Abstract: - We propose an approach using models in the knowledge-base for the system design of real-time distributed systems. Our approach belongs to a logical approach and our modeling is state-based. Models in our knowledge-base are built from specifications in our specification description language. These models in the knowledge-base works greatly for (semi-)automation of system design in the upper software development processes, i.e., the verification of their system design with deadlock detection, direct execution of specifications by interpretation, and plan synthesis. We explain how our system works, using ATM system design as an example problem.

Key-Words: - Software design, Model, Knowledge-base, Design verification, Direct execution, Plan synthesis

1 Introduction

Recently, the needs of effective development method for real-time distributed system software have been increasing quite rapidly. These systems must handle events, message communication, and treatment of time information.

The specification methods of concurrent systems can be divided into the following two categories: one is behavioral approaches using scenarios [1] and algebraic specifications, and the other is approaches using logical specifications, such as Eiffel[2], VDM-SL[3], some works related with OCL[4], Model Driven Architecture (MDA)[5] using both UML[6] and OCL[7]. The merits of the former approaches enable to describe simple specifications using operation sequences and enable direct execution. The latter (logical) approaches are centered with state-based models, which are relatively easy to understand and to automate generation and refinement of plans.

Our approach belongs to a logical approach and is based on state descriptions. Recently, MDA is popular, but MDA seems to start from the class diagrams in platform independent model whose contents are already fixed. But, generally speaking, in the early stage of software development process, requirements are mutable, and we need a lot of trials by prototyping to fix designs with definite specifications.

We believe that knowledge is the most suited representation for reuse, so that we propose an approach which uses knowledge-based models with precondition-postcondition specifications which can be used effectively for the following design activities: (1) forward verification of target systems consisting of interaction-phases and component actions with the capability of detecting deadlocks existing in those system designs (2) direct execution of specification descriptions [8] (3) plan synthesis using a partial-order planner [9]. These activities use graph-like data structures for the design representation of target systems which is easy to edit and can preserve mapping between elements in the design model and those in the program model which are obtained through refinement processes.

We explain how our system works, using ATM system design as an example.

2 Terms and system organization

2.1 Terms

We explain some terms in models used for representing target system designs.

- Project: model of a whole target system consisting of interaction-phases and transitions between them (similar to the high-level message sequence chart, abbreviated as hMSC, in [1]). See the ATM Project and its hMSC in Fig.1.
- Component: agent[8]-like independent autonomous objects. User, ATM, Consortium,
and Bank are the components in the ATM system.

- Interaction-phase: model of a meaningful unit of interactions between components (similar to the basic message sequence chart, abbreviated as bMSC in [1]). See the badPassword Interaction-Phase and its bMSC in Fig.2.
- System-state: global state between adjacent interaction-phases.
- P-element: object which works passively, e.g., a database in a bank.
- Action: operation belonging to each component or p-element. Message-sending nodes and message-receiving nodes in bMSC in Fig.2 are the instances of actions of components.
- Local-state: local state between two adjacent actions of a component or of a p-element.

Our scenario-like model representations are suited to reflect the change of requirements to models and to add interaction-phases for exception-handling such as user cancel operations and system failures which are important in the design of real-time systems.

2.2 System organization

Fig.3 shows our system organization. We use text-representations for inputting knowledge in specification description language based on simplified predicate-logic expressions (3.1.1). These knowledge are transformed into model representation and stored in our knowledge-base (3.2), and we also give the internal structure descriptions of project and interaction-phases using a structure description language like Finite Sequential Processes (abbreviated as FSP [1]) (3.1.2).

Models in the knowledge-base are used in the following activities of upper software development process:
(1) Forward verification and deadlock detection of design model of target systems (4.1)
(1-1) Verification of project and its hMSC based on...
precondition-postcondition specifications of project and each interaction-phase and on assertions given to system states.

(1-2) Verification of each interaction-phase and its bMSC based on precondition-postcondition specifications of the interaction-phase and actions of components or p-elements belonging to each interaction-phase. We can find deadlock paths during this verification process.

(2) Direct execution of specification descriptions (4.2) Each specification (precondition-postcondition) description of actions in bMSCs can be interpretively executed using test data so that we can check target system design.

(3) Plan synthesis (4.3)
A partial ordered planner with backtracking is generally used for plan synthesis, connecting goals and premises of modules with partially ordered sequences of actions based on the causal links between predicates in the preconditions of actions and those in postconditions of other actions.

3 Specification languages and model representation in the knowledge-base

3.1 Input languages
As mentioned above, our system uses two input languages.

3.1.1 Knowledge description language
One language is for specifying various knowledge in our knowledge-base. Preconditions and postconditions of actions and modules (i.e., projects and interaction-phases) are given in simplified predicate-logic expressions with negation ('!'), logical-and ('&'), logical-or (limited use), and implication ('=>' ) operators, and limited forall and exist expressions. (We use negation operator '!' explicitly for negating state predicates in postconditions of action specifications. This frame action technique is convenient for representing state changes and for automating plan synthesis.)

Fig. 4 shows the definitions of the action 'send_enterPassword' of the component 'User' and the action 'receive_enterPassword' of the component 'ATM', which represent the message label 'enterPassword' between User and ATM in Fig.2 by attaching 'send_' at the sender side and 'receive_' at the receiver side, respectively, in front of the edge label. We use 'sent' clauses in the postcondition of send-actions and 'message' clauses in the precondition of receive-actions both of which have arguments in the following order <sender component, receiver component, message contents>. Those strings which have '?' or '*' at their heads are variables ('*' is used for representing a component). The values of these variables are the instances of classes specified by the types.

3. Specification description of interaction-phases consists of precondition, postcondition, and list of components working in the interaction-phase. Fig.5 shows the interaction-phase specification for 'BadBankPassword', where each instance of User, ATM, Consortium and Bank is listed as a component and the predicate "badPassword(?pass)" in the postcondition shows the verification result returned from the action 'badPasswordDetect' of p-element 'Account-Base' in the Bank component.
3.1.2 FSP language

The other language is for specifying internal structure of modules.

(1) Transition between interaction-phases

Internal structure (hMSC) of a project is specified by a list of system-states (with their assertions), a list of interaction-phases, and transitions between them. We show a portion of the project ATM-system specification in Fig.6 which corresponds to the hMSC in Fig.1. The ATM system starts from InitialState and transits between interaction-phases, following the order designated by the symbol ‘->’. Assertions of each system state are given at the column ‘ASSERTION’.

(2) Parallel sequential processes

Internal structure (bMSC) of each interaction-phase is specified by parallel process descriptions, namely, action-sequences (using symbol ‘->’ ) of parallel components.

3.2 Knowledge-base

Sentences described in our specification languages are parsed, transformed into model representation and stored in our knowledge-base. (and stored simultaneously in the knowledge index, which is mainly used for extracting action candidates from predicate information in the plan synthesis phase.) We have been building our knowledge-base by making use of Java class hierarchy. This knowledge-base consists of classes such as Project, InteractionPhase, SystemState, Component, PElement, Action, and LocalState as well as fine-grained classes such as LogicalOperator, Predicate, Variable, and Constant.

4 Verification, direct execution and plan synthesis

4.1 Verification of target systems

As mentioned in 2.2, we verify (1) project and hMSC and (2) each interaction-phase and its bMSC. Here we explain the process of the latter verification.

Following the action sequences of each component in an interaction-phase, we construct a verification tree (or graph) which has all the (logically) possible branches of interleaving component action sequences, where the precondition of newly expanded action must be satisfied with immediately preceding local state and the next local state reflects the state change caused by the postcondition of the action. Also causal links between sent-clauses of message-sending actions and message-clauses of message-receiving actions must be taken into consideration. These work as constraints on action sequences between components. Fig.7 shows the causal links between predicates of the precondition of 'BadBankPassword' interaction-phase and those of the precondition of 'send_enterPassword' action of ATM component, and causal links between predicates of the postcondition of 'send_enterPassword' action of ATM component and those of precondition of 'receive_enterPassword' action of User component. We can find deadlock paths during this verification process.

From the hMSC of a verified project and action sequences of verified interaction-phases, we can derive control structure model of each component which is indispensable to direct execution of
specifications and refinement to program models. The control structure model is a graph model, like the activity diagram in UML, representing control structure with message communication between components, but it has no judge symbols but plural edges indicating their possible continuation. We give a portion of the control structure model of ATM component in Fig.8.

![Fig.8. Control structure model of the ATM component.](image)

### 4.2 Direct execution of specifications

We can write the interpreter for the logical specification and can directly execute the test run based on the models of specification. This hint came from agent programming [8], as our components in our interaction-phases can be interpreted as agents whose programs are represented in rules in production systems. Each component and its actions correspond to an agent and its production rules respectively. Preconditions and postconditions of actions correspond to condition-parts and execution-parts of rules with the interpretation of negation predicates in postconditions as the delete-lists in rules (and with some use of subroutine-calls in the definition of actions). For each component in an interaction-phase a thread is assigned that shares the basic message sequence chart, interprets those actions along their partial order and behaves itself accordingly. The interactions between a user and an ATM are interpreted as display of the message output from computer to user and read-in of data through keyboard from user.

### 4.3 Plan synthesis

Plan synthesis proceeds backward along the direction from goal to premise, finding proper actions in the current interaction-phase or candidate actions in the knowledge-base which has predicates in the postconditions which satisfy predicates in the goal or derived subgoals.

We use the planner mainly in refinement stages, when some actions which are not present in abstract level are required to bridge the gap between actions, and when models introduced from different views are required to be merged into a unifying model by arranging those actions in both modules into logically consistent order through examining causal links. The latter planning is useful when we use design-pattern knowledge, e.g.,
when embedding control-structure models into the body-part of 'multiclient-server' architecture pattern [10].

5 Conclusion

We proposed an approach using models in the knowledge-base for the system design in the upper software development process and explained how these models are used in verification, direct execution and plan synthesis, using the ATM problem as an example.

Explanations on the treatment of information related with time, refinement processes from abstract level to concrete level (Java program level), and the use of knowledge of design patterns were not given in this paper. Topics related with these will be given in our forth-coming papers.

References: