A Framework of a Visual Language with Dynamic Specification

Jun'ichi Miyao  
Faculty of Integrated Arts and Sciences  
Hiroshima University  
Higashi-Hiroshima, 739, Japan  
miyao@mis.hiroshima-u.ac.jp

Shi-Kuo Chang  
Department of Computer Science  
University of Pittsburgh  
Pittsburgh, PA 15260, U.S.A.  
chang@cs.pitt.edu

Abstract

Behavior of complex software may be changed depending on external environment such as user profile, available resources, input data, time, and so on. Design of such dynamic software in conventional languages needs extra effort to keep consistency among behaviors for all states. In this paper, we propose an object-oriented visual language for dynamic software and its visualized specification with dynamics presentation. The language supports dynamic behavior according to both time-dependent and time-independent external states based on dynamic objects. In the visualized specification, we present uniform dynamics presentation for all elements in the language based on three concepts of snapshot, overlap, and sweep.

1 Introduction

Recently, software with dynamic behavior has become popular in many fields. For example, multimedia systems change its presentations dynamically according to media types. Also, database systems have dynamic queries which display query results continuously reflecting database contents [8].

In this paper, we present a framework of a fully dynamic visual language with dynamic specification by an object-oriented approach. We will introduce dynamics into data structures and program structures. They are specified by uniform visual presentation based on snapshot, overlap, and sweep. An example will be illustrated in Figure 9.

Previously, there are many contributions on program dynamics. However, their dynamics are not program specification itself, but dynamics of program execution as in [5]. We propose a visual language with dynamic specification which is changed according to external environment with a time dimension.

2 Conceptual Framework for Dynamic Language based on Objects

We will consider a language which has a dynamic specification that is varied according to an external environment, and which has a visual specification.

2.1 Dynamics in Programming

The dynamics in programming are divided into program specification, and program execution. The dynamics of program execution is caused by program itself and/or interactive inputs. Thus, this dynamics is realized in ordinary languages. However, it is hard to find another dynamics in text-based languages and visual languages.

Let us consider three sets: an external environment \( E \), a dynamic programming space \( P \), and a static programming space \( S \). \( P \) and \( S \) consist of all possible programs with dynamics and without dynamics, respectively. The external environment is a state from a set of external states \( E = \{ e_1, e_2, \ldots \} \). In general, \( E \), \( P \) and \( S \) are infinite, and for given \( p_i \) and \( e_k \), a static
program $s_f$ is determined. Then, the dynamics of a dynamic program is caused by its external environment, and defined by a mapping $\text{snapshot}: P \times E \rightarrow S$. The semantics of the program $p$ is varied according to its environment $c_k$ or $c_k$.

2.2 Concepts of Dynamic Object

Let us consider a program which consists of objects, and of which execution is performed by message passing among objects. Based on this structure, we will introduce dynamics into objects.

A program is defined by a set of dynamic objects’ definition, and it is executed by instances of the objects. Conceptually, a dynamic object consists of three parts: a set of messages, a set of methods, and a set of abstract data (see Figure 3). This is the same as ordinary object oriented languages. However, each of them has dynamic nature.

There exist ordinary abstract data and dynamic data called dynamic constants, of which values are varied according to the external states of environment. Its value is given by dynamic function attached to it. Next, a program structure is also dynamic since acceptable input messages and creatable output messages are determined by their logical expressions which contain dynamic constants. Furthermore, each method has dynamics since it can use dynamic constants in it, and its input and output messages are controlled by logical expressions. Thus, the dynamic object is totally and essentially dynamic.

In order to specify a dynamic object, a special method and environment must be prepared because of its dynamics. This is discussed in Section 4.

3 Formal Definition of Dynamic Language

Definition 1 Let us consider a finite set of raw external states $R = \{r_1, r_2, \ldots, r_n\}$ which are independent of time. Then, a set of external states $E$ is defined by $R \times T = \{< r_1, t_0 >, < r_1, t_1 >, \ldots, < r_2, t_0 >, < r_2, t_1 >, \ldots \}$ where $T = \{t_0, t_1, \ldots\}$ is an infinite set of time, which is defined by countable time unit. Thus $E$ is an infinite and countable set. If $T$ is finite then $E$ is called a finite external states denoted by $E_f$.

Definition 2 Let us consider two types of total orderings on the external states: state-time and time-state (see Figure 1).

![Figure 1: Two types of total ordering.](image)

- **state-time**: Fix each raw state, then enumerate states in ascending order of time, i.e., $< r_1, t_0 >, < r_1, t_1 >, \ldots, < r_2, t_0 >, < r_2, t_1 >, \ldots$.
- **time-state**: Fix each distinct time, then enumerate raw states in ascending order of their indices, i.e., $< r_1, t_0 >, < r_2, t_0 >, \ldots, < r_n, t_0 >, < r_1, t_1 >, \ldots$.

Definition 3 A dynamic constant $x$ is defined by its domain denoted by $\text{DOM}(x)$ and a dynamic function $f$ as follows: $f : E \rightarrow \text{DOM}(x)$.

Definition 4 A message consists of 3-tuple $(1D, OR, PR)$

- $1D$: a message type which identifies the message,
- $OR$: an originator of the message, and
- $PR$: parameter(s) of the message.

Next, we will show a definition of a dynamic object.

Definition 5 A dynamic object is defined by 5-tuple $(1M, OM, MD, DC, AD)$

- $1M$: a set of input messages, each of which has a logical expression with at least one dynamic constant. The logical expressions define preconditions of input messages.
- $OM$: a set of output messages, each of which has a logical expression with at least one dynamic constant. The logical expressions define postconditions of output messages.
- $MD$: a set of methods, each of which receives at least one message with dynamic constant and outputs at least one message with dynamic constant.
- $DC$: a set of dynamic constants. Each element $x$ of $DC$ has a dynamic function $f_x : E \rightarrow \text{DOM}(x)$ where $\text{DOM}(x)$ is a domain of data type $x$.

---

2It is noted that the state time ordering is defined on only the finite external states $E_f$.

3The preconditions and the postconditions of messages enable consistency checking of message passing among objects.
• AD: a set of abstract data. □

Definition 6 A dynamic program is defined by 3-tuple \((ME, OB, IO)\),

• ME: a set of messages.
• OB: a set of dynamic objects.
• IO: a subset of OB which defines instantiated objects initially. □

In order to explain the dynamic language with its concept, we show a simple example. Though the example does not contain dynamic instantiation of dynamic objects to make explanation simple, the dynamic language supports such ability by IO in Definition 6, and dynamic instantiation from methods.

[Example 1] Let us consider a terminal program for POS (Point of Sales) registers in a store. The register is used in two departments: Food and Electronic Products. The functions of the register are varied depending on where it is used at, since the Electronic Products require warranty processings. Thus, the function must be changed according to department where it is used. Additionally, the store has a discount sale in the morning. Then, the discount processings must be done in the morning time 9am to noon.

An example of the program is shown in Figure 2, and objects “elect_dept” and “discount_calc” are shown in Figures 3 and 4, respectively. There are 7 dynamic objects and 8 types of messages. The external states \( E \) consists of a set of raw states \{Food, Electronic\} and time \( T \). Then, \( E = \{<\text{food}, t_0 >, <\text{food}, t_1 >, \ldots , <\text{elect}, t_0 >, <\text{elect}, t_1 >, \ldots \} \).

In Figure 3, there exists a dynamic constant “elect” which has a dynamic function \( f_{elect} \) that has a boolean value “true” if the external state is \(<\text{elect}, * >\) where * shows a wildcard of the entry. In this object, all messages are enabled when dynamic constant “elect” = “true”. Figure 4 shows an object “discount_calc” which is invoked by the time interval from 9am to 12pm. There exists a dynamic constant “disc_time” with a dynamic function \( f_{disc\_time} \) where its value is true if \( e \in \{< x, t > \mid 9am \leq t \leq 12pm \} \).

The relationships among objects are defined by acceptable messages. The dynamics of this program is shown in Figure 5. There exist time depend and time independent dynamics. The former is related to discount sale in the morning, and the latter is related to department. This program is varied its contents depending on external environment. □

![Figure 2: An example of a POS terminal program.](image)

![Figure 3: An object “elect_dept” in the POS terminal program.](image)

4 A Visualized Specification of Dynamic Language

In this section, we will discuss visualization of dynamics of the language. A uniform and consistent method, which presents dynamics of all levels such as message passing, dynamic objects and dynamic constants, is proposed.

4.1 Visualization of Dynamics

In the visualization of dynamics, we define the following terms.

Definition 7 Three subsets \( I_1, I_3, \) and \( I_L \) of an external environment \( E \) are defined as follows:

• \( I_e \): an element \( \{e_i \mid e_i \in E \} \).
• \( I_s \): a subset of \( E \)
• \( I_L \): a list of elements of \( E \) with sequential order such as \( < e_1, e_2, e_3, \ldots , e_n > \). The order is determined according to the state-time ordering or the time-state ordering. □
Step 1: Collect conditional statements in dynamic constants.

Step 2: Investigate boundary states for each of the conditional statements, where a boundary state is a state to change the truth value of the condition.

Step 3: Make the interval graph for states which give true value for the statements.

Step 4: Get each 1-com-max by extracting states between an interval which has no boundary time in it. In this operation, there may be more than one intervals which belong to the same 1-com-max. In order to detect such duplicate intervals, a pattern matching algorithm is applied.

Step 5: Get 1-dis-max by extracting exactly one element from each 1-com-max interval.

Let us consider the computational complexity of the above algorithm. It depends on the number of boundary states obtained in Step 2. Then, we discuss the complexity from Step 3 to Step 5. At Step 3, it takes $O(m \times n)$ where $m$ and $n$ are the number of messages and boundary states, respectively. The steps 4 and 5 do not exceed the complexity. Thus, it is $O(m \times n)$.

We will explain the algorithm by an example

[Example 2] By using Algorithm 1, the 1-dis-max for messages in Example 1 is obtained as follows. In the following, we assume state-time total order, and finite external states for $9am \leq t \leq 10pm$.

In Step 1, the following conditions are collected.

- norm_time: true if $e \in \{< s, t > | t < 9 \text{ or } t > 12 \}$
- disc_time: true if $e \in \{< s, t > | 9 \leq t \leq 12 \}$
- elect: true if $e = < elect, s >$
- food: true if $e = < food, s >$

In Step 2, boundary states for each conditions are investigated. Then, the following boundary states are obtained.

- norm_time : $< elect, 9 >, < elect, 12 >, < food, 9 >, < food, 12 >$
- disc_time : $< elect, 9 >, < elect, 12 >, < food, 9 >, < food, 12 >$
- elect : $< elect, 9 >$
- food : $< food, 9 >$

Next, in Step 3, an interval graph for true conditions is constructed as in Figure 6.

In Step 4, the following 1-com-max is obtained based the interval graph:

- $1-com\_max_1 = \{< food, t > | 9 \leq t \leq 12 \}$
Figure 6: Interval Graph for true conditions.

- \( I - \text{com} - \text{max}_1 = \{ \text{<food, } t \text{> } | 12 < t \leq 10 \} \)
- \( I - \text{com} - \text{max}_3 = \{ \text{<elect, } t \text{> } | 9 < t \leq 12 \} \)
- \( I - \text{com} - \text{max}_4 = \{ \text{<elect, } t \text{> } | 12 < t \leq 10 \} \)

Then, in Step 5, the following I-dis-max is obtained by extracting exactly one element from each I-com-max.

\( I - \text{dis} - \text{max} = \{ \text{<food, } 9 \text{>}, \text{<food, } 1 \text{>}, \text{<elect, } 9 \text{>}, \text{elect, } 1 \} \quad \square \)

Next, we will propose uniform visualization method based on above subsets.

- Snapshot is a visual presentation which corresponds to a given external state for \( I_e \).
- Overlap is a visual presentation which is obtained by overlapping a set of snapshots for \( I_S \).
- Sweep is an animation of snapshots each of which corresponds to an element in \( I_L \).

This method can be applied to all levels of a program such as a relationship among objects, dynamic objects and dynamic constants. Furthermore, the concepts of I-com and I-dis are utilized to acquire program behavior from a lot of external states. In addition, the two total orderings such as state-time and time-state give different visualization vectors for the sweeping. The sweeping based on state-time or time-state is referred to as a state-sweep and a time-sweep, respectively. We will illustrate this method in the next example.

Example 3: Visual presentation for each level is realized as follows:

1. Dynamics of message passing: Let us consider that message passings are presented by directed arcs among objects. Then, Figure 5 (a) – (d) show snapshots for \( I_e = \{ \text{<food, } 9am \text{>}, \text{<food, } 1pm \text{>}, \text{<elect, } 9am \text{>}, \text{<elect, } 1pm \} \). Moreover, the four states constitute I-dis-max. Next, Figure 2 shows an overlap for \( I_S = \{ \text{<food, } 9am \text{>}, \text{<food, } 1pm \text{>}, \text{<elect, } 9am \text{>, \text{<elect, } 1pm \} \). Next, if pictures (a) through (d) in Figure 5 are presented one by one in this order as an animation, then this realizes a state-sweep for \( I_L = \{ \text{<food, } 9am \text{>, \text{<food, } 1pm \text{>, \text{<elect, } 9am \text{>, \text{<elect, } 1pm \} \). (2) Dynamics of an object: A snapshot of an object is presented by showing acceptable messages and executable methods decided by their input messages and current values of dynamic constants. The overlap and sweep are the similar to the message dynamics.

3. Dynamics of a dynamic constant: Figure 7 (a) shows three presentations for the dynamic constant “elect”. A value of the constant is either “true” or “false” by the dynamic function \( f_{\text{elect}} \) in Example 1. We can use graphical presentation by a circle and a cross for “true” and “false”, respectively. Then, a snapshot shows either a circle or a cross, and overlap shows overlapping of them, and sweep shows them alternately. This illustrates their dynamics associated with external states intuitively as in Figure 9(b).

We will illustrate another example for non-boolean value. Let us consider a dynamic constant \( x \) with a dynamic function \( f_x(t) = \sin(2\pi t) \). Then its snapshot is represented by a point in a graph (see Figure 7(b)). Its overlap for \( I_S = \{ t | 0 \leq t \leq 1 \} \) is presented as (b-2), and a sweep is represented as a moving point between the interval. \( \square \)

4.2 Overview of Language Specification

As mentioned before, a dynamic program is defined by a set of messages, and a set of objects. It is necessary to specify and present them in a visual and con-
sistent manner. In [4], it is pointed out that zooming is effective for large software. The similar approach called fishCEO is proposed in [7]. Based on these approach, we will consider specification of the dynamic language as follows.

Figure 8 shows an editor of the program specification. It consists of a dynamic language editor and an object editor. The former defines object’s names, and the latter defines inside of objects in detail. Then, they can define programs in the dynamic language.

However, there does not exist dynamics in this representation. Then, it is hard to recognize its dynamics. Thus, we propose a special tool to present dynamics visually. Figure 9 shows the tool called a dynamic viewer. If it is overlapped on objects in the editor window, then dynamics is presented as shown in the figure. The presentations correspond to definitions of subsets of external environment in Definition 6.

At the level of a program, it shows relationships among objects in every external states. At the level of methods and dynamic constant, it shows dynamics of methods by displaying executable methods at the moment, and dynamics of the dynamic constant by displaying its value dynamically. Here, the constant is represented by a circle and a cross which correspond to its values “true” and “false”, respectively. In (1-3) of Figure 9, methods are not displayed since their input messages are not acceptable when elect=“false” (represented by a cross).

4.3 Dynamic Description

In this subsection, we will discuss how to specify dynamic functions for dynamic constants with real-time behavior. Let us consider a window opened from an icon in GUI. If the opening of the window begins at slow speed and accelerates gradually, and finishes slowly, then the presentation quality of GUI is increased. Control of speed is important in GUI and multimedia computing.

Basically, the specification of speed can be given by using dynamic constants in the proposed language. Furthermore, the sweep presentation of the dynamic viewer enables real-time presentation as it is. This realizes WYSIWYG with timing dimension.

However, we must resolve some problems. One problem is relative time. In the definition of the external environment, time represents an absolute time(clock) in the system. In order to represent the window open speed by the dynamic function, relative time started at when the open function is called. Another problem is how to specify dynamic behavior. This is realized by using predefined mathematical functions or a numeric table. However, it needs heavy effort to create and enter them, and is not easy-to-use for end users.

For the first problem, we will introduce a new time variable $t_r$ which represents the instantiating time of an object. Then, a relative time is given by $t - t_r$ where $t$ is the absolute time.

In order to resolve the second problem, we will propose a dynamic description which accepts real-time motion from users. Examples of real-time motion of users are mouse cursor movement, key-click tempo, gesture, etc.

Figure 10 shows a tool to realize dynamic description from mouse cursor movement. In the tool, continuous velocity and acceleration of a mouse cursor from a start line to an end line are measured. The result is displayed by using static graphs (velocity and acceleration) or real-time movement of an object.

The concept of dynamic description can be applied to many applications such as game, music system, education, virtual reality. Especially, gestures in the virtual reality is an interesting application of this concept.
(a) Visual representation of objects by dynamic viewer.

(b) Visual representation of methods and dynamic constant by dynamic viewer.

Figure 9: Presentation of dynamics by dynamic viewer.
3. Dynamic linkings between dynamic objects and messages should be introduced.
4. Concerning visualization of dynamics, effective utilization of color should be considered.
5. Implementation of real-time operations should be considered, for both UNIX systems and special purpose embedded real-time systems.

References


5 Conclusion

We have presented a visual language with dynamic specification and visualization of dynamics in it. The language is totally dynamic from program structure defined by message passing to each datum. The proposed visual presentation is uniform and consistent for all levels of the language based on the concepts snapshot, overlap and sweep. Additionally, in order to describe real-time behavior, dynamic description by measuring mouse cursor movement is also introduced. The dynamic behavior of active index can be specified and visualized using this methodology [1].

We will enumerate future research items as follows.
1. Effectiveness of the proposed language should be shown by empirical studies using practical examples.
2. Integrated programming environment with a debugger and a simulator for the language should be designed.

Figure 10: Dynamic description tool.