On the Refinement of Web Application Systems from Upper-Stream Models to Program Models

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

Abstract: - We propose a knowledge-based approach of refining web application systems from upper-stream models to program-models. The uppermost specifications and upper-stream models have significant effects on the program models from which we can derive source programs in text format. For deriving program models, analysis of upper-stream models and design integration of information obtained from the analysis are required, as program models and the structure of them often include those parts which are not included explicitly in the upper-stream model. We discuss specifications, models, knowledge, and refinement process on each of domains: user interaction, communication subsystem and database.

Keywords:- Client-Server system, Refinement, Upper-stream model, Program model, Knowledge-base

1 Introduction

We have been making researches on the upper-stream modeling of web application software consisting of hetero-genius domains: user-interaction, communication subsystem (client-server system), and database (at present, data structure handling program) [1], and also on the derivation of the upper-stream models from their uppermost specifications [2]. In this paper, we propose a knowledge-based approach of refining upper-stream models to program models. Program models are finally transformed into source codes in Java. Our approach belongs to a logical state-based approach using specifications expressed in the modified predicate 1ogic [3][4][5].

The purpose and the goal of our approach are close to those of the MDA (Model Driven Architecture) [6] using OCL (Object Constraint Language [7]) expressions. In MDA, PDM (Platform Dependent Model) is derived from PIM (Platform Independent Model), using class diagrams as their context. MDA process seems to employ just local transformation of upper-stream models by transformation rules. Invariant conditions are part of their programs in the declarative format and finally transformed to procedure programs. MDA is proposed as a programmer-oriented refinement method.

On the contrary, our refinement method is knowledge-based and employs a uniform logical representation formalism from upper-stream level to lower program-model level so that it is more suited for automated refinement. In addition, our method comprises of both analysis of upper-stream model and design integration of the result, because structure of program models are often different from their upper-stream models, and there exist some parts introduced through design decisions.

The contents of this paper are as follows. In Chapter 2, we briefly introduce our refinement process, and refer to the uppermost specifications, the upper-stream models, and the program-models. In Chapter 3, we describe implementation knowledge and the program knowledge base of our system. Then in Chapter 4, we explain the process of refining upper-stream models to program models, where we use some problem-solving methods. From our program model we can derive complete Java source program with exception handling and package import statements, which we do not mention in this paper. We explain our refinement method, using ATM system as an example.

2 Specifications and models related with refinement

2.1 Overview of our development process

We show our development process for target systems in Fig.1. As shown in the figure, uppermost specifications and upper-stream models with knowledge in the knowledge base have effects on refining upper-stream models to program models. Specifications, models, and knowledge are dependent on each of domains: user interaction, communication subsystem (client-server system), and database (at present, data structure handling program). So we explain the feature of each domain in the following sections.

2.2 Uppermost specifications

The uppermost specifications of our system are as follows.

Processing structure of a target system

Representation of the processing structure of a target system is given with the multiplicity information between adjacent components. Interaction command sequence and server-side database composition are also shown there. Parametric design pattern and architectural pattern definitions [8] [9] bring us flexible usage of knowledge on actions [10]. In our system, send-action and receive-action specifications of components are determined by using client-server and multi-client server architecture patterns.

User interface

Specification of interaction sequences are specified with functional definition of each command :such as verify, enumerate, retrieve, update, select, and so on.

Data structure

Specification on the structure and internal constraints such as keys and functional dependency are given for database in the server.

2.3 Upper-stream models [1]

Whole movement of system control in target systems are represented with transition graphs consisting of interaction-phases and system states and transitions between them. Each interaction-phase corresponds to the pursuit of a certain command by some cooperating independent components using passive elements such as database. The operations of each component (including message-sending and message-receiving operations) or passive-element are defined by their action definitions using precondition-post-condition specifications. Our upper-stream models use only string-type data.

2.4 Program models

Our system is a meta-system for synthesizing program-models in the program-model-knowledge-base, so that our program models are not only coarse-grained but also fine-grained. We take up the following classes as the classes for our program-model knowledge-base: ClassM, FieldM, MethodM, ConstructorM, Statement, MethodCall, CreateInstance, ControlConstructs, InnerVariable, Predicate, LogicalOperator, and so on. Super-class (or sub-class) relationships between the instances of classes are also defined.

The inner structure of a method is represented as the amalgamation of a program-tree (see Fig.4) and hierarchical plan structures. Each level of a program tree corresponds to a plan which is directed acyclic graph consisting of instances of Statement node in the level and edges representing “before” relationships between nodes.

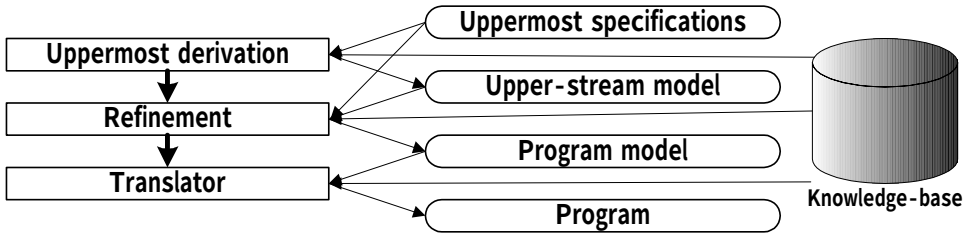

Fig.1 Development process of target systems

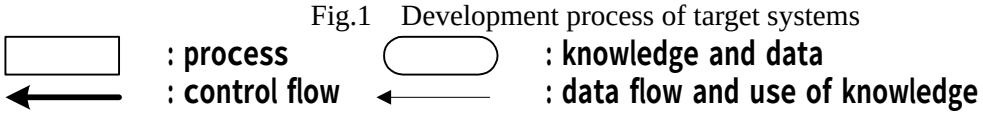

Table 1 Domain Knowledge related with the refinement of upper-stream level to program model level.

problem specifications	implementation knowledge	use of information in the upper-stream model	program model knowledge base
communication subsystem: processing structure	selection of communication stream		BFU (master-server, slave-server, client)
user interface: interaction sequence	command realization	transition graph on interaction-phases	BFU (dialog, etc)
database: structure description of data and its invariant	implementation data structure	ordering relationships, reference relationships	BFU (data-types)
overall:	transformation rules	transformation of assertions in system states from upper-stream model to program model	Java API model, meta-code statement

3 Knowledge and knowledge-base

3.1 Overview

Table 1 is the table of problem specifications, implementation knowledge, use of information in the upper-stream model, and knowledge and program model knowledge base, in each domain of communication system, user-interface, server-side data structure, and overall domain in the refinement of upper-stream level to program model level. The details are explained in the following sections.

3.2 Implementation knowledge

Design decision rules are used in the implementation in our refinement process.

As for communication subsystem, user of our system selects a stream type (text stream, or byte stream such as object stream), which leads to decision of read-write-operations to be used in the implementation.

As for user Interface, each command in the uppermost specification has some graphic methods to implement, one of which is to be selected.

As for the data structure in the server, some concrete data structures have to be selected to implement abstract data-types.

As for the overall issues, decision of implementation method leads to employ concrete transformation rules for state-assertions in upper-stream models to those in program models.

3.3 Use of information in upper-stream models

Information included in the upper-stream models can be used to decide not only the structure and elements of their program models but also relationships between them.

- The transition graph on interaction-phases of target systems are preserved even in the program-model level.
- Assertions attached to system states in the upper-stream level are transformed to those in the program-model level, where constructed program-models are checked for their consistency.
- Ordering relationships of operations in the upper-stream models have significant effects on those of operations in the program model.
- Reference relationships in the upper-stream models work as the specification for its program model.
- The data transmitted from client to server introduce the data class (such as ObjectStream class).
- Some data extracted from upper-stream models are classified and grouped with some interpretation, for composing graphical layout and for constructing object-stream.

3.4 Knowledge in the program-model knowledge base

Knowledge in the program-model knowledge base is roughly classified into those related with codes and production rules.

A. Knowledge related with codes

Knowledge related with codes is further classified into those in the model level and those in the meta-model level. For use, meta-level knowledge has to be patched for its parameterized parts.

Model level

(1) Usual classes with invariant conditions and methods wrapped with precondition-postcondition specifications: Each classes and methods in Java API has the counterpart knowledge in our knowledge base.

(2) Customized Functional Unit (CFU): These are obtained from customizing Basic Functional Units in the meta-level (described later).
(3) Application modules generated by our system: Classes and their methods in our knowledge base are represented in the form described in 2.4.

Our knowledge-base has an index system to retrieve efficiently method-models which have the specified predicates in their precondition or post-condition, because our system wants to know whether newly inserted statement might have some effects on the causal links of the target CFU.

Meta-Model level

(1) Meta-code-statement with its precondition, post-condition, and program meta-code
Meta-code representation can be applied to a wide range. For example, statements of assigning a value passed to instance field are used often in constructors. Statements of generating instances (and its assignment to a variable of the same type) are also used quite often in object languages.
(2) Basic functional unit (BFU)
BFUs are domain-dependent general and meaningful units for constructing methods (and sometimes, classes). For example, essential part of the operations of client, master-server, or slave-server and their operation sequences can be extracted and stored as one of BFUs. These BFUs are useful for constructing multi-client-servers in the ATM system.
(3) Design patterns and architecture patterns [2],[10]
These patterns are also very useful as the knowledge for automating refinement process.

B. Production rules

Design process is a process of integrating results of analysis, which can be done using production rules. For example, constructing a graphical hierarchical structure consisting of graphic elements with necessary attributes requires bottom up composition process using production rules.

4 Refinement to program model

4.1 Outline of refinement process

Fig.2 shows the outline of the procedure to derive the program model of a target system from its system specification and its upper-stream model. We describe some central issues in the following sections.

4.2 Generation of classes and their methods

Fundamental classes for each domain of a target system are introduced from the program-model knowledge base by the design decision of implementation, and their associations are specified. Methods for each of those classes are introduced through the inspection of upper-stream models.

Of course, in the primary classes corresponding to

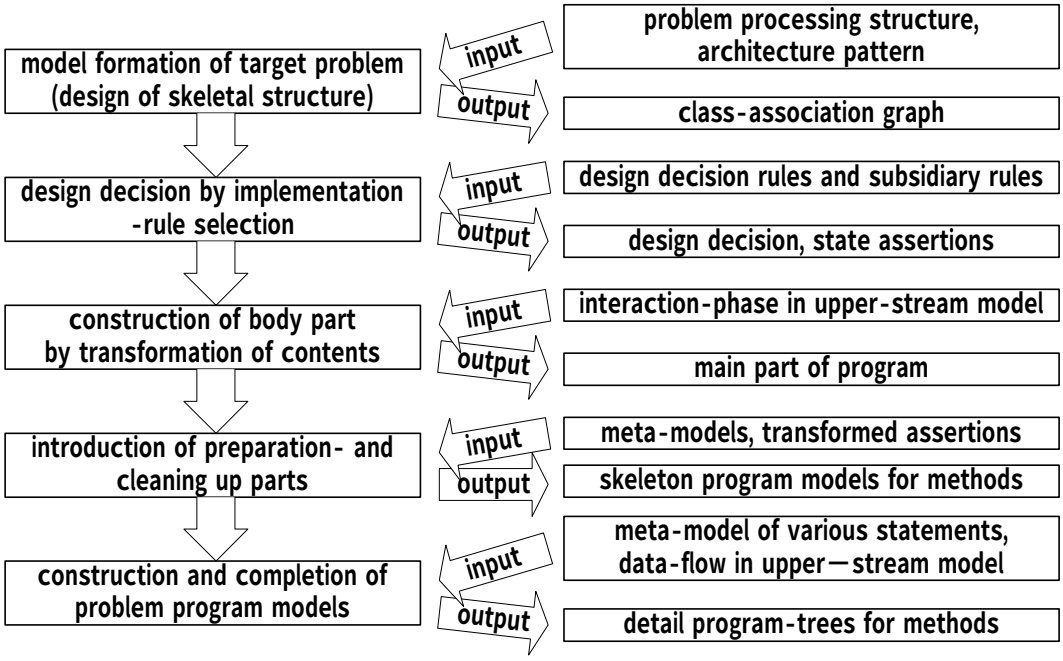

Fig.2 Flow of refinement process.

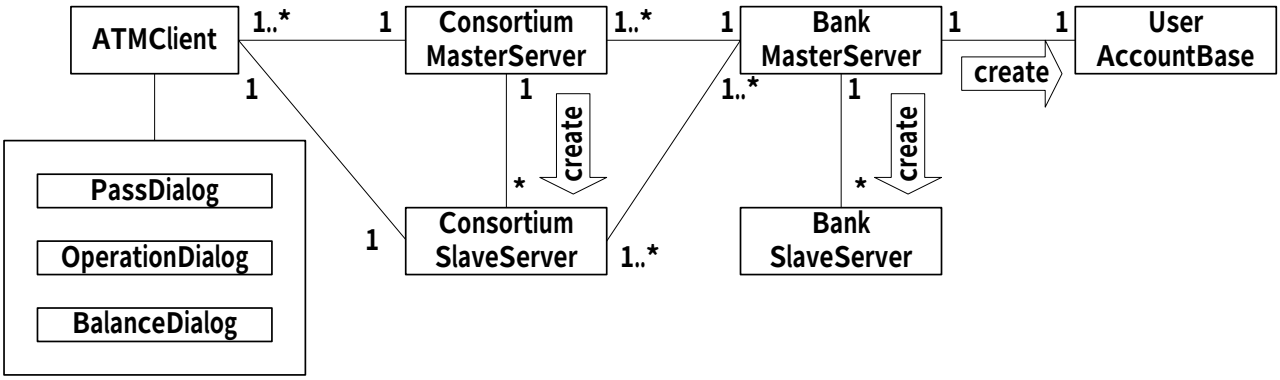

Fig.3 Class-Association Diagram of the ATM System.

components in the upper-stream-model, main method are required to create instances of these fundamental classes.

ATM example: Fig.3 shows the classes and their associations of the ATM system.

4.3 Construction of method programs

Main techniques for constructing methods of classes in the program-model are (a) construction of plans, (b) insertion of statements, and (c) access path generation. Each of these processes require consistency checking.

ATM example: The Inner structures of some methods in the ATM system are shown in Fig.4.

(a) Construction of plans [11]

Complete program models consist of both essential parts and subordinate parts. For example, the introduction and attachment of network connect part before the body part is indispensable. This construction is realized through plan formation process using causal links between assertion of local state and postcondition of "Prepare" CFU.

(b) Insertion of statements

Usually, use of BFU requires modification by inserting statements.

ATM example: A statement generating an

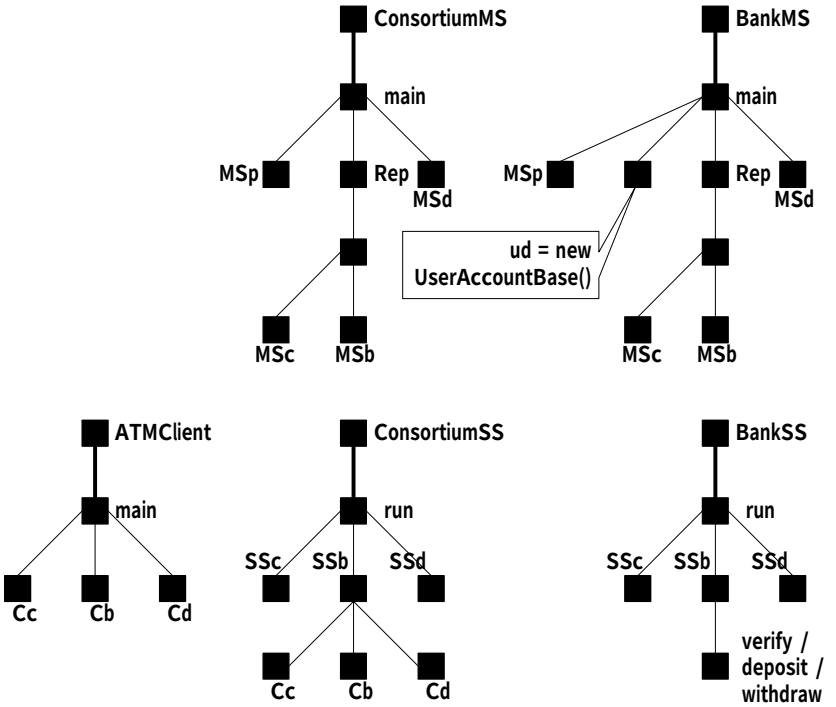

Fig.4 Program-tree models of some methods in the ATM system.
C: Client, SS: SlaveServer, MS: MasterServer, p: preparation –part, c: connect-part, b: body-part, d: disconnect-part

instance of UserAccountBase class is inserted into BankMasterServer, as shown in Fig.4.

(c) Access path generation

To fulfill the reference specification of upper-stream model, we need search to find the reference path from a sink to its source along associations represented in the application program model, and from the result of successful search we insert suitable elements to make flow of data along the path of instance generation of another class.

ATM example: Tracing "create path" along the opposite direction from BankSlaveServer to BankMasterServer, and from BankMasterServer to the UserAccountBase along the normal direction we can find the reference path that satisfies the requirement of dataflow from Bank to UserAccountBase in the upper-stream model. Along the reverse direction of this path we can make the reference path by adding a formal argument of type "UserAccountBase" to the constructor of BankSlaveServer class and inserting a statement for preserving the value to newly added instance field of the class.

.5 Conclusion

We proposed a refinement 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 the refinement processes using the ATM simulator as an example. We also try to refine a student-teacher-schedule managing system, and are now implementing our refinement system.

Finally, the authors thank anonymous reviewers for their comments, which are very helpful for making our paper easy to read. Also 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] T.Kaneko and N.Mano: Derivation of Upper-Stream Models for Web Application Systems from their Uppermost Specifications, *WSEAS Trans. on Information Science & Applications* Issue 4, Volume 4, 2007, pp.885-892.

[3] C.B.Jones: *Systematic Software Development using VDM [Second Edition]*, Prentice-Hall, 1990.

[4] J.Fitzgerald and P.G.Larsen: *Modelling Systems*, Cambridge University Press, 1998.

[5] B.Meyer, *Object-Oriented Software Costruction [Second Edition]*, Prentice-Hall (1997).

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

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

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

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

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

[11] S.Russel, and P.Norvig: *Artificial Intelligence - A Modern Approach [Second Edition]*, Prentice-Hall, 2003.