On the Synthesis of Upper-Stream Models for Various Web Application Systems

Takakazu Kaneko, Nobuoki Mano
Graduate School of Informatics
Meisei University
2-590 Nagabuchi, Ome-shi, Tokyo-to 198-8655 Japan

Abstract:- A system for synthesizing platform-independent models of web application systems such as ATM simulator, students-teacher schedule managing system, and so on from problem specifications using various kinds of domain knowledge is described. System design which solves problems in this area are synthesized as models of problem solution based on the structure descriptions of data which are assigned as the domain boundaries of user interface, network system, and database in the server.

Keywords:- Web application, Client-Server system, Knowledge-base, Synthesis of Upper-stream models

1 Introduction

ATM simulator, conference-room reservation system, and online-shopping system are classified into a kind of service-oriented system. These systems are comprised of heterogeneous domains such as user-interface at the client-side, communication between client and server, and server-side database handling with business logic operations. As the prevailing web service system, we often find system configuration where TOMCAT server, Servlet, JSP, and SQL at the server-side system and browser at the client are used. However, it is pointed out that mixture of various kinds of languages and complex dependency relationship between pages make the extension and maintenance of application development difficult. We also feel that application systems mentioned above are worked out from common knowledge of each domain.

So we have been making researches on a system-design method that makes full use of model in our knowledge base [1]. We give in this paper an intelligent method for generating upper-stream models from problem structure, specifications of interactive commands, and structure description of user database. These upper-stream models, which reflect graph-like problem structures in UML [2] and include logical specifications like OCL (Object Constraint Language [3]), resemble PIM (Platform Independent Model) in MDA (Model Driven Architecture) [4] and will be refined to PSM (Platform Specific Model). The derivation method proposed in this paper relieves us not only from the labor of writing down rigorous upper-stream models, which are checked for their internal consistency by forward verification and directly executed to show that they satisfy user requirements [1], but also from the labor of writing down a lot of specifications, which are required in the case we have to write down when not deduced automatically.

This paper consists of the following chapters. In Chapter 2 we explain features of problem definition, knowledge base, and models in the upper-stream level comprising of 3 domains, i.e., user interface, communication system, and database (for simplicity, we use data structures). In Chapter 3 we discuss the derivation of upper-stream models from problem specifications. We use ATM system and students-teacher schedule managing system as our example problems.

2 Problem definition, knowledge base, and models in the upper-stream level

2.1 Web applications and their composing domains

One example is the ATM system, where User, ATM, Consortium, and Bank (with AccountBase) are the components. Interaction command sequence and server-side database composition must be given, too.

Another example is a schedule managing system as follows. One schedule table is a two-dimensional array consisting of days of week and time slots. Each student has a schedule table for himself. The teacher can have plural schedule tables distinguished by titles. Students are able to look at their own table and a fixed table of their teacher, and the teacher can look at all the time-tables of all students in addition to all of his schedule tables.
User schedules are in the database on the master-server. These web application examples are consisting of 3 domains , namely, communication subsystem, user-interface, and data-structure handling subsystem.

Each of these domains has the following features (in the upper-stream level).

① Communication between client and server

In this paper we only discuss synchronous systems, namely, request-and-reply of client-server architecture (especially for plural users, what we call multi-client-server architecture where an exclusive slave-server is assigned to each client).

②Client user interface and interaction

Data input from user, data output and message from server are shown in the user interface. Data exist first, and from them picture elements to manipulate data are introduced in the lower level.

③ Database (or data structures) and data extraction operations

In any application of web application systems we have some databases in the server, and from them we extract necessary data to reply to user request, or we update the data.

Table 1 is a table which shows problem specifications, knowledge base, models, and problem-solving of each of these domains in the upper-stream level.

2.2 Problem specifications

Our system requires following problem specifications as the input.

problem specifications	knowledge base	problem-solving	models
①communication:			
processing structure	architecture patterns	derivation of definitions for communication operations	
②user interface:			
interaction sequence	commands		transition graph on interaction-phases
③data-base:			
structure description of data and its invariant	abstract data-types	derivation of definitions for data structure handling operations	definitions for data structure handling
④overall:			
		interaction-phase decomposition	action sequence of components

Table 1 Features of domains in the upper-stream level.

① Processing structure

Components, which denote autonomous entities, such as user, client, server, and intervening component, and combination between them are specified with multiplicity which shows the correspondence information between these sets. ATM system consists of set of components: User, ATM, Consortium, and Bank (with AccountBase).

② Specification on interaction sequence

A system transition graph consisting of interaction-phase set and system state set is obtained from the regular expressions of interaction sequences. An assertion must be given to each system state. Each of the precondition-post-condition specification of interaction-phases is determined from those assertions of adjacent system states. A command (shown in the next section) with an input-output specification corresponds to each interaction-phase.

③ Specifications on structure and constraints of the database on the server

For example, the structure of AccountBase in the ATM system is given by the relation, { <accountNumber, password, balance>} and the constraints are given by the description

key(accountNumber),
accountNumber → password,
accountNumber → balance,

where '→' and ',' represent functional dependency relationship and logical-and, respectively.

② and ③ must be specified with same vocabularies as for their common parts.

2.3 Knowledge base

①Architecture pattern and related send/receive-action specifications

Parametric design pattern and architectural pattern definitions [5],[6],[7] bring us flexible usage of knowledge on actions. In our system, send-action and receive-action specifications of components are determined by using client-server and multi-client server architecture patterns.

As the result of matching the architectural pattern with concrete problem substructure (e.g., ATM-ConsortiumSlaveServer), component names, and messages (command name with argument part) in the role are fixed with corresponding concrete names.

The context descriptions of component roles (|RoleName represents role name) in the multi-client-server pattern are as follows. In Fig.1, types of arguments of actions are represented as unary predicates in the precondition. '*' and '?' marks represent a component variable and a usual data variable, respectively. verify(?acc, ?password) is the message-body, and dynamic (i.e., delegating and acting-for) relationships

```
context |Slave::|receive-|action(?|Type1, ?|Type2)
  pre: |Client(*|Client), |Slave(*|Slave),
       connected(*|Client, *|Slave),
      |Type1(?|Type1), |Type2(?|Type2),
    message(*|Client, *|Slave,
                |action(?|Type1, ?|Type2));
  post: |action(?|Type1,?|Type2},
        |actionFor(*|Slave, *|Client);
```

Fig.1 Meta-level definition of receive-action of slave server (used with the "multi-client server" architecture pattern).

are represented by attaching "Ing" and "For" to the tails of action names.

② Commands

The following are the representative linguistic commands which can be used at the upper-stream level:

register, add, create, delete, verify, retrieve, enumerate, update, modify, select, signal, fill-in, and some commands related with data handling operations.

③ Data structure

Super and subclass relationships of data-types are included as part of knowledge-base. Relation data-type has the correspondence information between sets represented with the concept of primary key and function dependency, as the data-type invariant. Its overwrite operator is inherited from the mapping type. Complex data are composed of direct product combination and hierarchical combination of simple types.

2.4 Concepts related with our upper-stream model

Here we explain some concepts in our upper-level models used for target systems [1].

- Project: model of a whole target system consisting of interaction-phases and transitions between them.
- Component: agent-like independent autonomous object. User, ATM, Consortium, and Bank are the components in the ATM system.
- Interaction-phase: model of a meaningful unit of interactions between components.
- System-state: global state between adjacent interaction-phases.
- Passive-element: object which works passively, e.g., a database in a bank.
- Action: operation belonging to each component (including message-sending and message-receiving actions) or passive-element.
- Local-state: local state between two adjacent actions of a component or of a passive-element.

3 Derivation of the upper-stream model - basic principle with examples -

We pay special attention to the structure of data for each command on the boundaries of each domain along the path between user interface in the client and user database in the server.

Upper-stream models are derived from application problem specifications by the following process.

(1) Specify your target problem (as for the system structure with description of interaction sequence at the client and specification description of data structure at the client and server)

(1-1) generate a transition graph consisting of interaction-phases and system-states.

Attach an assertion to each system state. From the assertions of preceding system states and those of following system states, the specification of the interaction-phase bridging these states can be determined.

(2) Decide the applicable abstract data-type

Applicable abstract data-types for the data structure of a target problem are found through matching structure and data-type invariant of the problem with those in the knowledge base.

(3) Try to form a plan (or a hierarchical group of plans) for each interaction-phase (with related modifications)

(3-1) for the interaction-phase, determine its command specification.

(3-2) determine action specification for the database in the server.

(3-3) determine send-action/receive-action specifications of communicating subsystem.

We can obtain precondition–post-condition specification of corresponding action (as for, "receive-verify" action of Consortium, see Fig.2) from the action template of component role in the pattern (see Fig.1) [5]. Its message part is replaced by the command and its arguments.

```
Consortium::receive-verify(?acc, ?password)
  pre: ATM(*ATM),Consortium(*Consortium),
       connected(*ATM,*Consortium),
       accountNum(?acc), password(?password),
       message(*ATM,*Consortium,
                verify(?acc,?password));
  post: verify(?acc,?password),
       verifyFor(*Consortium,*ATM);
```

Fig.2 Action specification of “receive-verify” in the Consortium component (from the architectural pattern specification matched with problem processing structure).

(3-4) generate a consistent plan for each interaction-phase.

Each predicate in the post-condition of an interaction-phase is satisfied with predicates in the post-condition of actions, or those in the preconditions of interaction-phase. Each predicate in the precondition of actions in the interaction-phase is satisfied with the predicates in the post-condition of actions or those in the precondition of the surrounding interaction-phase. Consistency of combination of these predicate links (called causal links) is checked. Here, “consistent” means that the following conditions have been proved:

(a) there exist predicates in the post-condition of other actions or those in the precondition of interaction-phase in the plan,

(b) form of the "before" relationships between actions including causal links composes a partial order directed acyclic graph.

(c) there exist no disturbance of side effects already included in the plan.

(d) temporarily established relationships such as "delegating" and "acting-for" are dissolved, and not existent afterwards.

(3-5) if we can get consistent plans for all interaction-phases, go to (4). If unexamined interaction-phases remain, continue (3).

(3-6) if we cannot get consistent causal links over the plan, we will try to fix transition graph in either or both of the following way.

(3-6-1) try to generate specification definitions of actions.

Fig.3 is an example of specification generation of "withdraw" action in the ATM problem, based on the invariant condition that balance >= 0. We can obtain (e) by going back to (a) which includes the operation (b) generating the non-allowable value "balance'". The post-condition is further divided into two parts as to allowable-range and non-allowable range.

(3-6-2) try to decompose an interaction-phase into the sequence of some subcommands.

Fig.4 is an example of interaction-phase decomposition in the schedule-managing system. Here, the argument part of each command is separated by the symbol ';' into 2 parts: the former part represents input arguments and the latter part represents output argument. An intermediate system state where candidates to be selected are shown is inserted as a stepping-stone. The "enumerate" command collects the specified candidates in the database and shows them, and the "select" command is a subcommand in a larger context, where one is selected by the user among the shown candidates.

(3-7) if you cannot find fixing methods, go back to the starting point and re-examine the problem.

(4) Transform the completed transition graph of interaction-phases into the models centered around each component like activity diagrams in UML [2].

(a) retrieval of the relational data including balance
based on an account number
<?acc, ---, ?balance>
(b) calculation of withdrawal
?balance' == ?balance - ?amount
(c) update of the relational data
self ++ <?acc, ---, ?balance'>,
Here, '++':is an overwrite operator, and
self == {<String: accountNum, String password, int balance>},
key(accountNum),
(d) violation of the invariant condition on balance
allowable range: (?balance >= ?amount):
put those postconditions of (a), (b), and (c) together
non-allowable range: (?balance < ?amount)
SystemResponce("NO")
(e) actionDef AccountBase::withdraw(?acc, ?amount)
pre: accountNum(?acc), amount(?amount);
post: self == {<?acc, ---, ?balance>}
post-1: ?balance >= ?amount ⇒
?balance' == ?balance - ?amount,
self ++ <?acc, ---, ?balance'>
post-2: ?balance < ?amount ⇒
SystemResponce("NO");

Fig.3 Example of generation of an action specification.

```
commandDef  selectAndShowScheduleTable
                (?setObject ; ?scheduleTable)
    pre:    setObject (?setObject);
    post:   scheduleTable(?scheduleTable);
=>
subCommandDef
enumerate(?setObject ; ?candidateElements)
    pre: setObject(?setObject);
    post: objectList (?candidateElements);
subCommandDef:
select(?candidateElements ; ?selectedElement)
    pre: objectList (?candidateElements);
    post:   scheduleTable(?selectedElement),
        ?selectedElement∈?candidateElements;
```

Fig.4 Decomposition example of an interaction-phase.

4 Conclusion

In this paper we proposed an intelligent derivation method of upper-stream models from the viewpoint of meta-model and showed the derivation process using ATM problem and schedule managing system as the examples. With this method we can relieved from the burden of rigorous and quantitative specification writing. This method is effective to developing various web application systems. The research on the refinement of upper-stream models of problems to program models of them has been pursued [8].

Finally, the authors would like to express our appreciation to Prof. K. Saishu and the staff of our Graduate School of Informatics for their help and support.

References

[1] N.Mano and T.Kaneko: A Knowledge-Based Modeling Approach for Verification, Direct Execution and Plan Synthesis of System Design, *WSEAS Trans. on Information Science & Applications* Issue 8, Volume 2, 2005, pp.1065-1070.

[2] H.E.Erickson, M.Penker, B.Lyons, and D.Fado: *UML 2 Toolkit*, Wiley Publishing, Inc., 2004.

[3] J.Warmer and A.Kleppe: *The Object Constraint Language [Second Edition]: Getting your Models Ready for MDA*, Addison-Wesley, 2003.

[4] A.Kleppe, J.Warmer, and W.Bast: *MDA Explained - The Model Driven Architecture: Practice and Promise*, Addison-Wesley, 2003.

[5] R.B.France, et al.: A UML-Based Pattern Specification Technique, *IEEE Trans. on Software Engineering*, Vol.30, No.3, 2004, pp.193-206.

[6] F.Buschmann, et al.: *A System of Patterns: Pattern-Oriented Software Architecture*, Wiley 1996.

[7] E.Gamma, R.Helm, R.Johnson, and J.Vlissides: *Design Patterns: Elements of Reusable Object-Oriented Software [second edition]*, Addison-Wesley, 1998.

[8] T.Kaneko and N.Mano: On the Refinement of Upper-Stream Models to their Program Models in Web Application Systems, to be published.