ICOT Technical Memorandom: TM-0988

TM-0988

An Application of Structural Modeling and Automated Reasoning to Real-Time Systems Design

by

S. Honiden, N. Uchihira,
K. Matsumoto (Toshiba)

December, 1990

The Journal of Real-Time Systems, 1, 313–331 (1990)
© 1990 Kluwer Academic Publishers. Manufactured in The Netherlands.

An Application of Structural Modeling and Automated Reasoning to Real-Time Systems Design

SHINICHI HONIDEN, NAOSHI UCHIHIRA, KAZUNORI MATSUMOTO, KAZUO MATSUMURA AND MASAHIKO ARAI
Systems & Software Engineering Laboratory, Toshiba Corporation,
70 Yanagi-cho, Saiwai-ku, Kawasaki 210, Japan

Abstract. This paper presents an application of structural modeling and automated reasoning as a software development environment for real-time systems. This application satisfies two major requirements for such an environment: (1) to synthesize an absolutely correct program and, (2) to increase software productivity. The real time systems, which consist of concurrent programs, are described by a Prolog based concurrent object-oriented language, called MENDEL/87. As a typical concurrent program consists of two parts: functional and synchronization parts; the functional part in the reusable component to be registered in a library will be generated by a structural modeling through the use of structuring functions with respect to data flows. The synchronization part will be synthesized from temporal logic specifications by the use of an automated reasoning mechanism. This paper also describes the MENDELS ZONE implemented on a Prolog machine, which is the working base for the presented application method.

1. Introduction

As the complexity of real-time systems increases, greater demands will be placed upon a software development environment. Generally, there are two major requirements for the software development environment.

(i) To synthesize an absolutely correct program.
(ii) To increase software productivity.

The real-time systems consist of concurrent programs. A typical concurrent program consists of a functional part and a synchronization part. The functional part is correspondent to the task or process content and the synchronization part indicates the timing and temporal relation among tasks or processes. Therefore, synthesizing a correct program is with respect to function and timing correctness.

Concurrent programs have various difficult inherent problems, such as synchronization and dead-lock problems. Solving these problems becomes major work for programmers, in order to develop correct application software. Methods such as the theorem proving method (Manna and Waldinger 1980), which synthesize the program from formal specifications by automated reasoning, have been presented to solve these problems. However, a common obstacle among these methods is the computational complexity problem, which is proportional to the size of the program to be synthesized. As a result, only toy-level programs can be synthesized efficiently. It is evident that these methods will not satisfy the second requirement of a software development environment, which is to increase software productivity.

	Correctness	High productivity
① Software reuse method	fair	good
② Automated reasoning method	good	bad
① + ②	good	good

Figure 1. Software reuse method and automated reasoning method integration.

Therefore, if the automated reasoning method is to be adopted to synthesize the correct program, it has to be fused with another method, which can contribute to high-software productivity. One such method, known as the software reuse method, has been considered and used in the sequential program domain, such as the business application and several other domains, as one of the most effective methods to increase software productivity (Honiden et al. 1988; Jones 1984). Therefore, one of the candidates for satisfying both requirements for a software development environment is the integration of software reuse method and automated reasoning method, as shown in Figure 1. In order to satisfy both the above mentioned requirements for a software development environment, the authors' approach presents the automated reasoning method to complement the disadvantage of the software reuse method.

The authors' software development environment: the MENDELS ZONE is based on the presented approach and has been developed on ICOT's Prolog machine PSI-II (Honiden et al. 1989). MENDELS ZONE represents the *intelligent* zone for the nucleus language-MENDEL/87, which is a Prolog based concurrent object-oriented language (Honiden et al. 1986; Uchihira et al. 1987). Each reusable component is written in MENDEL/87. Therefore, the OBJECT in MENDEL/87 is the reusable component and the synchronization part is realized by message-passing among OBJECTs. This paper focuses on an application of a structural modeling and automated reasoning method to real-time systems design. First, the structural modeling method is used to generate the MENDEL OBJECT. Then, the automated reasoning method is used to retrieve and interconnect the MENDEL OBJECTs and to synthesize the synchronization part among the OBJECTs.

In this paper, Section 2 describes the concurrent reusable component and its description language MENDEL/87; Section 3 describes an overview of the presented method;

Section 4 describes the MENDELS ZONE behavior using a simple example; and Section 5 includes unique features of this method.

2. Concurrent reusable component

2.1. Software reuse in concurrent program

The reusable component can be regarded as a concurrent processing unit, which is called a task, or a process, in a concurrent program. Individual contents in a concurrent processing unit are accomplished sequentially.

Most real-time systems consist of many concurrent software modules. Software reuse methods have been considered among the most effective methods in increased software productivity and are used in several domains. When adopting the software reuse method in the real-time systems domain, several problems which relate to the retrieval and interconnection reusable components need to be solved.

First, various specification methods, such as keyword, case grammar, and formula have been presented for use in retrieving the reusable component. However, these methods can only retrieve one reusable component using one specification statement at a time. The size of the specification is then proportional to the number of desired reusable components. Therefore, the size of the specifications necessary to retrieve the reusable components must be as small as possible.

Second, a software reuse method does not always synthesize a correct program, even if the reusable components are assured to be correct. This is because real-time systems usually consist of several reusable components for which the interface program among reusable components is required to be imposed by some means according to pertinent applications. This work is comparatively easy in a sequential program, but is extremely difficult in a concurrent program. Each reusable component is a concurrent processing unit in the concurrent program. Therefore, a synchronization problem among reusable components may occur, unless each reusable component has a synchronization part which includes the timing and temporal relation to other components. When registering reusable components in the library, it is possible to impose the synchronization part to a reusable component. However, since synchronization is an important relationship among the reusable components, imposing this in a reusable component during registration causes a decrease in the reusable component generality. If the reusable component generality is low, this means that the reusable component is used only under limited applications. In typical concurrent programs, the interface part, which includes the synchronization part, is realized by a relatively small scale program and is required to be absolutely correct. Therefore, it may be possible to synthesize the synchronization part by the automated reasoning method.

2.2. Role of register and reuse persons

There are several view points regarding software reuse. One view classifies software reuse according to the user. Users who take part in software reuse are classified into two groups: a

USER CLASSIFI-CATION	USER ROLE		SUPPORTING METHOD
Register Person	Generation of reusable components which correspond to a functional part		STRUCTURAL MODELING
Reuse Person	Reuse of the reusable components	Retrieval and interconnection of the functional part	PLANNING
		Synthesis of the synchronization part	THEOREM PROVING

Figure 2. Relationship between users and MENDELS ZONE.

reusable component registration user, who generates and registers new reusable components, and a reusable component user, who retrieves and interconnects the reusable components to satisfy his requirement. In this paper, the former is referred to as a register person, the latter, a reuse person. As mentioned previously, if the register person takes into consideration the synchronization of a component with other reusable components during registration, the generality of this component decreases. Hence, when generating and registering in the library, the register person should not take into account applications in which the component is used together with other components. This means that only the register person should consider and generate a functional part in the reusable component.

On the other hand, the reuse person uses several reusable components to satisfy his requirement. Thus, synchronization among these components, according to pertinent application, must be determined by the reuse person. In other words, the synchronization part among the reusable components should be generated only by the reuse person. As a result, the reusable component to be retrieved and interconnected in a library is correspondent to a functional part. It does not include a synchronization part among the reusable components.

According to the user classification mentioned above, a system which supports the software reuse has two major functions, as shown in Figure 2.

(i) To generate a reusable component, which corresponds to a functional part.
(ii) To reuse a reusable component.

In the authors' approach, in order to implement these functions, a structural modeling method is applied to generate a reusable component. An automated reasoning method is applied to enable reusing of the reusable component. The concept of reusing the reusable component function, that is, the above mentioned function (ii), includes the following two sub-functions.

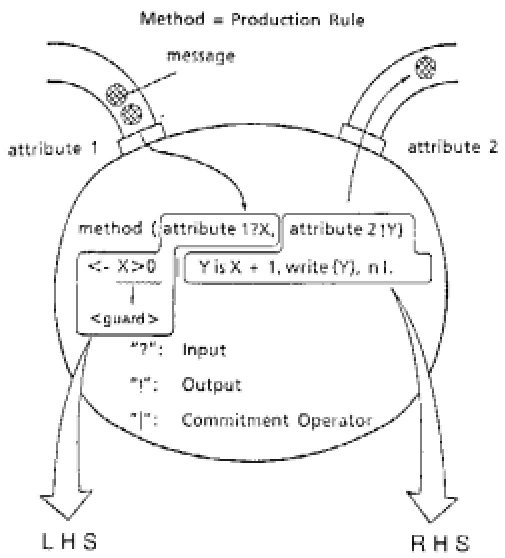


Figure 3. Method part in MENDEL OBJECT.

(ii a) To retrieve and interconnect reusable components, which correspond to the functional part.
(ii-b) To synthesize a synchronization part among reusable components.

This means the automated reasoning method is applied to both sub-functions.

2.3. *MENDEL/87*

Each concurrent reusable component is written in MENDEL/87. In real-time systems, the combination of declarative knowledge description and actor-based object modeling is considered to be one of the effective methods to describe the reusable component. The interrelationships among objects are described by actor-based object modeling and the inner behavior of each object is described by declarative knowledge. The authors adopt MENDEL as the executable specification language to satisfy this requirement. MENDEL is a Prolog-based concurrent object-oriented language.

Since the OBJECT in MENDEL is a concurrent processing unit, the OBJECT can be regarded as a task or a process. Each OBJECT consists of one working-memory and several METHODs, which are declared as follows and shown in Figure 3.

```
METHOD (attribute?variable. . .attribute!variable)
<- <guard> | Prolog clauses.
```

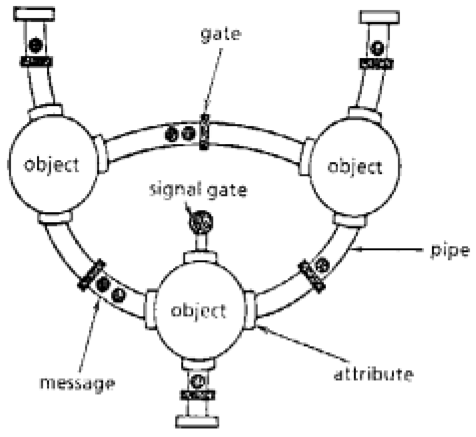


Figure 4. Interconnection among MENDEL OBJECTs.

Each message consists of an attribute name, an input/output identifier—"?" or "!", and a variable name. If a METHOD's variable after an attribute "?" has been received, that METHOD's Prolog clauses are executed. When the METHOD is executed, the variable after an attribute "!" will be unified and sent to the other OBJECTs. Each METHOD is regarded as a production-rule and is used by the forward inference mechanism. A METHOD consists of a left-hand side (LHS) and a right-hand side (RHS). The LHS contains the input messages and the RHS contains the output messages. Both LHS and RHS contain internal state variables which are stored in the working-memory. The body part in a METHOD consists of Prolog clauses. As the Prolog system can be regarded as a backward inference engine, each METHOD includes the backward inference engine. The overall architecture is a distributed production system, in which each OBJECT has inherent working-memory and both a forward and a backward inference mechanism.

Each OBJECT has finite pipe caps and can transmit messages only through the pipe caps, as shown in Figure 4. An attribute is assigned to each pipe cap and is used to identify input/output messages. Messages are transmitted between OBJECTs through the transmission pipe connected to the pipe caps. Each pipe has one gate only, which controls the message stream. In MENDEL/87, a simple synchronization mechanism is achieved by using a METHOD selection mechanism, similar to Dijkstra's guarded command. The OBJECT is suspended, until it receives all required messages. However, the mechanism is so simple that, for a complicated synchronization, it will require complicated pipe interconnection. Therefore, an additional synchronization mechanism, using the gate and the gate controller, is introduced. The gate is used for message stream control. The gate opens, lets only one message pass through and then shuts. These three actions constitute an atomic action of the gate. With no message at the gate, the gate cannot be opened. The gate is similar to the transition in a Petri Net model. The gate controller, which controls all gates, is called the synchronization supervisor.

In MENDELS ZONE, the OBJECT in MENDEL/87 corresponds to a functional part, and message-passing among OBJECTs, which is controlled by the gate controller, corresponds to the synchronization part, as shown in Figure 5.

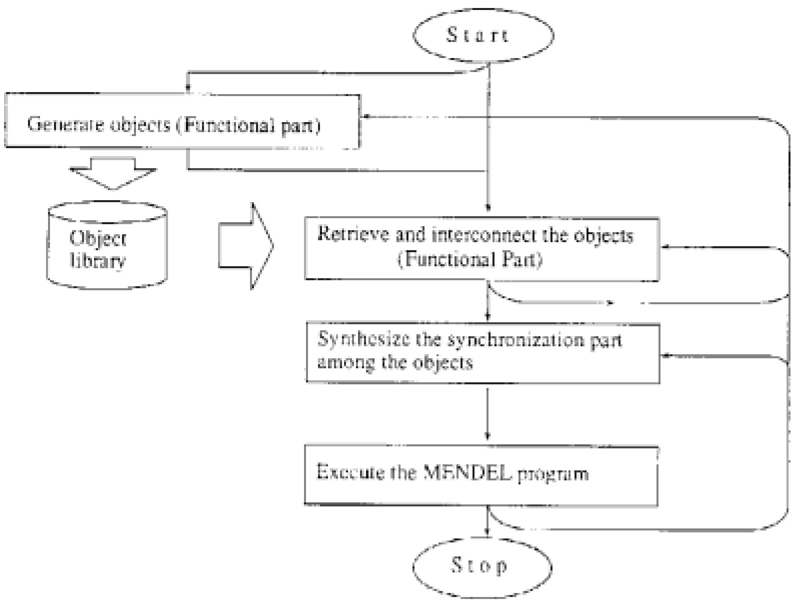

Start
Generate objects (Functional part)
Object library
Retrieve and interconnect the objects (Functional Part)
Synthesize the synchronization part among the objects
Execute the MENDEL program
Stop

Figure 7. MENDELS ZONE process.

3.1. Generation of the functional part in reusable components

The object in the object-oriented modeling is regarded as the conceptual instance, to enable modeling of the function or knowledge included in an actual subject. Such an object should be correspondent to the functional part in a reusable component. Structural modeling methods have been used to obtain functions from software requirements. Hence, such methods can be used to generate a functional part. As the functional part is realized as an OBJECT in MENDEL/87, this section describes the structural modeling method, used to obtain the MENDEL OBJECT candidate.

The structural modeling method, used to obtain functions from the software requirements, is called the SCI (Structuring by Cross-Interaction) method (Arai, Tamura and Mizutani 1981; Matsumura, Mizutani and Arai 1987). SCI is used to extract correct function concepts from the software requirements. The system itself can be regarded as incorporating common concepts between two sets of descriptions. By comparing two elements included in these sets, a relation is defined when there is a common concept in both elements. By pairwise comparisons, edges between the two sets are obtained in this manner. These edges, with the elements included in the sets, form a graph called a bipartite graph. Since an edge represents a common concept, included in the two different viewpoints, all the edges represent the system itself. Therefore, when a set of elements includes all the edges, it describes the system. Sets X and Y in Figure 8 are obvious examples. Set ($X1$, $Y1$) is another example.

concurrent program	concurrent reusable components	MENDEL/87
functional part	reusable components	objects
synchronization part	interfacing among reusable components	message-passing among objects

Figure 5. Functional and synchronization parts in concurrent program.

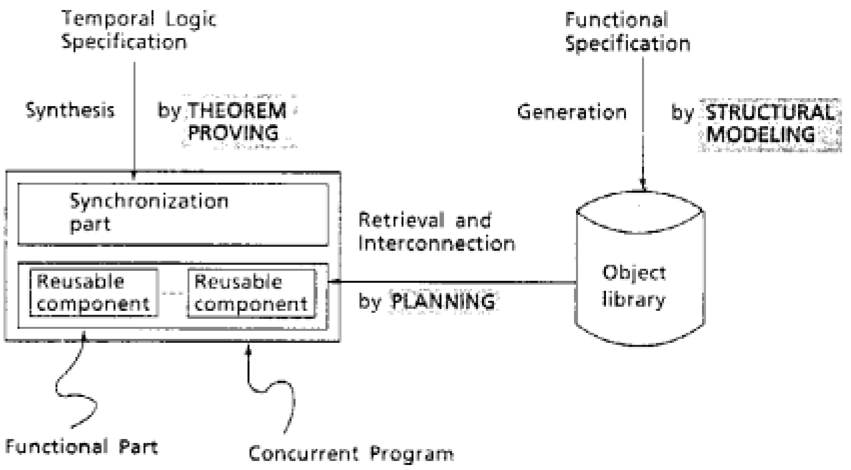


Figure 6. An overview of MENDELS ZONE.

3. Method overview

This section describes an overview of the proposed method which supports the software development environment as shown in Figure 6.

The proposed method consists of four phases, shown in Figure 7, as follows:

Phase 1: Generate a functional part in a reusable component
Phase 2: Retrieve and interconnect reusable components, which correspond to the functional part
Phase 3: Synthesize the synchronization part among reusable components
Phase 4: Execute visually

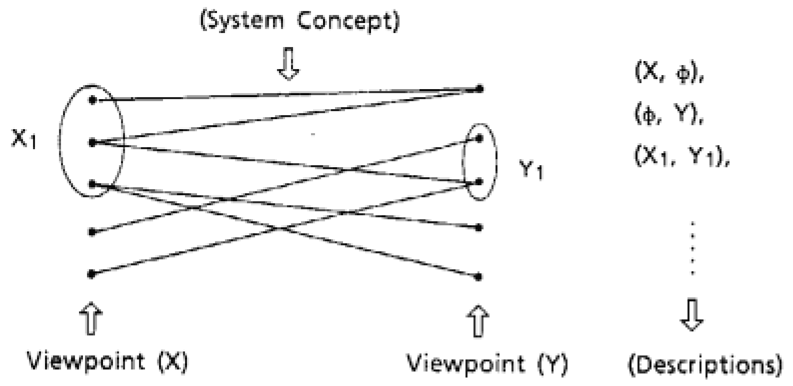


Figure 8. Multi view points.

Function \ Data or state	d1. Shipment request	d2. Cargo order	d3. Inventory master file	d4. Container	d5. Goods in stock	d6. Shortage, etc.	d7. List of goods out of stock	d8. Shipment order	d9. List of empty container No.	d10. Goods to be shipped	d11. Empty container
f1. Acceptance	7	7	9	3	3	7	7	7	7		
f2. Shipment delivery	7		7		3			7			
f3. Empty Container No. output	7		7	3					7		3
f4. Inventory master file Up dating	7		9	3	3		7				
f5. shipment request Response	7		9	3	3	7		7		3	
f6. Cargo order processing		7	9	3	3						
f7. Storage		3		9	9						
f8. Delivery of Container		3		9	9						
f9. Shipment	3			7	7			7		7	
f10. Shipment of goods in stock	3				7			7		7	
f11. Carrying out of empty container	3			7					7		7

9 points ··· ··· Create/update Storage data
7 points ··· ··· Input/output temporaty data
3 points ··· ··· Other relationship

Figure 9. Function-data relationship matrix.

Thus, when two sets, *X* and *Y* are given, there are many sets describing the system. These sets are called *coverings* in graph theory. In this way, a system is described by its corresponding coverings, and a structure can be extracted from the set of coverings.

In SCI, it is possible to find a MENDEL OBJECT candidate by making a structure of *functions* as Sets *X* and *data* as Sets *Y* in the following manner. For example, consider the stocker-OBJECT which accepts several products, holds them, and delivers them by request. In the first step, the functions and data are separated from each view of the requirements specification concerned. The relationship between function and data are defined on the stock. Points are then assigned to the functions and data on the matrix, as shown in Figure 9.

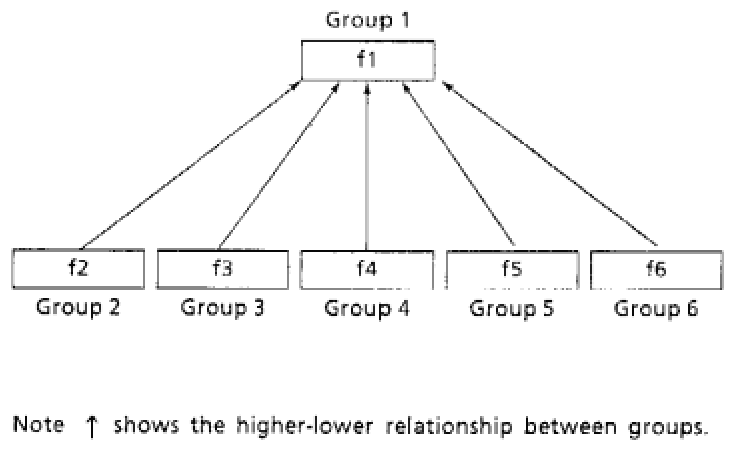


Figure 10. A sample of function structuring by SCI.

For example, *Acceptance* and *Shipment delivery* are typical functions of the stock. *Shipment request* and *Cargo order* are typical data stored in the stock. The matrix numbers show the degree of the relationship between function and data. Then, we find the set of functions and data, based on the relationship information by structural modeling method SCI. In SCI, the functions related to the same data form one group. Next, the group of functions related to many data, including the above mentioned data, is placed above the first group. In this manner, SCI implements the structure among the groups. Figure 10 illustrates a result of structuring by SCI. From this result, it is possible to find the most typical OBJECTs candidate. Figure 11 gives a sample OBJECT candidate, based on the structuring result. The higher group (groupl), being related to more data items than the lower group, becomes a MENDEL OBJECT candidate. The lower group, related to the common data part, is often regarded as a METHOD candidate in the MENDEL OBJECT. In this example, Figure 11 as a whole, shows the candidate OBJECT with five functions as its METHOD. As the result, it is possible to construct a skeleton of the reusable component stored in the library, using the structural modeling method.

3.2. Reusable components retrieval and interconnection

The OBJECT in MENDEL is a concurrent processing unit, which also corresponds to the concurrent reusable component. Each OBJECT has finite pipe caps and can transmit messages only through the pipe caps. In MENDEL/87, binding between pipe caps is accomplished automatically by the *planning* method, which selects the necessary OBJECTs and binds the transmission pipes to create the message passing route from input specifications to output specifications.

Planning carries out the automatic retrieval and interconnection, according to the following principles:

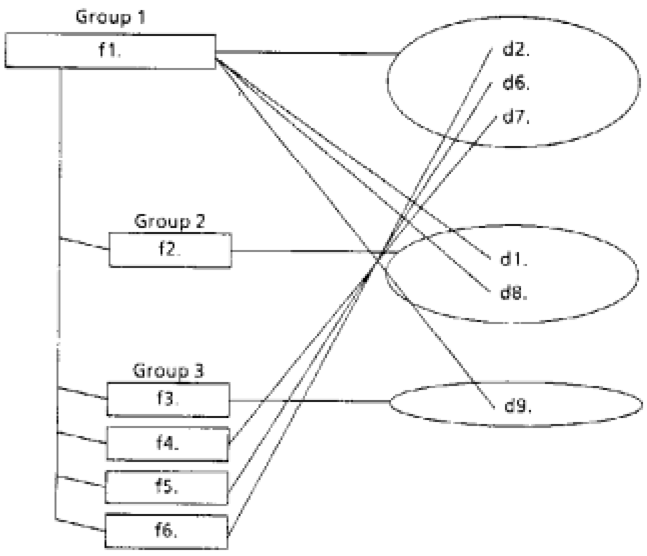


Figure 11. Example of a MENDEL object candidate.

(a) A pair of pipe caps, having the same or similar meaning attributes, can be interconnected.
(b) All required output specifications must be able to reach from given input specifications, through connected OBJECTs and pipes.

In some cases, it might happen that no candidate would be found, or that many candidates would be found. To deal with this problem, the authors adopt a kind of semantic network, which represents the attribute structure and defines a metric to assign priorities to the candidates on the semantic network (Uchihira et al. 1987).

Generally, planning achieves the generation of an action sequence or action program for an agent such as a robot (Nilson 1982). Input for planning includes the initial world, a set of actions which change the world, and the final world. The output from planning forms a sequence of actions, which is an acyclic-directed graph. As each action can be regarded as a reusable component and the world can be as input and output specifications, the sequence of actions is the set of reusable components necessary to satisfy the input and output specifications. Each reusable component has specification itself, named *F-rule* (Nilson 1982). *F-rule* consists of *precondition, add formula,* and *delete list. Precondition* corresponds to an input message into the reusable component, *add formula* corresponds to an output message from the reusable component, and *delete list* includes the input message not present in *add formula.* For the acyclic-directed graph, each node corresponds to a reusable component and each arc corresponds to the data flow between reusable components in the acyclic-directed graph. By using the planning method, the user can retrieve and interconnect several reusable components at one time, by giving input and output specifications only.

MENDEL also introduces a hierarchical planning mechanism, similar to ABSTRIPS (Sacerdoti 1974). In MENDEL, the assignment strategy for *criticality values* to the literals of an F-rule's precondition is based on the design information from the reusable component generation process, by structural modeling method SCI. In SCI, at first, the relationship between functions and data is defined by assigning points which indicate the degree between them. The result from SCI forms the higher group, in which a principal function among several functions, is clearly defined. As each OBJECT identifies principal function, the message associated to principal function can be recognized and the highest value is assigned as criticality value to this message of an F-rule's precondition.

3.3. Synthesis synchronization part

The synchronization part is synthesized from a formal specification using the automated reasoning method. The synchronization part is the scheduler which controls the message passing among OBJECTs. Message passing can actually be controlled by opening or closing a gate. Hence, to synthesize the synchronization part, it is pertinent to give the specification for either opening or closing the gate. This specification is written by RT-PTL (Real-time Propositional Temporal Logic), which is an extension of PTL (Propositional Temporal Logic) (Wolper 1982) for introducing timing constraints. An atomic proposition in RT-PTL will correspond to an atomic action of the gate in MENDEL/87. This means that the statement *proposition g is true for the state* implies *gate g opens, lets only one message pass through, and then shuts at the state.*

Two kinds of constraints are considered in this paper. The first one is timing constraints: maximum, minimum and duration constraints. PTL cannot describe timing constraints. In some cases, it is necessary to give a specification, depending on a situation which changes dynamically. It is difficult to give such specifications in PTL. Therefore, timing constraints are introduced as follows (Dasarathy 1985):

(a) Maximum: no more than t amount of time may elapse between the occurrence of one event and the occurrence of another.
(b) Minimum: no less than t amount of time may elapse between two events.
(c) Duration: an event must occur for t amount of time.

Each of them (called timing constraints statements in this paper) is specified as:

(a) When p occurs then q occurs max $= t$
(b) When p occurs then q occurs min $= t$
(c) While p occurs duration $= t$

Another constraint is temporal-ordering, which specifies the restrictions on ordering gates. This constraint is given by PTL formulas. Four temporal operators are added to the propositional logic (PL) to get the PTL. Its formulas are defined inductively as follows;

(a) Every PL formula is PTL formula.
(b) Next p, always p, eventually p, p until q, p and q, not p are formulas, if p and q are PTL formulas.

Other unusual logical connectives, such as imply, or, equivalent are defined in the usual manner. Each temporal operator has the same meaning, which the English word indicates. Precise definitions of PTL semantics is available in (Wolper 1982).

The decision procedure for RT-PTL is the same as that for PTL, except for the timing constraints treatment. The decision procedure for PTL (Wolper 1982) is an extension of the tableau method which is a kind of theorem proving method. In the tableau method, to decide the satisfiability for the given formula, a model graph, which is a state transition graph, may be constructed. The arc on the state transition graph consists of the event and the action, where the event means the gate-opening and the action means the user-defined processing time for the OBJECT caused by the gate-opening. If this graph cannot be constructed, the given formula is unsatisfiable. A state transition graph is a collection of all models to a given formula. Scheduling rules are translated from this model graph. A state transition on this graph for PTL formula is accomplished by implementing a fairness strategy. In RT-PTL, it is implemented by evaluating timing constraints.

The decision procedure for RT-PTL is as follows.

(i) It constructs a state transition graph, disregarding timing constraints.
(ii) As traversing the state transition graph, it accumulates the user-defined processing time for the OBJECT and checks to meet the timing constraints such as allowable maximum time between two events. If it finds arcs which violate the timing constraints, it deletes or restricts them. As the result, the restricted graph is assured to meet the timing constraints. If the restricted graph becomes empty, the functional part which consists of several OBJECTs cannot meet the timing constraints.

As a result, the RT-PTL description consists of two parts: a set of timing constraint statements and PTL formulas. Next, a simple example is explained.

Example.

There are one producer-OBJECT, two consumer-OBJECTs, and one stocker-OBJECT, which is generated in Section 3.1. A producer-OBJECT sends two products to a stocker-OBJECT ('*make*' in RT-PTL proposition), the stocker-OBJECT holds these products ('make_ok' in RT-PTL proposition), and two consumer-OBJECTS send requests to the stocker-OBJECT for the product ('req1', 'req2' in RT-PTL proposition), as shown in Figure 12. Synchronization requirements are described as follows:

(a) always (make imply (not(req1 or req2) until make_ok))
(b) always (make imply (next eventually (req1 or req2)))
(c) when make occurs then (req1 or req2) occurs max=30
(d) when req1 occurs then req2 occurs min=15
(e) when req2 occurs then req1 occurs min=15
(f) while make occurs duration 10

Statements (a) and (b) are PTL formulas, which indicate the temporal ordering among gates. For example, statement (a) means that 'req1' and 'req2' must wait to be received by stocker-OBJECT, while 'make' message processing. Statements (c), (d), (e), and (f) describe timing

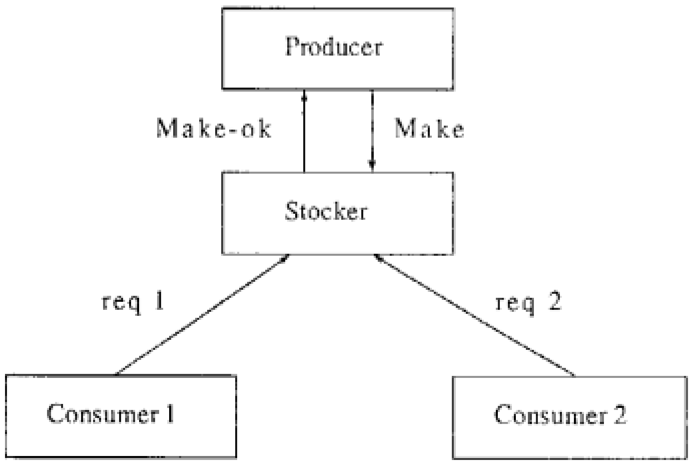


Figure 12. Example: Producer-consumer problem.

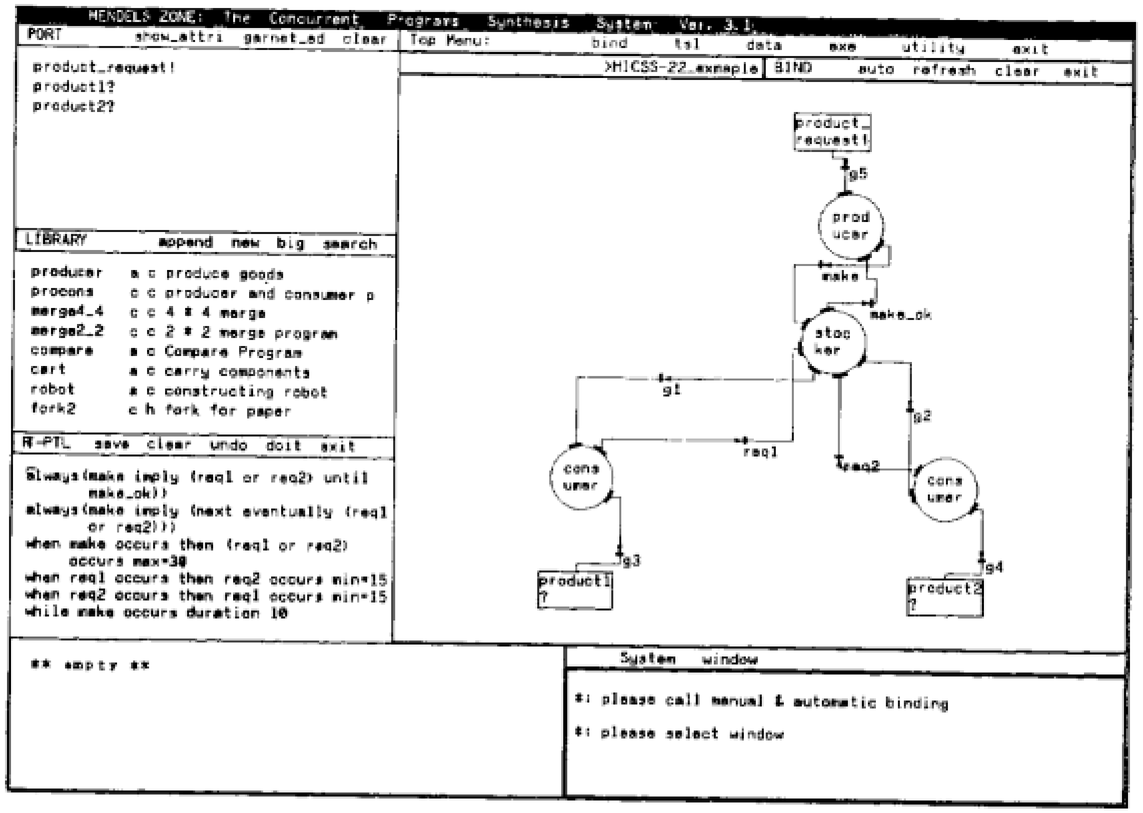
constraints. For example, statement (c) means that no more than 30 amounts of time may elapse between the occurrence of 'make' and the occurrence of 'reql' or 'req2'. Also, statement (c) restricts the number of particular loop on the state transition graph, which was generated from statements (a) and (b), in order to satisfy allowable maximum time between 'make' and 'reql' or 'req2'.

3.4. Visual execution

Execution is pseudo-concurrent on one CPU. The gate is implemented as a mail box. The interpreter selects one of the OBJECTs waiting at the mail box and allows it to receive a message according to the scheduling rules derived in previous steps. Through such a visual execution, the user can determine the system overview, which has been constructed from reusable components and synthesized from RT-PTL specification. This method supports the software prototyping. Therefore, is is possible to return to an arbitrary step which had previously been accomplished, if the system behavior does not match the user's requirement image.

4. MENDELS ZONE: A software development environment for real-time systems

This section describes an overview of MENDELS ZONE using a simple example. MENDELS ZONE is a software development environment for real-time systems, which has been implemented on Prolog machine PSI-II. MENDELS ZONE provides five windows as user interface, as shown in Figure 13:


MENDELS ZONE: The Concurrent Programs Synthesis System Ver. 3.1
PORT show_attri garnet_ed clear
Top Menu: bind tsl data exe utility exit
>HICSS-22_exmaple BIND auto refresh clear exit
LIBRARY append new big search
R-PTL save clear undo doit exit
System window
product_request!
g5
prod ucer
make
make_ok
stoc ker
g1
g2
req1
req2
cons umer
cons umer
g3
g4
product1 ?
product2 ?

```
product_request!
product1?
product2?
```

```
producer    a c produce goods
procons     c c producer and consumer p
merge4_4    c c 4 * 4 merge
merge2_2    c c 2 * 2 merge program
compare     a c Compare Program
cart        a c carry components
robot       a c constructing robot
fork2       c h fork for paper
```

```
always(make imply (req1 or req2) until
      make_ok))
always(make imply (next eventually (req1
      or req2)))
when make occurs then (req1 or req2)
    occurs max=30
when req1 occurs then req2 occurs min=15
when req2 occurs then req1 occurs min=15
while make occurs duration 10
```

```
** empty **
```

```
*: please call manual & automatic binding
*: please select window
```

Figure 13. MMI of MENDELS ZONE.

(a) System window, in which the user inputs his command to MENDELS ZONE and receives the messages from MENDELS ZONE,
(b) Library window, which displays the reusable component, which is MENDEL/87 OBJECT registered in an OBJECT library,
(c) Port window, in which the user determines the input and output specification, which shows functional requirement,
(d) RT-PTL window, in which the user determines the synchronization specifications in RT-PTL,
(e) Diagram window, which shows the retrieved and interconnected reusable components and visual execution.

The authors synthesize the producer-consumer program, which is mentioned in Section 3.3.

Phase 1. In this phase, the OBJECT is generated and registered by the register person. It is assumed that there are various OBJECTs generated by structural modeling and registered in an OBJECT library as mentioned in Section 3.1.

Phase 2. In this phase, the reuse person retrieves and interconnects the OBJECTs to satisfy his requirements, which are written in the input and output specifications. In this example, using the Port window, the reuse person determines the following specifications.

Input specification: product_request
Output specification: productl, product2

In the Diagram window, MENDELS ZONE shows several OBJECTs which have been retrieved and interconnected automatically by planning to satisfy the above specification as shown in Figure 14.

Phase 3. In this phase, the reuse person determines the synchronization part among four OBJECTs, Producer, Stocker, Consumerl, Consumer2, which were retrieved and interconnected in Phase 2. In the RT-PTL window, the reuse person inputs the synchronization specification in RT-PTL. In this case, synchronization requirement is mentioned in Section 3.3. These requirements are written by opening or closing the gate and are described in RT-PTL.

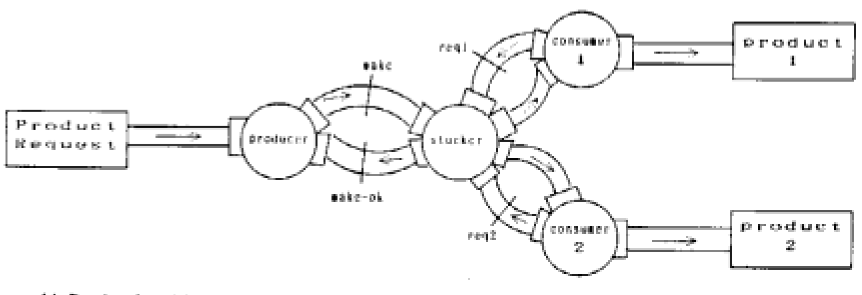


Figure 14. Retrieval and interconnected OBJECTs.

Phase 4. In this phase, visual execution is displayed in the Diagram window. An execution process is displayed in the Diagram window and the System window. Selected gate and activated OBJECT will blink in the Diagram window for easy recognition. The content of the message, going through the gate, is displayed in the System window. An actual system may consist of several hundred reusable components and may have a hierarchical structure. To deal with such a system, the MENDELS ZONE supports the hierarchical design of real-time systems. This means that arbitrary OBJECTs, displayed in the Diagram window, can be folded or unfolded freely, and that folded OBJECTs can be registered as one OBJECT in a library.

5. Related work

This section evaluates the presented method and compares the related work.

First, for MENDEL, the following two discussions are needed, an executable specification language, and a planning method. Various executable specification languages have been presented and used. They are classified into two groups: Operational approach, such as GIST (Diaz-Gonzalez and Urban 1989), PAISLey (Zave 1984), and PSDL (Luqi et al. 1988) and Functional approach, such as MODEL (Prywers 1984) and RPS (Davis, 1982). MENDEL belongs to the operational approach category. The disadvantage in MENDEL is the verification weakness, because only the synchronization part is verified using temporal logic specification. For the combination of actor-model and declarative knowledge representation, one of the languages most similar to MENDEL is Orient 84/K (Ishikawa and Tokoro 1987), which is an object-oriented concurrent programming language. The main difference between MENDEL and Orient 84/K is that Orient 84/K has several parallel control mechanisms as a programming language and does not support timing constraints. For planning, MENDEL's planning ability is the same as ABSTRIP's (Sacerdoti 1974), and the MENDEL limitation includes that for ABSTRIPS.

Lamport introduced the clock variable into his temporal logic to discuss real-time events (Lamport 1983). The value of this clock variable is updated according to the amount of task execution time. If all tasks in the system are serially executed, no problem arises relating this clock variable. However, parallel execution of tasks may invoke wrong update of it. Another approach to the formal verification of real-time systems is to use the first-order language. Jahanian and Mok propose the real-time logic (RTL) based on Presburg arithmetic which is the first-order language containing the natural number and their comparison operators (Jahanian and Mok 1986). In RTL, timing constraints can be verified. As it is known that the decision procedure for RTL formula needs high-computational complexity, applying RTL to large scale real-time system seems quite difficult. On the other hand, RT-PTL is designed for practical use. RT-PTL consists of PTL formula and timing constraints. State trasnsition graph, except for timing constraints, is generated from PTL formulas by theorem proving method and is assured to be correct. Timing constraints are used to restrict the state transition graph to satisfy them. If restricted graph is empty, it implies that functional part cannot meet the timing constraints.

Several software development environments for real-time systems are presented, such as SDL (Orr et al. 1988), STATEMATE (Harel et al. 1988), ENVISAGER (Diaz-Gonzalez and Urban 1989) and SREM (Alford 1985). The most similar system among them is ENVISAGER, in which the behavior of each of the objects, in terms of messages that can be sent or received, is synthesized from temporal logic. The main differences between ENVISAGER and MENDELS ZONE are that ENVISAGER provides several visual functions and adopts ITL (Interval Temporal Logic) which is validated by the simulation method, but does not support software reuse.

6. Conclusions

This paper has presented a software development method for real-time systems and the MENDELS ZONE based on this method. MENDELS ZONE supports generation of the functional part in reusable components, retrieval and interconnection of reusable components, synthesis of the synchronization part among reusable components and visual execution.

The unique features of this method are as follows:

(i) Generation of a functional part in a reusable component, using the structural modeling method;
(ii) Retrieval and interconnection of the functional part in a reusable component and synthesis of the synchronization part among reusable components, using the automated reasoning method.

Acknowledgement

Parts of this work have been supported by the Japanese Fifth Generation Computer Project and its organizing institute ICOT, as a subproject in the *Intelligent* Programming System. The authors would like to thank Ryuzou Hasegawa of ICOT for his encouragement and support. The authors are also grateful to Seiichi Nishijima and Takeshi Kohno of Systems & Software Engineering Laboratory, Toshiba Corporation, for providing essential support.

References

Alford, M. 1985. SREM at the age of eight: The distributed computing design system, *Computer*, 18, 4: 36–46.
Arai, M., S. Tamura, and H. Mizutani. 1981. A method for structural modeling of complex systems. In *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, pp. 676–681.
Cohen, D. 1984. A forward interface engine to aid in understanding specifications. In *Proc. AAAI-84*, pp. 56–60.
Diaz-Gonzalez, J.P. and J.E. Urban. 1989. Prototyping conceptual models of real-time systems: A visual perspective. In *Proc. HICSS-22*.
Dasarathy, B. 1985. Timing constraints of real-time systems: constraints for expressing them, methods of validating them. *IEEE Trans. Software Eng.*, SE-11 80–86.
Davis, A.M. 1982. Rapid prototyping using executable requirements specifications. *ACM SIGSOFT*, 39–44.
Harel, D., H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, and A. Shtul-Trauring. 1988. STATEMENTE: A Working Environment for the Development of Complex Reactive Systems. In *Proc. 10th ICSE*, pp. 396–406.
Honiden, S. et al. 1986. MENDEL: Prolog based concurrent object-oriented language. In *Proc. IEEE Compcon '86*, pp. 230–234.

Honiden, S. et al. 1988. Software prototyping with reusable components. *J. of Information Processing*, 9, 123-129. (Also in, Software Reuse: The State of the Practice. *IEEE tutorial* 1988.)

Honiden, S. et al. 1989. An application of structural modeling and automated reasoning to concurrent program design. *Proc. HICSS-22*.

Ishikawa, Y. and M. Tokoro. 1987. Orient 84/K: An object-oriented concurrent programming language for knowledge system. *Object-Oriented Concurrent Programming* (ed. by Yonezawa and Tokoro). Cambridge, MA: MIT Press.

Jahanian, F. and A.K. Mok. 1986. Safety analysis of timing properties in real-time systems. *IEEE Trans. Software Eng.*, SE-12: pp. 890-904.

Jones, T.C., 1984. Reusability in programming: A survey of the state of the art. *IEEE Trans. Software Eng.*, SE-9: 488-494.

Lamport, L. 1983. What good is temporal logic. In *Proc. IFIP, Inform. Processing*. (R.E. Mason, Ed.) Amsterdam, The Netherlands: North-Holland.

Luqi, et al. 1988. Rapidly prototyping real-time systems. *IEEE Software*, 25-36.

Manna, Z. and R.J. Waldinger. 1980. A deductive approach to program synthesis. *ACM TOPLAS*, 2, 90-121.

Matsumura, K., H. Mizutani, and M. Arai. 1987. An application of structural modeling to software requirements analysis and design. *IEEE Trans. Software Eng.*, SE-13, 461-471.

Nilson, N.J. 1982. *Principles of Artificial Intelligence*, Springer-Verlag.

Orr, R.A., M.T. Norris, R. Tinker, and C.D.V. Rouch. 1988. Tools for real-time systems design. In *Proc 10th ICSE*, pp. 130-137.

Prywers, N.S. 1984. Automatic program generation in distributed cooperative computation. *IEEE Trans. Syst. Man. Cyber.*, 14, 275-286.

Sacerdoti, E.D. 1974. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence* 5: 115-135.

Uchihira, N., T. Kasuya, K. Matsumoto, and S. Honiden. 1987. Concurrent program synthesis with reusable components using temporal logic. In *Proc. IEEE Compsac '87*, pp. 455-464.

Wolper, P. 1982. Synthesis of communicating processes from temporal logic specification. STAN-CS-82-925, Stanford Univ.

Zave, P. 1984. The operational versus the conventional approach to software development. *Comm. ACM*, 27, 2: 104-118.