A Procedural Model of Dynamic Adaptability and Its Description Language

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
Tel: +81–761–51–1256
Fax: +81–761–51–1149
Email: {n-amano, takuo}@jaist.ac.jp

Abstract

A system has *dynamic adaptability* if it can adapt itself to dynamically changing runtime environment. 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 software with dynamic adaptability and, have designed and implemented a programming language based on the model. The basic idea is to introduce a mechanism that changes the actual runtime codes of *callee* procedure dynamically according to the runtime environments. Although the model itself is simple, it can realize highly dynamic adaptability and extensibility of the software. Moreover, the model is independent of specific programming languages.

1 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 its dynamically changing runtime environments beforehand. The changes of runtime environments have given a strong influence on the execution of the programs, which we cannot ignore. For example, *1*) Although we can use the large & luxurious application such as *Netscape* on desk-top environments comfortably, we cannot use it on poor 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, we need to change those applications' behaviors according to the runtime environments. However, it is nearly impossible to develop such specific applications individually. Since we cannot predict the runtime environments and their states' dynamic changes beforehand, the software for such systems must have the ability which can adapt itself to the dynamic changes of them 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 itself easily and safely. The users must be able to improve the dynamic adaptability of the software by using such mechanism. We aim the dynamic adaptability which includes the *extensible* mechanism.

The remainder of this paper is organized as follows. Section 2 introduces the basic concepts of our software model. Section 3 gives our languages for the model. In section 4, we discuss related works. Finally, Section 5 concludes this paper.

2 Basic Concepts

We show the key ideas for our dynamically adaptable software model as the follows.

- 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 according to the runtime environments, and then it is invoked. We call the policy which selects a suitable adaptable method depending on them *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*, therefore the description of dynamic adaptability (adaptation code) is separated from the description of primary subject domain of the software (domain code).

The adaptation codes are realized as the description of meta-level which is independent of the description of base-level (domain codes). This approach has the following advantages.

1. We can easily realize dynamic adaptability based on procedures.
2. We can add & modify adaptable methods easily and safely, thus we can also improve the mechanism.
3. The sharing and/or reusing of adaptation codes become easier.
4. We can easily realize flexible dynamic adaptability between coarse grain and fine grain by increasing and/or decreasing the codes (contents) of adaptable methods.
5. We can easily apply this approach to software in existing languages which have *procedure call*.
6. The control mechanism of dynamic adaptability becomes simple, therefore the costs of the control also decreases.

2.1 The Model

The dynamic adaptability of our software model is realized by *states of runtime environments*, *adaptable procedures* and *adaptation strategies*. We define them here. We use the word *states* of runtime environments to denote the states of the following elements;

- configuration elements on hardware,
- configuration elements on software,
- processes in a computer,
- input and/or output data on programs,
- physical and/or logical surroundings of a computer.

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 that 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 i-th 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:

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

■

Fundamentally, 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. The adaptation strategy is important factor for dynamic adaptability. Because the dynamic adaptability of software changes according to the policy of adaptation strategy. Here, we define the following adaptation strategies as the instances.

Example: (**Definitions of adaptation strategies**)
Assume that σ is the state of runtime environment at a certain moment, then an adaptable procedure ρ is called at that time. Here, $\sigma \in Rstate$, $\rho \in Aproc$, $m_i \in Amethod$, $\gamma_i \in StateCond$, $v_j \in Value_j$ and $c_j^i \in Cond_j$ $(1 \leq i \leq k,\ 1 \leq j \leq N)$.

$$\begin{array}{lcl} \sigma & = & (v_1, \cdots, v_N) \\ \rho & = & (m_1, \cdots, m_k) \\ m_i & = & (\gamma_i,\ \beta_i) \\ \gamma_i & = & (c_1^i, \cdots, c_N^i) \end{array}$$

An adaptation strategy is a policy to select a suitable adaptable method depending on the state of runtime environment from an adaptable procedure ρ. For example, we show the following strategies.

and strategy: selects an adaptable method whose dispatch condition satisfies

$$\forall j.c_j^i(v_j) = 1.$$

It selects an adaptable method which all conditions for environmental elements become *true*. If there are more than one adaptable methods which satisfy the above, this strategy selects one of them whose index number i is smallest. ■

or strategy: selects an adaptable method whose dispatch condition satisfies

$$\exists j.c_j^i(v_j) = 1.$$

It selects an adaptable method which one or more conditions for environmental elements become *true*. If there are more than one adaptable methods which satisfy the above, this strategy selects one of them whose index number i is smallest. ■

The following two strategies use weight w_j (*priority number*) for each environmental element.
positive priority number strategy: selects an adaptable method whose *positive sum*,

$$\sum_{j=1}^{n} w_j \cdot c_j^i(v_j)$$

is maximum. The *positive sum* is the sum of weights whose corresponding environmental elements satisfy the conditions. If there are more than one adaptable methods which satisfy the above, this strategy selects one of them whose index number i is smallest. ■

negative priority number strategy: selects an adaptable method whose *negative sum*,

$$\sum_{j=1}^{n} w_j \cdot (1 - c_j^i(v_j))$$

is minimum. The *negative sum* is the sum of weights whose corresponding environmental elements do not satisfy the conditions. If there are more than one adaptable methods which satisfy the above, this strategy selects one of them whose index number i is smallest. ■

These adaptation strategies are just only four examples. More important thing is that we can define various strategies like the above, and can realize flexible dynamic adaptability by their implementations easily.

2.2 Adaptation Strategies and Their Adaptability

The dynamic adaptability of our software model is simple, but it can realize sufficient adaptability by the adaptation strategies. As an example, we consider the following case. Let r be the states of runtime environment, p be an adaptable procedure, and $m_1,\ m_2,\ m_3$ be adaptable methods. Moreover, assume the followings about these adaptable methods.

$$\begin{array}{lcl} r & = & \{v_1,\ v_2,\ v_3\} \\ p & = & \{m_1,\ m_2,\ m_3\} \\ m_1 & = & (s_1,\ b_1) \quad (s_1 = (c_1^1,\ c_2^1,\ c_3^1)) \\ m_2 & = & (s_2,\ b_2) \quad (s_2 = (c_1^2,\ c_2^2,\ c_3^2)) \\ m_3 & = & (s_3,\ b_3) \quad (s_3 = (c_1^3,\ c_2^3,\ c_3^3)) \end{array}$$

Since we cannot obtain accurate information of the states of runtime environment beforehand, the states which we can predict are limited. Thus, the adaptable methods (for the states) which we can define beforehand are also limited. Assume $m_1,\ m_2,\ m_3$ such as adaptable methods, and the following strategies.

1. ***strategy*$_1$**: selects an adaptable method which all conditions for environmental elements become *true*.

2. ***strategy*$_2$**: selects an adaptable method which all conditions for environmental elements become *true*. Moreover, if there is no suitable one, this strategy selects m_1 as the default one.

3. ***strategy*$_3$**: selects an adaptable method which part of conditions for environmental elements become *true*.

Consider the following states of runtime environment.

	θ_1	θ_2	θ_3
$state_1$	(1, 1, 0)	(0, 0, 0)	(0, 0, 0)
$state_2$	(0, 0, 0)	(1, 1, 0)	(0, 0, 0)
$state_3$	(0, 0, 0)	(0, 0, 0)	(1, 1, 0)

Let $\theta_1,\ \theta_2,\ \theta_3$ be as the followings.

$$\begin{array}{lcl} \theta_1 & = & (c_1^1(v_1),\ c_2^1(v_2),\ c_3^1(v_3)) \\ \theta_2 & = & (c_1^2(v_1),\ c_2^2(v_2),\ c_3^2(v_3)) \\ \theta_3 & = & (c_1^3(v_1),\ c_2^3(v_2),\ c_3^3(v_3)) \end{array}$$

The dynamically adaptable software using $strategy_1$ cannot adapt itself to all the above states ($strategy_1$ cannot select any adaptable methods). The $strategy_2$ always selects the default adaptable method m_1 for all the above states. However, the $strategy_3$ can realize more desirable dispatching as follows.

$strategy_1$	$: (state_1,\ p) \rightarrow$
	$: (state_2,\ p) \rightarrow$
	$: (state_3,\ p) \rightarrow$
$strategy_2$	$: (state_1,\ p) \rightarrow m_1$
	$: (state_2,\ p) \rightarrow m_1$
	$: (state_3,\ p) \rightarrow m_1$
$strategy_3$	$: (state_1,\ p) \rightarrow m_1$
	$: (state_2,\ p) \rightarrow m_2$
	$: (state_3,\ p) \rightarrow m_3$

Considering from the above assumption, the $strategy_3$ is more desirable than $strategy_{1,2}$. We can say that the $strategy_3$'s adaptability is higher than the $strategy_{1,2}$'s one. However, the $strategy_{1,2}$ may be more desirable than $strategy_3$ according to the properties of actual behaviors. Namely, it is able to realize the dynamic adaptability in flexible way by changing adaptation strategies. In our model, adaptation strategies are realized as an adaptable procedure. We can change them dynamically according to the states of runtime environments, and can realize highly & flexibly dynamic adaptability easier. Moreover, we can describe adaptable procedures as the behaviors of adaptable methods, and can realize the nested adaptable procedures.

2.3 Separation of Concerns

Dynamically adaptable software consists of adaptation codes (adaptable procedures including adaptation strategies) and domain codes (including adaptable procedure call). If it is realized as the monolithic structure such as Fig.1-(a), the both codes are mixed and hard-coded. The mixture and hard-coding of them occur the following problems.

- We cannot have enough readability and extensibility of the software. Thus, the maintenance of the software becomes harder.
- The sharing & reusing of adaptation codes also become harder.

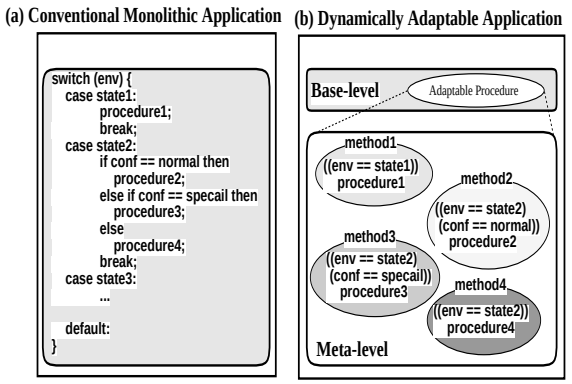

Figure 1: Separation of Concerns

In our approach, dynamically adaptable software forms *meta-level architecture* such as Fig.1-(b). The adaptation codes (meta-level) are separated from domain codes (base-level) completely. Not only this structure solves the above problems, but also it makes easy & safe extension of the adaptation codes possible. Moreover, thus the adaptation codes consist of adaptable procedures, the readability, extensibility and maintainability of them also improve. We can improve the dynamic adaptability by adding and/or modifying adaptable procedures & adaptable methods. The improvement is easy & safe, because the adaptation codes are meta-level separated from the base-level, and adaptable procedures & adaptable methods are independent modules without depending any other ones.

That adaptation strategies are realized as an adaptable procedure also contributes greatly to the improvement of dynamic adaptability of the software. By changing adaptation strategies, we can improve the dynamic adaptability of the software. Namely, we can add & modify the strategies as adaptable methods easily. Even if we modify & change adaptation strategies, we need not modify & change any other adaptable procedures and their methods. By changing adaptation strategies only, we can realize different behaviors of the same software in flexible way. Thus, the dynamic adaptability in our software model is realized as the structure which we can extend it easily and safely.

3 Description Languages

We need the description language to realize dynamically adaptable software. We may be able to use existing languages to realize it. However, if we use them, we must be more careful and laborious. They have no mechanism to separate adaptation codes from domain codes, and to realize adaptation codes as adaptable procedures and adaptable methods which can be extended, shared and reused easily & safely. However, we can use existing languages by extending them to realize our software model. Our software model is independent of specific description languages.

3.1 LEAD

We had designed the language LEAD to realize dynamically adaptable software, and had implemented the prototype of LEAD as an interpretive language which is an extension of *Scheme*. We had also implemented some dynamically adaptable applications in LEAD. LEAD provides almost mechanisms required to realize dynamically adaptable software. Although the prototype is an extended *Scheme*, we can describe the behaviors of adaptable methods in other existing languages. Namely, using LEAD, we can easily introduce dynamic adaptability into software in existing languages. Dynamically adaptable software in LEAD forms meta-level architecture and has *reflectivity*. Thus it can realize dynamic adaptability in flexible way. As the characteristic functionalities of LEAD (see [1]), there are remote execution of adaptable methods depending on the states of runtime environments, asynchronous events which are informed the applications as the occurrence of the states' changes asynchronously, etc.

3.2 LEAD++

We are currently working on the language LEAD++ which introduces the object oriented concepts such as CLOS[4] into LEAD. In a dynamically adaptable application in LEAD, the runtime environment is only one for it and behaviors in it. Thus, we cannot introduce & describe flexible dynamic adaptability according to the abstraction of each behavior in the application. This problem is great restriction to realize dynamically adaptable software in practice, because the situation that the abstraction of each behavior in an application differs is nothing remarkable in practical software. Dynamically adaptable software in LEAD++ can create several *environment objects* which express the virtual runtime environments, and can assign them for each behavior in the software according to its abstraction. We can define an environment object's abstraction according to a behavior's one in software. The operations for environment objects are defined as the generic functions for their classes. The states of runtime environments reflect the states of environment objects. Therefore, we can solve the problem of LEAD completely, and can realize dynamically adaptable software in more flexible way.

Since our software model is based on procedures, it is well-suited to the object model such as CLOS's

one which has generic functions and their methods. In LEAD++, *meta objects* which express the language constructs (classes, generic functions, methods, adaptable procedures, adaptable methods, etc.) as their own objects are introduced. Moreover, LEAD++ provides the specification of meta objects as *meta object protocol* (MOP)[4]. Dynamically adaptable software in LEAD++ is realized by using the MOP.

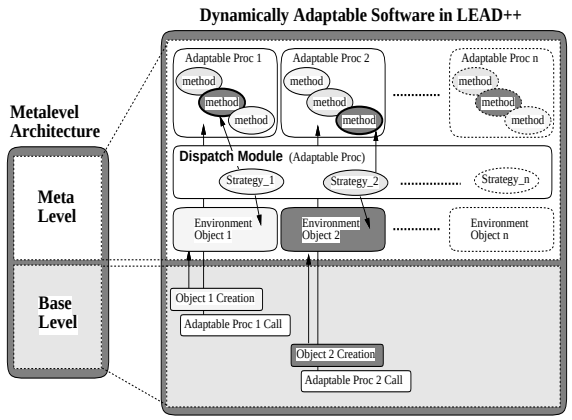

Figure 2: Dynamically Adaptable Software in LEAD++

Dynamically adaptable software in LEAD++ forms meta-level architecture (Fig.2). As the description of meta-level, we define the class which is a template of environment object. In the class definition, we specify the name of adaptation strategy for adaptable procedures which use the environment object (based on the class) as the virtual runtime environment. Then we define operations of the object as generic procedures for the class. Namely, they are functions to obtain the states of environment objects which reflect actual runtime environments. By using them, we define adaptable procedures and adaptable methods in meta-level.

As the description of base-level, we describe the instantiation of environment objects and adaptable procedure calls. The dispatching of adaptable methods depending on the states of environment objects is realized by the meta object (strategy object) which implements adaptation strategies. The adaptation strategy is decided by the specification in the environment object. The strategy object which implements the strategy evaluates the dispatch conditions of each adaptable method, and then 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 environment object from users' programs. The change of meta-level reflects the selection of adaptable methods (the behaviors of the base-level's programs). Dynamically adaptable software in LEAD++ has *reflectivity* like this.

4 Related Works

There are several works on the linguistic approach for dynamically adaptable software: Rome[2] , Clovers[7] , Compositional adaptation[3] Gaea[5] , AL-1/D[6], etc. Our software model and its languages (LEAD, LEAD++) have the following advantages. *1*) Since it is independent of the specific description languages, it can be applied various kinds of languages and their software easily. *2*) We can improve the dynamic adaptability of adaptable procedures by adding and/or modifying their adaptable methods easily and safely. *3*) We can customize adaptation strategies in flexible way. *4*) We can improve dynamic adaptability by changing adaptation strategies only.

5 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[4]. For future work, we also plan to apply our software model to mobile agents in Java and/or Telescript, and to realize dynamically adaptable mobile agents.

References

[1] N. Amano and T. Watanabe. Lead: Linguistic approach to dynamic adaptability for practical applications. In *Proceedings of the System Implementation 2000 Conference*, February 1998. (to appear).

[2] B. Carre, L. Dekker, and J.-M. 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] H. Nakashima. Organic programming for cooperative computation. In *Proceedings of the World-Wide Computing and its Applications (WWCA'97)*, pages B–1–3–1–8, March 1997.

[6] 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.

[7] 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.