

# Real-Time Object-Oriented Specification and Verification

Satoshi Yamane

Dept. of computer science, Faculty of science, Shimane University

1060 Nishikawatu ,Matue City , ZIP code 690, JAPAN

E-mail:yamane@cis.shimane-u.ac.jp

## ABSTRACT

It is useful to introduce object-orientation into analysis and design methodology for hard real-time software. But in the former object-oriented analysis and design methodology for hard real-time software, there are some problems as follows.

(1)There is no methodology in order to specify and verify timing conditons.

(2)There is no methodology in order to clarify relations between three models.

In this paper, I propose object-oriented methodology and specification language for hard real-time software as follows.

(1)Timing conditions are formally specified and verified by formal language theory.

(2)Relations between models are verified by mapping object model to other models.

I show this method effective by the example of automobile system.

## 1.Introduction

Hard real-time software such as automobile and airplane, military system consists of many concurrent processes and many states, and timing constraints are strict. It is necessary to model and specify ,verify hard real-time software including timing constraints. There have been many development methods and specification methods such as real-time structured analysis and Petri net, automaton, process algebra, temporal logic, timed Petri net, timed automaton, timed process algebra, real-time temporal logic<sup>1)</sup>. But there are many problems in these methods, such as lack of modeling technique and intuitonic understandability, documentability .

On the other hand, object-orientation is promising in database and information processing because of modeling technique and information hiding, polymorphism, inheritance<sup>2)</sup> . Moreover, in hard real-time software, object-orientation is effective in following points.

(1)encapsulation

As hard real-time software has many processes and states, system specification has huge numbers of states. But as system states are divided into local states of each object, these can be encapsulated. From this point, the large system can be specified.

(2)concurrency

Hard real-time software consists of many concurrent processes. As object is computed by message communication and logically autonomous, object is suitable for concurrency.

As object-oriented analysis/design method for real-time software, there are many methods

such as OMT(Object Modeling Technique)<sup>3)</sup> and Schlaer/Mellor method<sup>4)</sup>, Booch method<sup>5)</sup>, Buhr method<sup>6)</sup>, ROOM(Real-time Object-Oriented Modeling)<sup>7)</sup>, Objectchart<sup>8)</sup>. And there are many object-oriented specification languages such as TROLL<sup>9)</sup>, OBLOG<sup>10)</sup>, TLOOM (Temporal Logic based Object-Oriented Model)<sup>11)</sup>, OS<sup>12)</sup> based on previous methods. There are various problems in these methods and specification languages as follows.

- (1) There is no methodology in order to explicitly specify and verify timing conditions.
- (2) There is no methodology in order to clarify relations between object model and dynamic model, functional model.
- (3) There is no specification language in order to completely specify object model such as generalization, inheritance and dynamic model including timing constraints.

In general, in software development, (a)theory, (b)specification language, (c)tool to support it, are important. From this point, I propose object-oriented method for hard real-time software in order to solve previous problems. The proposed method is based on OMT because OMT is most mature. The method has following feature.

- (1) To formally specify and verify timing specification  
This notion has never been included in former methods.
  - (a) The theory is that timing specification and verification are based on timed automaton and language inclusion problem<sup>13)</sup>.
  - (b) Specification language is based on timed statechart(afterward defined)<sup>14)</sup>.
  - (c) The tool inputs timed statechart and verification property, and outputs whether verification property is satisfied or not.
- (2) To clarify relations between object model and dynamic model, functional model  
A part of this notion has been included in OMT.
  - (a) The theory is that associations of object model give mapping relations to other models. For examples, aggregation means concurrency of dynamic model and functional model.
  - (b) Relations between three models are described in specification language.
  - (c) The tool checks consistency of model relations.

The structure of the rest of the paper is as follows. The specification language is proposed in Section 2. The timing verification method is proposed in Section 3. The method is shown effective by automobile example and implemented tools in Section 4. The conclusion is in Section 5.

## 2. Real-time object-oriented specification language

This section explains real-time object-oriented specification language. This section assumes that class is equal to object and object has no creation /destruction.

### 2.1 Real-time object-oriented specification technique



Real-time object-oriented specification consists of object model and dynamic model, functional model based on OMT as follows.

(1) object model

Object model represents object and its association. Object's definition consists of name and attributes, operations, and associations, which consists of inheritance and aggregation, refer/use.

(2) dynamic model

Dynamic model represents object behaviors by timed statechart, and consists of states and events, timing constraints, functions. Each dynamic model keeps balances of events all over the system.

(3) functional model

Functional model represents functions by dataflow diagram, and consists of dataflows and functions, stores. Each functional model keeps balances of data all over the system.

(4) relations between three models

The relations are defined according to associations as follows.

(a) Case aggregation

Aggregation maps dynamic model into concurrent statechart, and maps functional model into concurrent program.

(b) Case inheritance

Inheritance maps dynamic model into hierarchical statechart, and maps functional model into differential program.

(c) Case refer/use

Refer/use maps dynamic model into sequential statechart, and maps functional model into sequential program.

(5) other relations

Attributes and operations of object model depend on following factors of dynamic model and functional model.

(a) Attributes correspond to states of dynamic model and stores of functional model.

(b) Operations correspond to functions of dynamic model and functional model.

Modeling process is based on OMT, and repeats following (1)~(6).

(1) Definition of specification scope

This phase corresponds to context of structured analysis. Specification scope and interfaces between external entities are defined.

(2) Definition of object model

Objects are extracted from problem definition. Names and attributes, operations, associations are defined.

(3) Definition of dynamic model

Define behavior and state transition, timing constraint. Events between objects correspond

to associations.

(4)Definition of functional model

Define dataflow and function. Data between objects correspond to associations.

(5)Verification of model relations

Verify whether relations between three models correspond to associations such as inheritance, aggregation, refer/use or not.

(6)Verification of timing constraints

Verify liveness and safety, which include time responsiveness.

## 2.2 Syntax of object-oriented specification language

Syntax of real-time object-oriented specification language consists of twelve definition parts such as class name and attribute, operation, etc. Syntax is defined in Fig.1. In Fig.1, State transition definition part (9) has notable feature as follows.

(9)State transition definition part

Dynamic model is based on timed statechart combining timed Buchi automaton and statechart. Because timed Buchi automaton has simple acceptance condition.

Here, formally define timed statechart.

[Definition 1]( timed statechart)

Specification language consists of eight tuples  $(\Sigma, S, S_0, C, E, F, \rho, \psi)$ .

where

$\Sigma$  : a finite set of events

$S$  : a finite set of states

$S_0 \subseteq S$  : a finite set of start states

$C$  : a finite set of clocks

$E \subseteq 2^S \times 2^S \times \Sigma \times 2^C \times \Phi(C)$  : a set of transitions

$F \subseteq S$  : a set of accepting states

$\rho : S \rightarrow 2^S$  : hierarchical function.  $\rho$  determines substates of each state.

$\psi : S \rightarrow \{\text{AND}, \text{OR}\}$  : type function.  $\psi$  determines type of each state.

$\Phi(C)$  represents timing constraints  $\delta$  of clock  $C$ , and is recursively defined by a set  $X$  of clocks and a time constant  $D$  as follows.

$$\delta := X < D \mid D < X \mid \neg \delta \mid \delta_1 \wedge \delta_2$$

A run  $r$  is accepting if some state from the set  $F$  repeats infinitely often along  $r$ . If  $\psi(S) = \text{AND}$ ,  $\rho(S)$  is AND-decomposition(concurrency). If  $\psi(S) = \text{OR}$ ,  $\rho(S)$  is OR-decomposition(hierarchy).  $\square$



specification language:= {object definition part}

object definition part:=Class name definition part ; (1)  
Attribute definition part ; (2)  
Operation definition part ; (3)  
Association definition part ; (4)  
[ State definition part ; (5)  
Event definition part ; (6)  
Initial state definition part ; (7)  
Acceptance state definition part ; (8)  
State transition definition part ; (9)  
Data definition part ; (10)  
Store definition part ; (11)  
Function definition part ; ] (12)  
Classend ;

(1)Class name definition part :=Class class name ;  
(2)Attribute definition part :=Attribute attribute name : type definition  
{, attribute name : type definition} ;  
where type definition :=integer | real | bool | char | set(a:b) | range(a:b)  
(3)Operation definition part :=Operation operation name {, operation name} ;  
(4)Association definition part :=Association association : association name [attribute] (class name)  
{,association : association name [attribute] (class name)} ;  
where association :=aggregation | inheritance | refer/use  
(5)State definition part :=State state name {,state name} ;  
(6)Event definition part :=Event event name : type definition [(class name)]  
{,event name : type definition [(class name)] } ;  
where class name which inputs/outputs events  
(7)Initial state definition part :=state name {,state name} ;  
(8)Acceptance state definition part:=state name {,state name} ;  
(9)State transition definition part:=Transition state name→event, timing constraint/function→next state name  
{,state name→event, timing constraint/function→next state name} ;  
where timing constraint  $\delta$  := $x < d$  |  $x > d$  |  $\neg \delta$  |  $\delta 1 \wedge \delta 2$  |  $x:=0$   
x : clock variable d : time constant  
(10)Data definition part :=Dataflow data name:type definiton [(class name)]  
{,data name:type definiton [(class name)] } ;  
(11)Store definition part :=Store store name : type definition [(class name)]  
{,store name : type definition [(class name)] } ;  
(12)Function definition part :=Function data name | store name {, data name | store name}  
→function→data name | store name {, data name | store name}  
{ ,data name | store name {, data name | store name}  
→function→data name | store name {, data name | store name} } ;

where  $\square$  and  $\{ \}$  , | are BNF notation

Fig.1 Definition of syntax of specification language

## 2.3 Semantics of object-oriented specification language

As this specification language has two syntax feature of object-orientation and timed statechart, define semantics from two feature. The former(static semantics) is mapping relation from object model to dynamic model and functional model. The latter(dynamic semantics) is dynamic property of real-time object.

### 2.3.1 static semantics

This semantics is mapping relations in Fig.2. Fig.2 is described as follows.

[Definition 2](Aggregation )

Syntax definition is defined as follows.

Class  $O_1$  ;

Association aggregation :  $O_2, O_3$  ;

Here as  $\psi(S_1)=AND$ ,  $\rho(S_1)=\{S_2, S_3\}$ .

$S_1=S_2 \parallel S_3$

In dynamic model,  $S_1$  is concurrent statechart consisting of  $S_2$  and  $S_3$ .

In functional model,

Function  $F_1=F_2 \parallel F_3$

In functional model,  $F_1$  is concurrent program consisting of  $F_2$  and  $F_3$ .

Here

$\parallel$  : concurrent operator

$S_i$  : statechart of class  $O_i$

$F_i$  : function of class  $O_i$  ( $i=1,2,3$ )  $\square$

[Definition 3](Inheritance)

Syntax definition is defined as follows.

Class  $O_1$  ;

Association inheritanc :  $O_2, O_3$  ;

Here as  $\psi(S_1)=OR$ ,  $\rho(S_1)=\{S_2, S_3\}$ .

$S_1=S_2 \text{ OR } S_3$

In dynamic model,  $S_1$  is hierachical statechart consisting of  $S_2$  and  $S_3$ .

In functional model,

Function  $O_2=F_2 \triangle F_1$  ;

Function  $O_3=F_3 \triangle F_1$  ;

In functional model, Function  $O_2$  consists of  $F_2$  and  $F_1$ , and  $F_2$  is differencial program .

Here

$\triangle$  : differencial operator  $\square$

[Definition 4](refer/use)

Syntax definition is defined as follows.

Class  $O_2$  ;

Association refer/use :  $O_1, O_3$  ;

Here following specification can be described.

Class  $O_1$  ;

Association refer/use :  $O_2$  ;

Event  $I_1$  : bool( $O_2$ ) ;

or

Dataflow  $D_1$  : interger( $O_2$ ) ;

(store)

and

Class  $O_3$  ;

Association refer/use :  $O_2$  ;

Event  $I_3$  : bool( $O_2$ ) ;

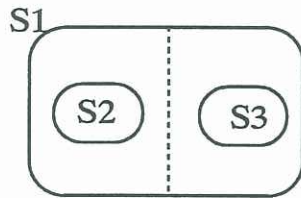
or

Dataflow  $D_3$  : interger( $O_2$ ) ;

(store)  $\square$

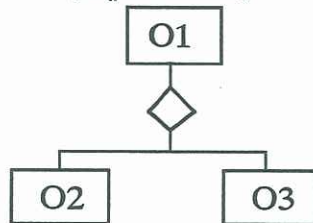
(1) Case aggregation

«dynamic model»



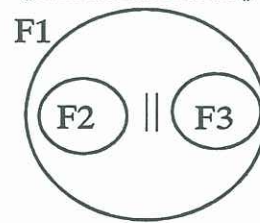
(a) concurrent statechart

«object model»



(b) aggregation structure

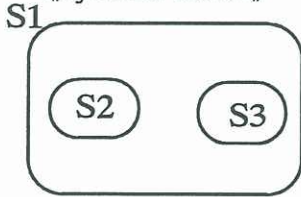
«functional model»



(c) concurrent program

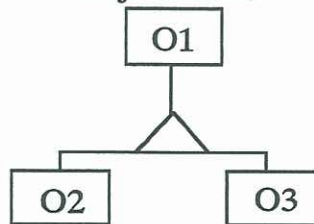
(2) Case inheritance

«dynamic model»



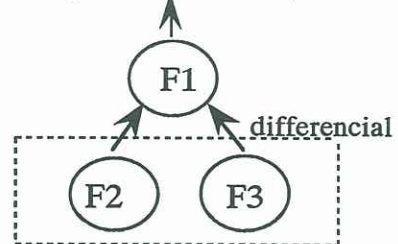
(a) hierarchical statechart

«object model»



(b) inheritance structure

«functional model»



(c) differential program

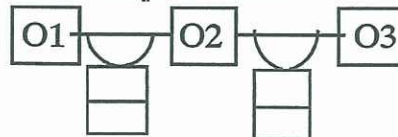
(3) Case refer/use

«dynamic model»



(a) sequential statechart

«object model»



(b) refer/use structure

«functional model»



(c) sequential program

Fig.2 relations of models



## 2.3.2 dynamic semantics

This semantics defines timing property of internal and collaboration object behavior. Fundamentally timed trace theory, which is extended by message passing, is applied to dynamic semantics. Each basic notion is as follows.

(1) Timed trace theory is based on trace theory of CSP<sup>15)</sup> extended by timed traces. I associate a set of observable timed events with each object.

(2) Computational behavior is defined by sending and receiving messages to objects.

[Definition 5](dynamic semantics)

Dynamic semantics of real-time object is defined by following trace.

<sending object, message, receiving object>

Message consists of  $(\rho_i, \tau_i)$ .

Here

$\rho_i$  : finite or infinite set of events

$\tau_i$  : time sequence

When sending receiving object is external environment, define as follows.

Env : external environment  $\square$

## 3. Timing verification method

This section explains timing verification method of dynamic model. In verification method, verification property specification language and verification algorithm are important. In this paper, formal verification method is based on language inclusion problem.

### 3.1 Verification property specification language

In order to unify both specification language and verification property specification language, I decide that verification property specification is described in timed automaton. In formal language theory, there is language inclusion algorithm as verification algorithm. And verification property specification language is described in deterministic timed Muller automaton. Because deterministic timed Muller automaton is more expressive than deterministic timed Buchi automaton and closed under complementation.

[Definition 6](deterministic timed Muller automaton(DTMA))

Deterministic timed Muller automaton consists of six tuples  $(\Sigma, S, S_0, C, E, F)$ .

where

$\Sigma$  : a finite set of events

$S$  : a finite set of states

$S_0 \subseteq S$  : a finite set of start states

$C$  : a finite set of clocks

$E \subseteq 2^S \times 2^S \times \Sigma \times 2^C \times \Phi(C)$  : a set of transitions



$F \subseteq 2^S$  : an acceptance family

It has only one start state,  $|S_0| = 1$ .

$\Phi(C)$  represents timing constraints  $\delta$  of clock  $C$ , and is recursively defined by a set  $X$  of clocks and a time constant  $D$  as follows.

$$\delta := X < D \mid D < X \mid \neg \delta \mid \delta_1 \wedge \delta_2$$

A run  $r$  is accepting if a run  $r \in F$ .  $\square$

### 3.2 Verification algorithm

If system specification and verification specification are described in automaton, verification problem reduces to language inclusion problem<sup>13)</sup>. The notion of verification is shown in Fig.3.

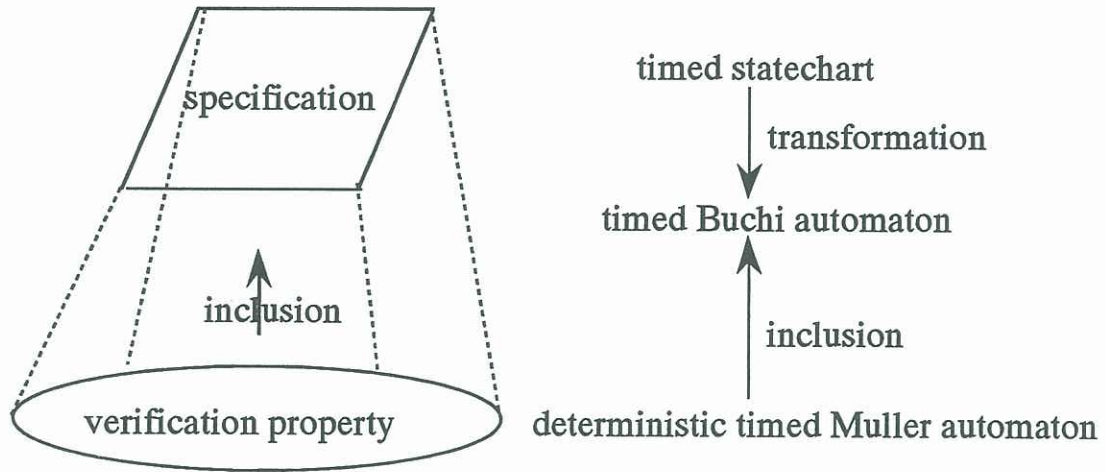


Fig.3 verification concept

[Defuntion 7](language inclusion problem)

The definition of language inclusion problem is as follows.

$$L(M_1) \subseteq L(M_2)$$

$$L(M_1) \subseteq L(M_2) \text{ is equal to } L(M_1) \cap \overline{L(M_2)} = \phi.$$

here

$M_1$  : timed automaton of system specification

$L(M_1)$  : language accepted by  $M_1$

$M_2$  : timed automaton of verification specification

$L(M_2)$  : language accepted by  $M_2$

The complement of  $L(M_2)$  should berecognized by DTMA.  $\square$

Firstly, it is necessary to convert timed statechart into timed Buchi automaton.

[Defuntion 8](to convert timed statechart into timed Buchi automaton)

Timed statechart is AND/OR tree, which has unique root. In order to convert timed statechart into timed Buchi automaton, following operation are repeated from root to bottom.

(1)If  $\phi(S) = \text{AND}$  at  $i$  level, produce intersecting automaton  $\rho(S)$  at  $i+1$  level.

(2) If  $\phi(S) = \text{OR}$  at  $i$  level,  $S$  is converted to  $\rho(S)$  at  $i+1$  level.  $\square$

[Defunition 9](verification algorithm)

The verification algorithm of language inclusion problem  $L(M_1) \cap L(M_2) = \phi$  reduces to checking emptiness of  $L(M_1) \cap L(M_2)$ . The verification algorithm consists of following procedures.

(1) constructing the product automaton  $L(M_1) \cap L(M_2)$

(2) searching for a cycle meeting all the desired acceptance conditions by Tarjan' depth-first search algorithm

(3) checking timing conditions of cycles by geometric region method<sup>16,17,18)</sup>

(4) If cycles satisfy timing conditions, verification property is not satisfied because of

$L(M_1) \cap L(M_2) \neq \phi$ . If cycles do not satisfy them, verification property is satisfied.  $\square$

Here, I explain geometric region.

[Defunition 10](geometric region method)

Geometric region method consists of following procedures.

(1) constructing DBM(differences bound matrix) per each state

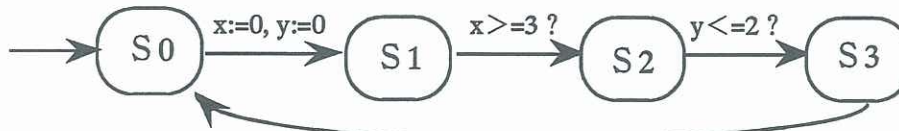
(2) constructing canonical DBM by Floyd-Warshall algorithm

(3) analyzing forward reachability of state transitions by the intersection of DBMs

(4) If there is a negative-cost cycle, there is no state transition because of unsatisfiable timing conditions. If there is no negative-cost cycle, there is state transitions.

Example of geometric region method is shown in Fig. 4.  $\square$





(1) Timed automaton

At S0, from  $x=y=0$

* 0 0		0 0 0
DBM D1 = 0 * *	canonical form DBM =	0 0 0
0 * *		0 0 0

At S1, from  $x=y \geq 0$ .

* 0 0		* 0 0
DBM D2 = * * 0	canonical form DBM =	* 0 0
* 0 *		* 0 0

At S2, from  $x=y \geq 3$ .

*-3-3		*-3-3
DBM D3 = * * 0	canonical form DBM =	* 0 0
* 0 *		* 0 0

At S3, from  $x=y \leq 2$ .

* * *		* * *
DBM D4 = 2 * 0	canonical form DBM =	2 0 0
2 0 *		2 0 0

At  $S2 \rightarrow S3$ , from  $x=y \leq 2$  after  $x=y \geq 3$ .

*-3-3		-5-8-11
intersection $D3 \cap D4$ DBM = 2 * 0	canonical form DBM =	-3-6-9
2 0 *		-6-9-12

In this case, as there is a negative-cost cycle,  $D3 \rightarrow D4$  is impossible.

where

\* : nopath

(2) Example of DBM

Fig.4 Example of geometric region method

## 4. Example of specification and verification

### 4.1 explanation of problem

Automobile consists of accel and engine, transmission, brake shown in Fig.5. Automobile communicates with driver and external environment. Automobile is an aggregation of accel and engine etc. Brake is inherited to normal brake and ABS(Antilock Brake System). Normal brake is operated by driver ,and ABS is computer-controlled in order not to lock wheels. Both normal brake and ABS control oil pressure and operate brake.

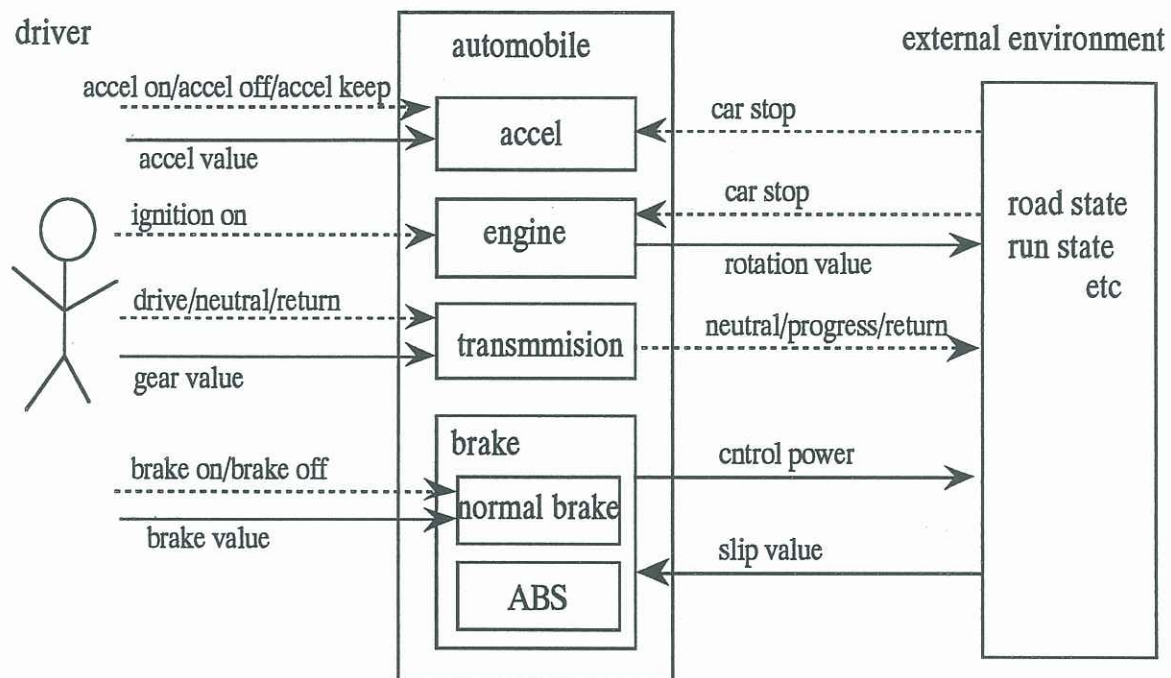


Fig.5 Example of automobile system



## 4.2 example of analysis and specification

Automobile consists of accel and engine, transmission, brake, and each is modeled as object. Typical event sequence between objects is shown in Fig.6.

Each model is specified as follows.

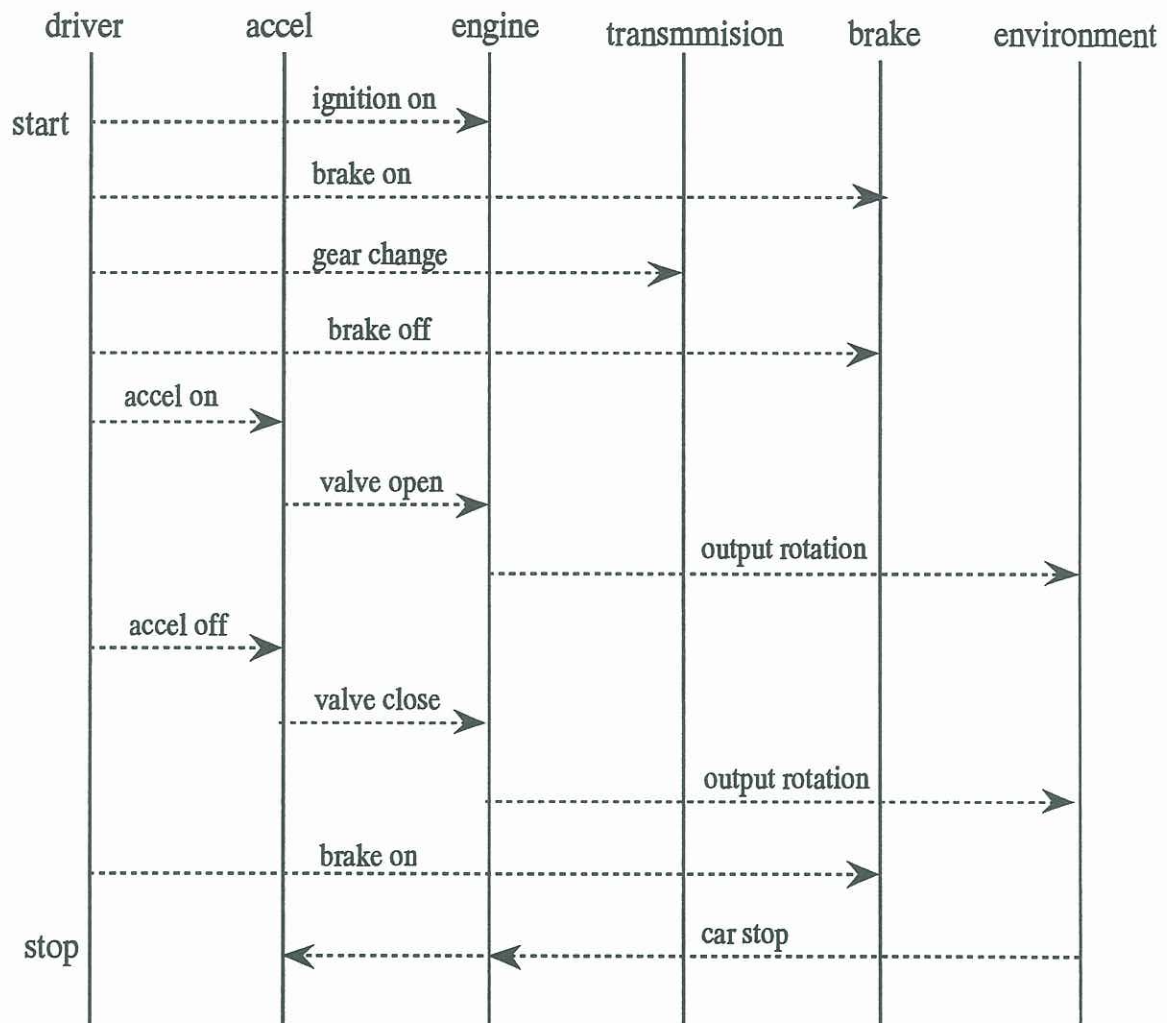


Fig.6 Event sequence diagram

## 4.2.1 object model

Object model has following feature as shown in Fig.7.

- (1) System is an aggregation of accel and engine, transmission, brake.
- (2) Normal brake and ABS are an inheritance of brake.
- (3) Accel and engine are combined by refer/use association of 「control valve」.

Transmmision and brake are combined by refer/use association of 「admit gear R」. Data and events are communicated by associations.

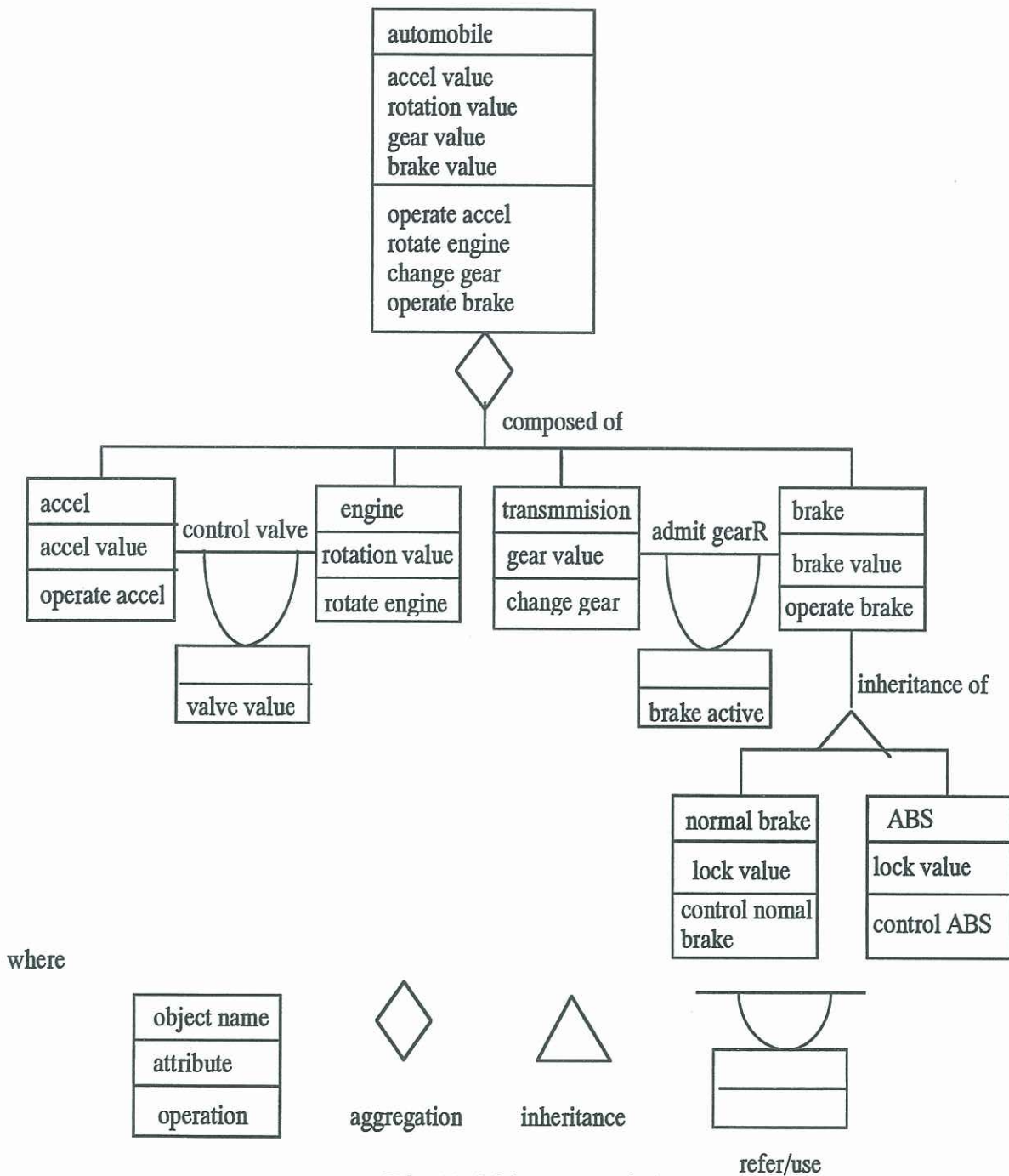


Fig.7 Object model



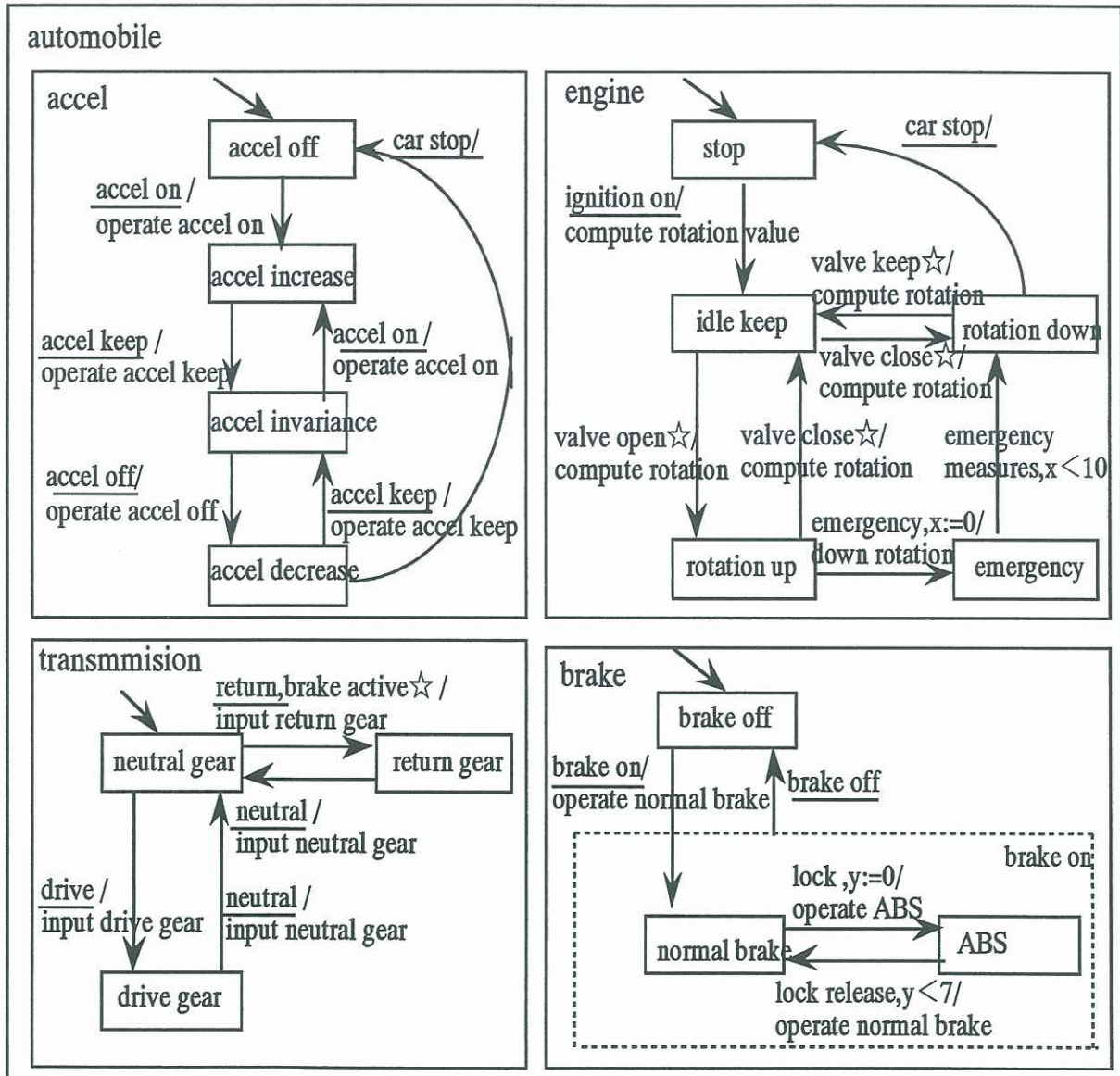
## 4.2.2 dynamic model

Dynamic model has following feature.

- (1) Concurrent statechart consists of accel and engine, transmission, brake.
- (2) Hierarchical statechart consists of normal brake and ABS.
- (3) Transmission and brake communicate by 「 brake active 」 event.

This statechart is timed statechart consisting of concurrent and hierarchical states based on timed  $\omega$ -language.

In order to make collaboration objects clear, underline is added to external event and star is added to event between objects, no symbol is added to internal event. As soon as brake locks, ABS operates. But if ABS operates for more 7 time, ABS has trouble. Dynamic model is shown in Fig.8.



where

external event : event name

event from another object : event name ☆

internal event : event name

Fig.8 Dynamic model

### 4.2.3 functional model

Functional model is shown in Fig.9 and has following feature.

(1)Accel and engine, transmission, brake concurrently behave.

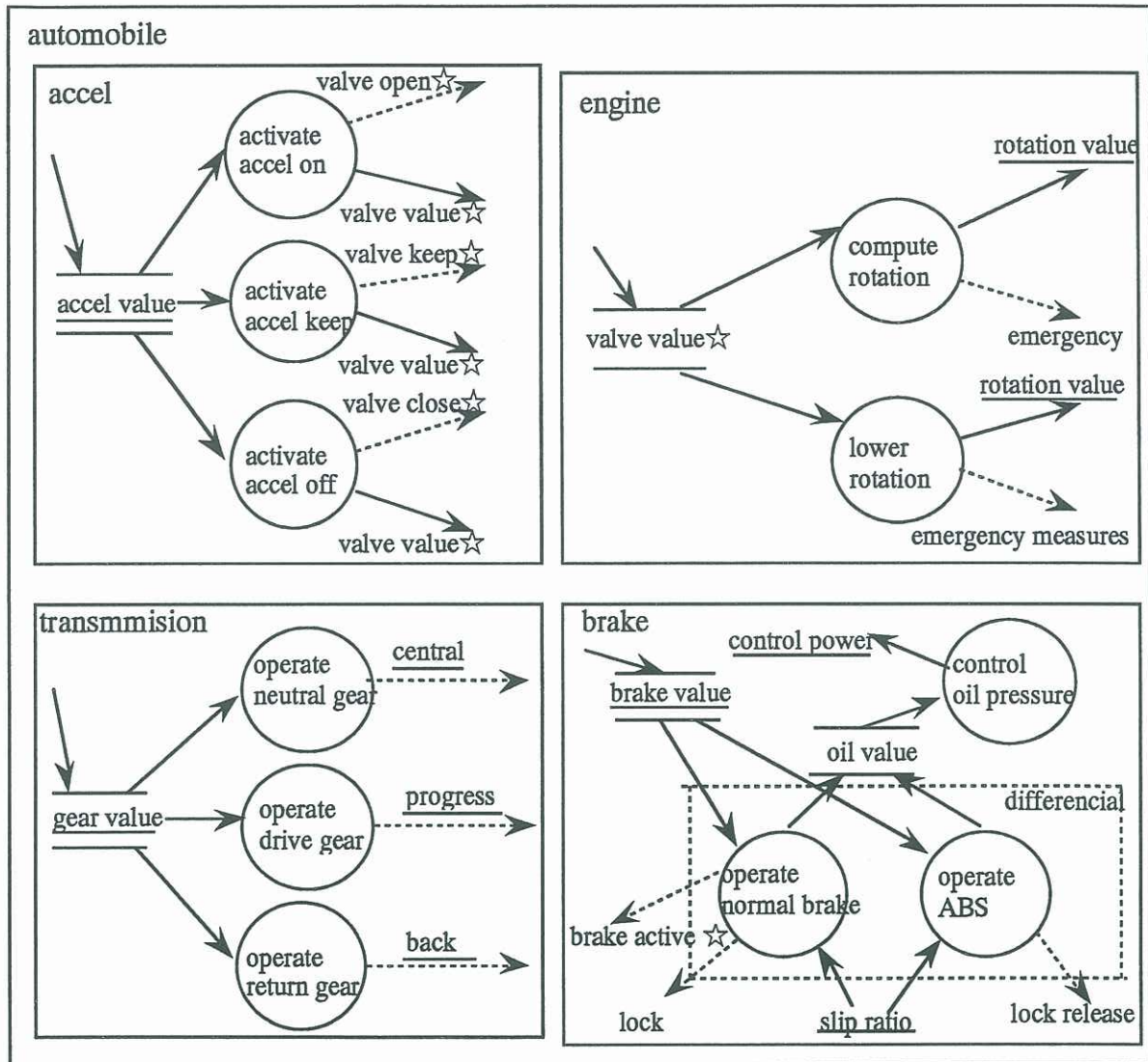
(2)Normal brake and ABS are described as differential program for inheritance.

In functional model, underline and star are added to data and event in order to make control scope clear. Like real-time structured analysis <sup>19)</sup>, following rule makes computational semantics clear.

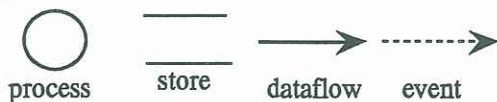
(1)Active process continues to execute during state transition.

(2)As process continues to be activated, data and event continue to flow.

(3)Process can not transact event, but process outputs event by data condition.



where



external event and data : event name or data name

event from or to another object : event name☆ or data name☆

internal event : event name or data name

Fig.9 Functional model



## 4.2.4 the whole model

Static semantics and underline, star symbol make behaviors clear. Graphic representation of Fig.7~9 can be converted into textual representation of Fig.10.

```
Class automobile ;
Attribute
  accel value : real ,
  rotation value : integer,
  gear value : set(N,R,D),
  brake value : real ;
Operation
  operate accel,
  rotate engine,
  change gear,
  operate brake;
Aggregation
  aggregation accel, engine, transmmision, brake;
Classend;

(1)automobile

Class accel ;
Attribute
  accel value : real ,
Operation
  operate accel,
Aggregation
  refer/use : control valve[valve value](engine) ;
State
  accel off, accel increase, accel invariance, accel decrease ;
Event
  accel on : bool(environment),
  accel keep : bool(environment),
  accel off : bool(environment),
  car stop : bool(environment) ;
Initial
  accel off ;
Acceptance
  accel off ;
  accel increase ;
  accel invariance ;
  accel decrease ;
Transition
  accel off→accel on/operate accel on→accel increase ;
  accel increase→accel keep/operate accel keep→accel invariance ;
  accel invariance→accel on/operate accel on→accel increase ;
  accel invariance→accel off/operate accel off→accel decrease ;
  accel decrease→accel keep/operate accel keep→accel invariance ;
  accel decrease→car stop/ ;
Dataflow
  valve value : real(engine) ,
  valve open : bool(engine) ,
  valve keep : bool(engine) ,
  valve close : bool(engine) ;
Store
  accel value : real (environment) ;
Function
  accel value→operate accel on→valve open, valve ovalue,
  accel value→operate accel keep→valve keep, valve ovalue,
  accel value→operate accel off→valve off, valve ovalue,
Classend;

(2)accel
```

Fig.10 Textual representation of specification language

Static semantics is shown in Fig.11. Main relations of models are as follows.

- (1) Aggregation and inheritance, refer/use are mapped into concurrent and hierarchical statechart, concurrent and differential program.
- (2) Attribute of object model is mapped into state variable and store, dataflow.
- (3) Operation of object model is mapped into function.

Verification of consistency check between static semantics and object model is realized by verification program of 300 lines.

Collaboration of objects is represented by global diagram shown in Fig.12. Global diagram visualizes objects and external interface, object interface.

#### 《dynamic model》

- (1) a set of automobile states = a set of accel states || a set of engine states  
|| a set of transmmision states || a set of brake states

where

|| : concurrent operator

- (2)  $\phi$  (brake on) = OR  
 $\rho$  (brake on) = {normal brake, ABS}

where

$\phi : S \rightarrow \{AND, OR\}$  , type function

$\rho : S \rightarrow \text{powerset}(S)$  , hierarchical function

#### 《functional model》

- (1) a set of automobile functions = a set of accel functions || a set of engine functions  
|| a set of transmmision functions || a set of brak functions

- (2) normal brake = operate normal brake  $\triangle$  control oil pressure  
ABS = operate ABS  $\triangle$  control oil pressure

where

$\triangle$  : defferencial operator

Fig.11 Example of static semantics specification

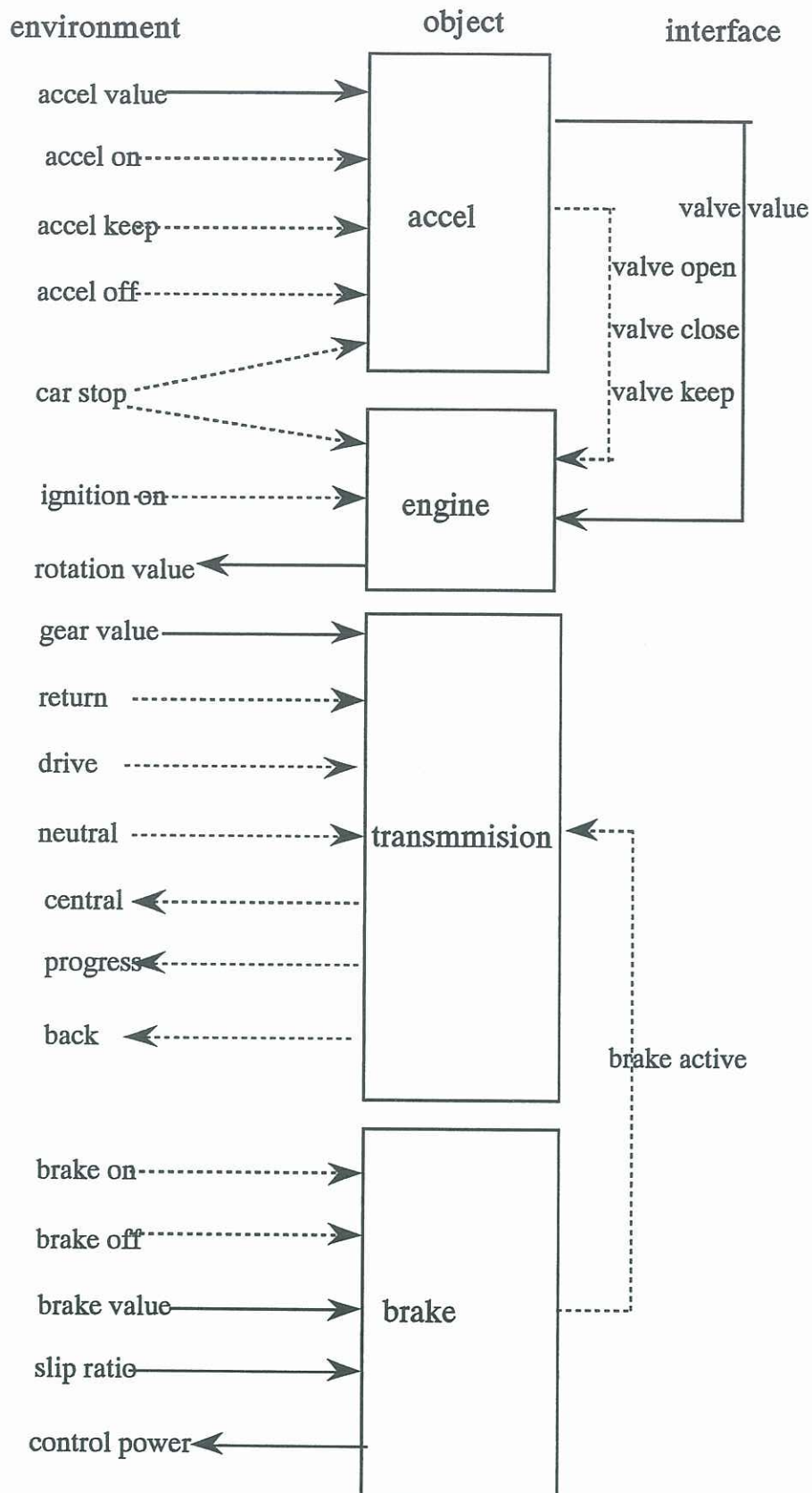
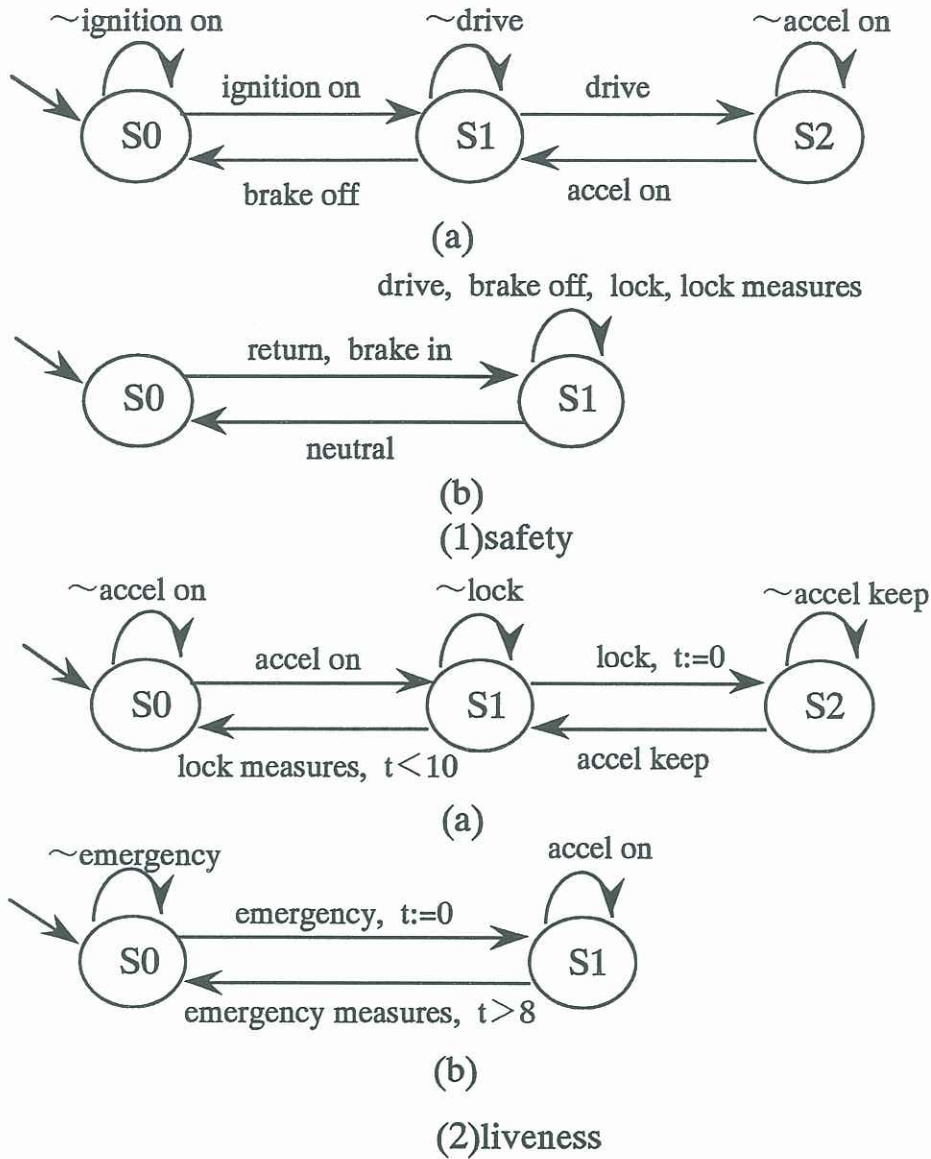


Fig.12 Global diagram between objects and environment



### 4.3 example of timing verification

Example of timing verification is shown as follows. Timing verification property such as liveness and safety is shown in Fig.13. Verification example of Fig.13(2)(b) is as follows.



where

$\sim$ : complementation

Fig. 13 Example of verification property specification

Fig.13(2)(b) property only depends on emergency and emergency measures. Intersection of system specification with the complement of verification property is shown in Fig.14 from interface rule<sup>20)</sup>. A set of accepting states of Fig.14 is as follows.

{ (stop, S0), (idle keep, S0), (rotation up, S0),  
(rotation down, S0), (emergency measures, S0) } ,  
{ (stop, S1), (idle keep, S1), (rotation up, S1),  
(rotation down, S1), (emergency measures, S1) } }

As there is no accepting loop using geometric region method, , this verification property is satisfied. Verification program quickly outputs YES.

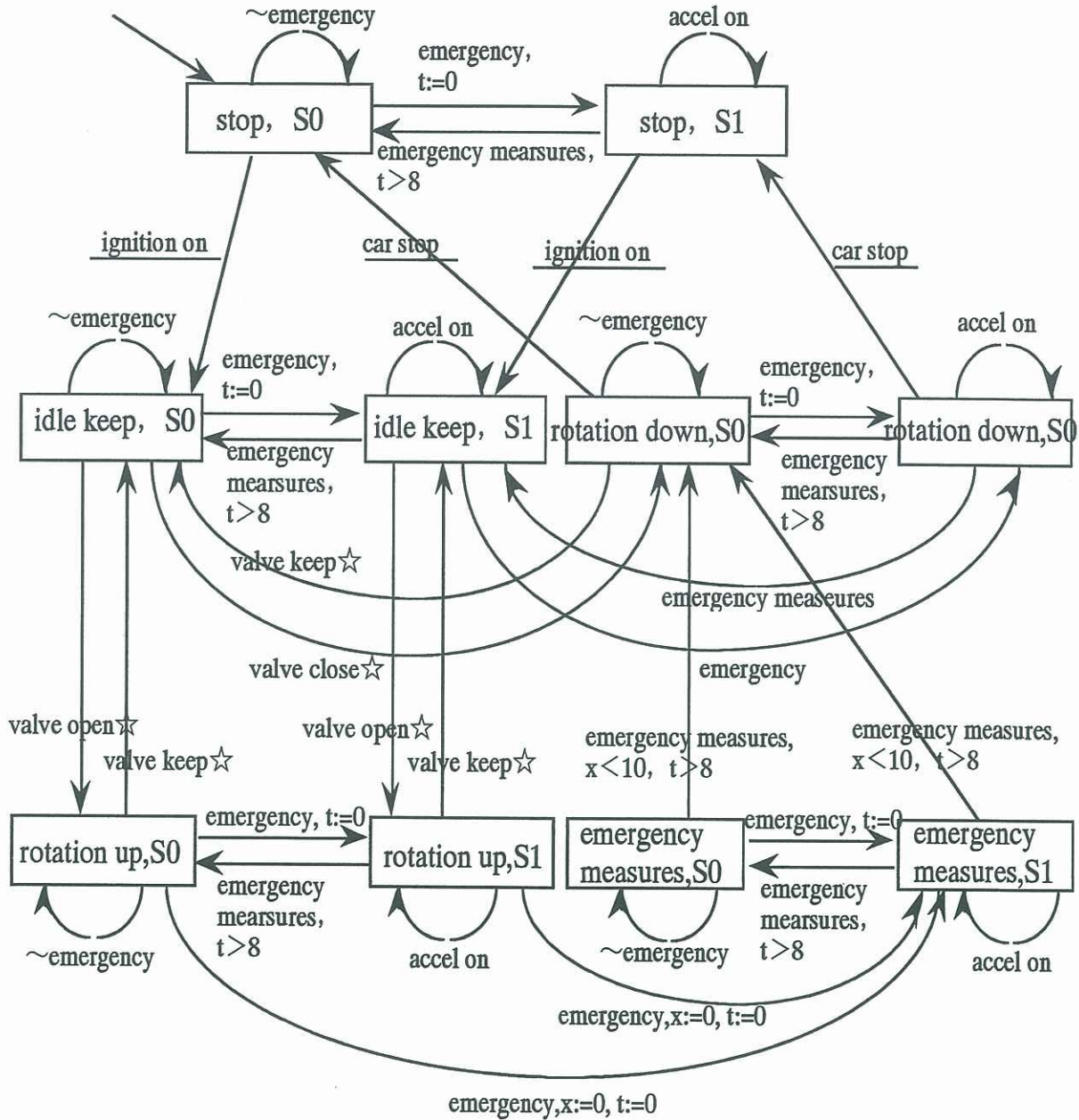


Fig.14 Verification example of liveness

Object behavior satisfying this verification property is specified by dynamic semantics shown in Fig.15.

```

    <Env, (ignition on, 1) , engine>
→<accel, (valve open, 2) , engine>
→<engine, (emrgency, 3) , engine>
→<engine, (emergencyt measures, 1 2) ,
engine>
→<Env, (car stop. 1 3) , engine>

```

Fig.15 Example of operational semantics description  
to satisfy liveness

## 5. conclusion

In this paper, object-oriented method for hard real-time software is proposed and it is shown effective by example of automobile. Main feature of method is as follows.

- (1) Formal specification explicitly including timing conditions is described in timed statechart, and verified by language inclusion problem.
- (2) Verification of consistency check of relations of models is realized by mapping object model into dynamic model and functional model.

Now verifacation program and automaton generator are implemented by 2~3 Kstep. I will implement GUI and mutual convertor between graphic representation and textual representation.

Real problem is that complex processes are multi-tasking and each process has timing conditions. If this method is applied to real problem, process van be specified as object and this method can be applied. Though there is state explosion problem, it is overcome by interface rule. I will try to apply this method to real problem.

## Reference

- 1)Kavi K.M.:Real-timeSystems,Abstraction,Languages,and Design Methodologies, P.660, IEEE Computer Society(1992)
- 2)Kim W. Lochovsky F.H.:Object-Oriented Concepts,Databases,and Applications,P.602, ACM PRESS(1989)
- 3)Rumbaugh J. Blaha M. Premerlani W. Eddy F. Lorensen W.:Object-Oriented Modeling and Design,P.500,Prentice Hall(1991)
- 4)Shlaer S. Mellor S.J.:OBJECT LIFECYCLES Modeling the World in States,Englewood Cliffs(1992)



- 5)Booch G.:Object-Oriented Design with Applications, Benjamin/Cummings(1991)
- 6)Buhr R.: "Architectures with Pictures",Proc. OOPSLA 92,pp.466-483ACM PRESS(1992)
- 7)Selic B. Gullekson G. Ward P.T.:Real-Time Object-Oriented Modeling,P.525,John Wiley&Sons(1994)
- 8)Coleman D.Hayes F.Bear S.: "Introduction Objectcharts or How to Use Statecharts in Object-Oriented Design",IEEE Trans. on SE, Vol.18, No.1, pp.9-18(1992)
- 9)Jungclaus R. Saake G. Hartmann T.: "Language Features for Object-Oriented Conceptual Modeling",Proc. 10th ER-approach, pp.309-324(1991)
- 10)Sernadas A. Sernadas C. Ehrich H-D.: "Object-Oriented Specification of Databases:An Algebraic Approach",Proc. VLDB'87, pp.107-116(1987)
- 11)Arapis C.: "Temporal Specification of Object Behavior",LNCS 495, pp.308-324(1991)
- 12)Breu R.: "Algebraic Specification Techniques in Object Oriented Programming Environment",Springer-Verlag(1991)
- 13)Alur R. Dill D.: "The theory of Timed automata",LNCS 600, pp.45-73(1992)
- 14)Harel D.: "Statecharts: A Visual Formalism for Complex Systems", Science of Computer Programming 8, pp.231-274(1987)
- 15)Hoare C.A.R.: Communicating Sequential Processes, P.256, Prentice-Hall(1985)
- 16)Dill D.: "Timing assumptions and verification of finite-state concurrent systems",LNCS 407, pp.197-212(1989)
- 17)Alur R. Courcoubetis C. Dill D. Halbwachs Wong-Toi H.: "An Implementation of Three Algorithms for Timing Verification Based on Automata Emptiness",Proc. Real-Time Systems Symposium, pp.157-166(1992)
- 18)Rokicki T.G. Myers C.J.: "Automatic Verification of Timed Circuits",LNCS 818, pp.468-480(1994)
- 19)Hatley D.J. Pirbhai I.A.: Strategies for Real-time System Specification, P.377, Dorset House(1988)
- 20)Clarke E. M. Long E.E. McMillan K.L.: "Compositional Model Checking",Proc. 4th Logic in Computer Science, pp.353-362(1989)