assumptions to a behavior, and employs a policy to seek for the primitives to represent the assumptions. Her policy is close to that of us in this sense. Our representation includes her classification and applies it to produce an abstract explanation of the function of a component and classification of the concepts of trouble.

Her classification of the function is employed in the work by B.Chandrasekaran, et al.(Chandrasekaran, Goel, & Iwasaki 1993) in which a functional representation framework is proposed by extending (Sembugamoorthy & Chandrasekaran 1986), which can be applied to various kinds of tasks especially design and diagnosis.

Because their representation of the function in (Chandrasekaran, Goel, & Iwasaki 1993) depends on the structure of the component of the system, description of the function of two systems differs from each other if their structure differs from each other. Thus their representation method seems to be weak in terms of re-usability of the description of function. On the other hand, our representation of the function of a component can be applied to wide range of components of the same input-output relations.

Y.Iwasaki,et al.(Vescovi *et al.* 1993)(Iwasaki *et al.* 1993) extended the framework for representing function proposed in (Sembugamoorthy & Chandrasekaran 1986) to represent the intention of designers, and proposed a framework for supporting refinement process of design through verification between behavior of a designed artifact and the intention of its designer. The function they captured is close to our concept. However, as the basic idea of functional representation is the same as (Sembugamoorthy & Chandrasekaran 1986), their method also seems to be weak in reusability of the described components.

In order to decrease the cost for diagnosis, A.Abu-Hanna, et al.(Abu-Hanna, Benjamins, & Jansweijer 1991) propose a method to describe functional model of a system at three levels of abstraction.

We also regard abstraction and conceptualization of a component have close relations to the function of the component, since the goal of a component is always abstract one to enable interpretation of its behavior at the knowledge level. Thus we have investigated what primitives contribute to conceptualization of function. As mentioned in former sections, however, the enumeration of the primitives is not exhaustive.

M.Pegah, et al.(Pegah, Sticklen, & Bond 1993) apply the functional representation in (Sembugamoorthy & Chandrasekaran 1986) to F/A-18 aircraft fuel system, one of the large scale and complex systems, to achieve causal understanding of the system.

Our interest in application of this work goes to **KC III**(Kitamura *et al.* 1994), a model-based diagnostic shell which currently deals with a heat transportation system including negative feedbacks among its components whose explanation is difficult.

Conclusion

This paper has proposed a new method and a vocabulary for representing components captured from the viewpoints of behavior and function. Our method does not rely on domain specific terms in representing components at a certain level of abstraction. The method enables model builders to describe a component at various levels of abstraction. Still we have several problems to be discussed about the concept of function as we discussed earlier. They are under discussion and its

result shall be reflected to refine the proposed representation method.

The potential of a functional representation method should be evaluated by not only the number of phenomena the method can describe but also the enhanced quality of a task by application of the model described by the method. Currently we aim at applying our method to the diagnostic shell **KC III**(Kitamura *et al.* 1994) in two ways. One is to support users in model building, and the other is to help users understand the process and result of a diagnosis.

W.Swartout, et al.(Swartout, Paris, & Moore 1991) propose a framework to build Explainable Expert Systems(EES). They regard three design aspects such as justifications of the system's action, explications of general problem-solving strategies, and descriptions of the system's terminology, as important things for producing good explanation in design activity. EES provides explanation from these apsects in a dialogue style. Another investigation by T.R.Gruber and P.O.Gautier(Gruber & Gautier 1993) shows a technique to explain the system's action and the knowledge which the system possesses.

Currently our interest goes to what information should be conveyed by the explanation for the users of **KC III**, and discussion about how can the representation method contribute to the explanation is on progress.

References

Abu-Hanna, A.; Benjamins, R.; and Jansweijer, W. 1991. Device understanding and modeliing for diagnosis. *IEEE Expert* 26–32.

Chandrasekaran, B.; Goel, A.; and Iwasaki, Y. 1993. Functional representation as design rationale. *COMPUTER* 48–56.

de Kleer, J. 1984. How circuits work. *Artificial Intelligence* 24:205–280.

Gruber, T. R., and Gautier, P. O. 1993. Machine-generated explanations of engineering models:a compositional modeling approach. In *Proceedings of the IJCAI*, 1502–1508.

Hirai, T.; Nomura, Y.; Ikeda, M.; and Mizoguchi, R. 1991. The organization of the domain knowledge oriented to sharability - domain-specific tool based on the deep knowledge(**KC II-DST**). In *Proceedings of the 5th Annual Conference of JSAI*, 325–328. Japanese Society for Artificial Intelligence. in Japanese.

Iwasaki, Y.; Fikes, R.; Vescovi, M.; and Chandrasekaran, B. 1993. How things are intended to work: Capturing functional knowledge in device design. In *Proceedings of the IJCAI*, 1516–1522.

Keuneke, A. 1991. Device representation:the significance of functional knowledge. *IEEE Expert* 24:22–25.

Kitamura, Y.; Sasajima, M.; Ikeda, M.; and Mizoguchi, R. 1994. Model building and qualitative reasoning for diagnostic shell. In *Proceedings of the '94 Japan/Korea Joint Conference on Expert Systems*, 41–46. Japanese Society for Artificial Intelligence.

Pegah, M.; Sticklen, J.; and Bond, W. 1993. Functional representation and reasoning about the F/A-18 aircraft fuel system. *IEEE Expert* 24:65–71.

Sembugamoorthy, V., and Chandrasekaran, B. 1986. Functional representation of devices and compilation of diagnostic problem-solving systems. In Kolodner, J., and Riesbeck, C., eds., *Experience, Memory, and Reasoning*. Hillsdale, N.J.: Lawrence Erlbaum Associates. 47–73.

Swartout, W.; Paris, C.; and Moore, J. 1991. Design for explainable expert systems. *IEEE Expert* 58–64.

Vescovi, M.; Iwasaki, Y.; Fikes, R.; and Chandrasekaran, B. 1993. CFRL:A language for specifying the causal functionality of engineered devices. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 626–633.