LEAD++: An Object Oriented Language for Dynamically Adaptable Software Model

Noriki Amano and Takuo Watanabe *

Graduate School of Information Science, Japan Advanced Institute of Science and Technology
1-1 Asahidai, Tatsunokuchi, Ishikawa, 923-1292 JAPAN
Phone: +81-761-51-1256, Fax: +81-761-51-1149, E-MAIL : {n-amano, takuo}@jaist.ac.jp

Abstract—**Software has *dynamic adaptability* if it can adapt itself to dynamically changing runtime environments. As open-ended distributed systems and mobile computing systems have spread widely, the need for software with dynamic adaptability increases. We propose a model of the software with dynamic adaptability. We are currently designing and implementing a programming language based on the model. The basic idea is that we introduce the mechanism which changes *procedure invocations* dynamically depending on the states of runtime environments. Our language LEAD++ provides the mechanism as a language functionality. Although our software model is simple, it can realize highly dynamic adaptability (including extensibility) of the software. By using LEAD++, we can develop *dynamically adaptable software* easily and systematically.**

I. INTRODUCTION

Nowadays open-ended distributed systems and mobile computing systems have come into wide use. When we design software for such systems, we cannot obtain accurate information of dynamically changing runtime environments beforehand; e.g., logical/physical location, hardware configurations (esp. including PC-card devices), software configurations, etc. The changes of runtime environments have given a strong influence on the execution of the programs, which we cannot ignore.

We gives two typical situations as examples. *1*) Although we can use the large & luxurious application such as *Netscape Communicator* on desk-top computers comfortably, we cannot use it on small personal digital assistants (PDAs) well. *2*) Applications such as teleconference systems which always need to connect with the network do not work well, when the network connectivity is changed from wired to wireless.

To handle these situations, it is an effective way to change those applications' behavior depending on the states of runtime environments. However, since we cannot predict the states beforehand, it is nearly impossible to develop such specific applications individually. Thus, the software must have the ability which can adapt itself to the states in runtime. We call the ability *dynamic adaptability*. Dynamically adaptable software can change its own functionalities flexibly to make full use of the runtime environments' properties such as the computing power, hardware & software resources, etc.

Since we cannot obtain accurate information of runtime environments and their states beforehand, there is limitation on dynamic adaptability that we can give to the software beforehand. Thus, dynamically adaptable software must have the mechanism which can extend its dynamic adaptability easily and safely. By using the mechanism, the users must be able to improve the dynamic adaptability of the software. We aim the dynamic adaptability which includes such *extensible* mechanism.

To realize the dynamically adaptable software, we propose a software model and its description language. The dynamically adaptable software model is based on procedures. The dynamically adaptable software can change procedure invocations dynamically depending on the states of the runtime environments. Moreover, it forms a *meta-level architecture*[5]. It consists of the base-level and the meta-level. The former is the codes for primary subject domain for the software (primary codes), and the latter is the codes for dynamic adaptability (adaptation codes). Our language LEAD++ is a successor of LEAD[1] (which we had designed & implemented) with object oriented concepts similar to CLOS[4]. It supports to change procedure invocations dynamically. We can describe the dynamically adaptable software easily and systematically by using LEAD++.

The remainder of this paper is organized as follows. Section 2 gives our dynamically adaptable software model more detail. Section 3 gives an overview of the language LEAD++ and the dynamically adaptable software in LEAD++. Section 4 mentions related works. Finally, Section 5 concludes this paper.

*All correspondence may be addressed to this author.

II. THE MODEL

A. BASIC CONCEPTS

We show the key ideas for our dynamically adaptable software (DAS) model as the follows (Fig.1).

- The mechanism which realizes dynamic adaptability is based on *adaptable procedures*. When an adaptable procedure is called, the actual runtime code (we call it *adaptable method*) of the procedure is selected dynamically depending on the states of the runtime environments, and then it is invoked. We call the policy which selects a suitable adaptable method depending on the states *adaptation strategy*.
- Adaptable procedures and adaptable methods are realized as highly independent modules which do not affect any other.
- Dynamically adaptable software forms *meta-level architecture*. It consists of adaptation codes (meta-level) and primary codes (base-level). The both codes are separated completely.

Such dynamically adaptable software has the following advantages.

- We can easily realize dynamic adaptability based on procedures in application programs.
- By adding & modifying adaptable methods, we can improve the mechanism easily and safely.
- The sharing and/or reusing of adaptation codes become easier among several DASs.
- We can easily apply this approach to software in existing languages which have *procedure call*.
- The control mechanism of dynamic adaptability becomes simple, thus the costs of the control decrease.

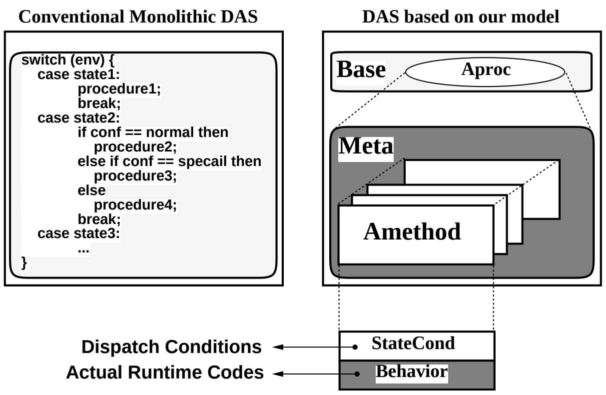

Fig. 1. Adaptable procedures and adaptable methods.

B. THE DEFINITION

The dynamic adaptability of our software model consists of *states of runtime environments*, *adaptable procedures* and *adaptation strategies*. We define them here.

Definition 1: (**states of runtime environments**) We assume that the number of the elements (*environmental elements*) in runtime environments is finite. Let N be the number of environmental elements, $Value_i$ ($i = 1, \cdots, N$) be the set of values which express the states of environmental elements. $Rstate$, the set of states of runtime environments, is defined as:

$$Rstate = Value_1 \times \cdots \times Value_N.$$

■

$Rstate$ is the set of the transitional states of runtime environments, then its element expresses a state of runtime environments at a certain moment.

Definition 2: (**adaptable methods**) $StateCond$ is the set of conditions to select an adaptable method. Let $Behavior$ be the set of actual behaviors. $Amethod$, the set of adaptable methods, is defined as:

$$Amethod = StateCond \times Behavior.$$

An element of $StateCond$ is the *dispatch condition* in an adaptable method. It is defined as:

$$StateCond = Cond_1 \times \cdots \times Cond_N.$$

An element of $Cond_i$ ($i = 1, \cdots, N$) is a condition for the ith environmental element. $Cond_i$ is expressed as the set of functions which take $Value_i$, and return 0 (*false*) or 1 (*true*).

$$Cond_i = Value_i \rightarrow \{0,\ 1\}.$$

■

Definition 3: (**adaptable procedures**) $Aproc$, the set of adaptable procedures, is the set of finite sequences of adaptable methods.

$$Aproc = Amehtod^+.$$

Note that X^+ denotes the set of finite sequences of elements of X. ■

Definition 4: (**adaptation strategies**) $Strategy$, the set of adaptation strategies, is defined as:

$$Strategy = Rstate \times Aproc \rightarrow Amethod.$$

■

An adaptation strategy is a function which takes the state of runtime environment and an adaptable procedure as the arguments, and returns an adaptable method which satisfies its dispatch condition for the state.

III. THE LANGUAGE LEAD++

A. OVERVIEW

We are currently designing & implementing the language LEAD++ to describe the dynamically adaptable software. It is an object oriented language like CLOS. The characteristic points of LEAD++ are the follows.

- classes, generic functions and methods
- meta objects and meta object protocol (MOP)
- library functions
- environmental objects
- adaptable procedures and adaptable methods

Since LEAD++ is an object oriented language like CLOS, it has classes, generic functions and methods. Our software model is based on procedures, thus it is well-suited to the object model of CLOS which has generic functions. Moreover, LEAD++ has *meta objects* as well as CLOS. They express the language constructs such as classes, generic functions, adaptable procedures, adaptable methods, etc. LEAD++ also provides the specification of them as the MOP. In LEAD++, dynamically adaptable software is realized by using the MOP.

B. LIBRARY FUNCTIONS

LEAD++ has library functions to obtain the states of *environmental elements* which consist of runtime environments. They are composed of functions which obtain the following states.

- hardware configurations
- software configurations
- processes in computers
- logical/physical surroundings of computers

C. ENVIRONMENTAL OBJECTS

Environmental objects express virtual runtime environments. Although they reflect actual states of runtime environments, their abstraction depends on the applications' and/or their procedures' one. Then, generic functions on the classes of environmental objects provide operations for the environmental objects. We can regard them the mechanism to define the abstraction of environmental objects, and/or to define the environmental elements that their abstraction is higher.

Environmental objects can be created arbitrarily in an application, and can be assigned to procedures in it. Namely, they are virtual runtime environments where adaptable procedures are executed on. Thus, the adaptation strategy is specified every environmental object. Environmental objects (E-Objects) are realized as meta objects, and their constructs are the follows.

$$
\begin{array}{lll}
\langle \textbf{E-Object} \rangle & ::= & (\langle \textit{Name} \rangle \langle \textit{Supers} \rangle \langle \textit{Strategy} \rangle \langle \textit{Slots} \rangle) \\
\langle \textit{Name} \rangle & ::= & \langle \text{Class name} \rangle \\
\langle \textit{Supers} \rangle & ::= & (\{\langle \text{Super class name} \rangle\}^*) \\
\langle \textit{Strategy} \rangle & ::= & \langle \text{Adaptation strategy name} \rangle \\
\langle \textit{Slots} \rangle & ::= & (\{\langle \text{Slot} \rangle\}^*)
\end{array}
$$

D. ADAPTABLE PROCEDURES & METHODS

In LEAD++, adaptable procedures and adaptable methods are also realized as meta objects. The constructs of adaptable procedures are the follows.

$$
\begin{array}{lll}
\langle \textbf{Aproc} \rangle & ::= & (\langle \textit{Name} \rangle \langle \textit{Env} \rangle \langle \textit{Args} \rangle \langle \textit{Amethod} \rangle) \\
\langle \textit{Name} \rangle & ::= & \langle \text{Adaptable procedure name} \rangle \\
\langle \textit{Env} \rangle & ::= & \langle \text{Environmental object} \rangle \\
\langle \textit{Args} \rangle & ::= & (\{\langle \text{Arguments for Behavior} \rangle\}^*) \\
\langle \textit{Amethod} \rangle & ::= & (\{\langle \text{Adaptable method} \rangle\}^*)
\end{array}
$$

The constructs of adaptable methods are the follows. As the actual behaviors (*Behavior*), we can describe the procedures' code, or the name of a procedure which implements the actual behaviors.

$$
\begin{array}{lll}
\langle \textbf{Amethod} \rangle & ::= & (\langle \textit{StateCond} \rangle \langle \textit{Behavior} \rangle) \\
\langle \textit{StateCond} \rangle & ::= & \langle \text{Dispatch condition} \rangle \\
\langle \textit{Behavior} \rangle & ::= & \langle \text{Proc's code} \rangle | \langle \text{Proc's name} \rangle
\end{array}
$$

E. DYNAMICALLY ADAPTABLE SOFTWARE

Dynamically adaptable software in LEAD++ forms meta-level architecture (Fig.2). The description for dynamic adaptability consists of the followings.

- meta-level description
  1. definitions of classes and generic functions for environmental objects and their operations
  2. definitions of adaptable procedures and adaptable methods by using the above 1
- base-level description
  1. instantiations of environmental objects
  2. invocations of adaptable procedures

As the description of meta-level, we define the classes which are templates of environmental objects. In the classes definition, we specify the name of adaptation strategy for adaptable procedures which are executed on the environmental objects (based on the classes). Then, we define generic functions for the classes as the operations of the environmental objects. Namely, they are functions to obtain the states of environmental objects which reflect the states of actual runtime environments. By using them, we define adaptable procedures and adaptable methods.

As the description of base-level, we describe the instantiation of environmental objects and the adaptable procedure call. The dispatching of adaptable methods depending on the states of environmental objects is realized by the meta objects (*strategy objects*) which implement adaptation strategies. The adaptation strategy is decided by the specification in the environmental object. The strategy object which implements the strategy evaluates the dispatch conditions in each adaptable method. Then, it selects the suitable one depending on the states. The selected one is executed in the base-level's context where the adaptable procedure is called.

The meta-level and base-level exist in the same address space in practice. We need not special functionalities of the language to do access the meta objects from the users' programs. We can change dynamically the specification of adaptation strategy in the environmental object from users' programs. The change of meta-level reflects the selection of adaptable methods (namely, the behaviors of the base-level's programs). Dynamically adaptable software in LEAD++ has *reflectivity*[5] like this.

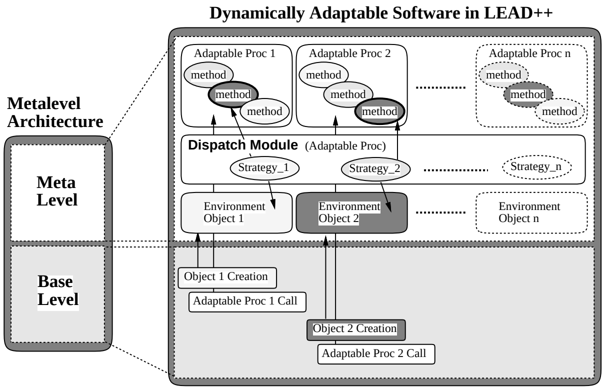

Fig. 2. Dynamically adaptable software in LEAD++.

IV. RELATED WORKS

There are several works on the linguistic approach for dynamically adaptable software: Rome[2] , Clovers[8] , Compositional Adaptation[3] Gaea[6] , AL-1/D[7], etc. Toward these related works, our software model and its languages LEAD++ have the following advantages.

- Since our model is independent of the specific description languages, we can apply it to various kinds of languages and their software easily.
- We can improve the dynamic adaptability of adaptable procedures by adding and/or modifying their adaptable methods easily and safely.
- We can customize adaptation strategies by adding and/or modifying their adaptable methods.
- We can improve the dynamic adaptability of software by changing adaptation strategies.

V. CONCLUDING REMARKS

We are currently working on the language specification and efficient implementation for LEAD++ in detail. As the platform of the implementation, we use LEAD and CLOS. Then, we consider to develop the dynamically adaptable mailer by using LEAD++. It can change its own behaviors depending on the network connectivity.

ACKNOWLEDGMENTS

We are grateful to Prof. Tatsuo Nakajima and Dr. Kazuhiro Ogata at JAIST. We also thank the members of the language design laboratory at JAIST for discussions and comments.

References

[1] N. Amano and T. Watanabe. LEAD: Linguistic Approach to Dynamic Adaptability for Practical Applications. In *Proceedings of the System Implementation 2000*, pages 277–290. Chapman & Hall, February 1998.
[2] B. Carre, L. Dekker, and J. Geib. Multiple and Evolutive Representation in the Rome Language. In *Proceedings of Technology of Object-Oriented Language and Systems (TOOLS'90)*, June 1990.
[3] Y. Honda, S. Watari, E. Osawa, and M. Tokoro. Compositional Adaptation: A New Method for Constructing Software in Open-Ended Systems. Technical Report SCSL-TR-90-012, Sony Computer Science Laboratory Inc, 1991.
[4] G. Kiczales, J. des Rivieres, and D. G. Bobrow. *The Art of the Metaobject Protocol*. The MIT Press, 1991.
[5] P. Maes and D. Nardi, editors. *Meta-level Architecture and Reflection*. ELSEVIER SCIENCE PUBLISHERS B.V., 1988.
[6] H. Nakashima. Organic Programming for Cooperative Computation. In *Proceedings of the World-Wide Computing and its Applications (WWCA'97), Lecture Note Computer Science, Vol. 1274, Springer-Verlag*, pages 133–141, March 1997.
[7] H. Okamura, Y. Ishikawa, and M. Tokoro. AL-1/D: A Distributed Programming System with Multi-Model Reflection Framework. In *Proceedings of the IMSA'92 International Workshop on Reflection and Meta-level Architecture*, November 1992.
[8] L. A. Stein and S. B. Zdonik. Clovers: The Dynamic Behavior of Types and Instances. Technical Report CS-89-42, Department of Computer Science, Brown University, 1989.