Browsing and Editing Tool for Programming in Pictures

YUTAKA WATANOE, NIKOLA MIRENKOV and RENTARO YOSHIOKA
Department of Computer Science and Engineering
University of Aizu
Aizu-wakamatsu, Fukushima 965-8580
JAPAN
yutaka@u-aizu.ac.jp, nikmir@u-aizu.ac.jp, rentaro@u-aizu.ac.jp

Abstract: A tool that enables programming in pictures based on Filmification of Methods (FIM) is presented. FIM is an approach where pictures and moving pictures are used to specify and program computational algorithms. The key contribution of the tool is the integration of sub-systems for organizing a visual programming environment in which end-users can easily access, through web browsers, special libraries of software components and “algorithmic characters” and create their own components. Using the tool, the users are able to retrieve library components, browse and edit them, generate executable codes, execute them to observe computational results, and upload and share the components. To facilitate the processes of understanding and manipulations with components, the tool is based on a few visual languages and appropriate interfaces. In addition, the tool allows to export the programs in pictures in several formats, including Adobe Flash animations, PNG or JPEG images to create, for example, educational materials. The scalability of the library of algorithmic characters and the library of software components are also discussed.

Key–Words: Programming in pictures, Filmification of Methods, CyberFilm

1 Introduction

With evolution of Web-related technologies, web sites have become sources of information as well as platforms that enable users to create new information resources [1]. For example, browser based applications such as Google docs, Gmail and Blog software have become especially popular. They are more viable and offer several advantages over traditional software: 1) they don’t have to be installed on each user’s machine, 2) they don’t need to be upgraded by the users, 3) they are platform-independent, and so on. In the area of software engineering, there are also on-line environments where users can develop programs in web browsers (see, for example [2]). Such environments allow the users to edit their codes, run applications, and share the products for other users to try. They are much more convenient than typical IDEs such as Eclipse which need to be downloaded and installed, and then compiled and executed on user’s machines. Although, there are no doubts that more powerful and flexible browser-based IDEs will come out, there is little hope for breakthrough in the programming production regarding human performance. Unfortunately, it is still difficult to find browser-based programming tools with a highly interactive visual interface and the corresponding programming languages. This is in spite of considerable activities on various methods and tools for human-computer interfaces, visual languages and end-user programming [3, 4]. On the other hand, if one wants to demonstrate and disseminate a new programming technology (and use it for educational purposes, for example), web-based technologies can be one of natural solutions.

Our goal is to provide a programming environment to promote Filmification of Methods (FIM) and enable end-users to create information resources by means of this approach where Algorithmic CyberFilms are applied to represent and specify computational algorithms as “activities” in different space-time structures [5, 6, 7, 8, 9, 10, 11, 12, 13]. So far, a large set of cyberFilms has been developed and evaluated to brush the visual language constructs, and theoretical basis to generate executable codes from cyberFilm specifications has been established [5, 6]. In this paper, a tool (or an environment) including browsers and editors to promote Filmification of Methods is presented. It is based on a client-server architecture and makes it possible to download/upload cyberFilms from/to the server, browse and edit them through the client environment. The client environment works within web browsers which give the end-users much more opportunities to find an appropriate library item, understand it and edit to create a new one (in principle, it is available as a stand-alone application with a special network connection). The tool facilitates un-
nderstanding software components and “programming by examples” processes through applying 1) four mutually supplemented visual languages to define different algorithmic features of a component, 2) programming in a rather compact view with embedded clarity annotations and 3) intelligent assistances to suggest elements which should be selected in editing processes and keep the compatibility of program elements. The tool also allows to generate executable codes in C++ from cyberFilm specifications after editing operations. The code generation is accomplished in the server side with prepared template programs, which are hidden from the end-users (unless they are provided as educational materials). The end-users can obtain the generated codes and observe computational results, but what they see in the programming processes is mainly pictures and moving pictures organizing algorithmic characters and constructs. The programming in pictures is principally performed by direct manipulation of the characters and constructs through using the mouse (or touch pens) including clicks and drag-and-drop operations. In this paper, the applicability of the approach to educational activities is shown by a set of the library components related to geometric and graph algorithms taught in the department of computer science and engineering. In addition, the scalability of the algorithmic characters set and the library of the components is also discussed.

The rest of this paper is organized as follows. In Section 2, related works are considered. In Section 3, an overview of the Algorithmic CyberFilm is presented. In Section 4, a system architecture of the tool is presented. In Section 5, the browsers and editors are demonstrated. In Section 6, the scalability of libraries is discussed.

2 Related Work

A large amount of visual programming systems and languages as well as methods for algorithm visualization have been developed over the past dozen years or so [3]. Recently, this type of activities is more and more related to browser based applications. Leonardo Web [14] and Lily [15] are examples of such applications. Leonardo Web is a collection of tools to animate algorithms, which can be useful for teaching, disseminating and e-learning. They are based on visual editors and each of them is available as an applet. Lily is a visual programming environment that lets users create programs graphically, without writing code, by drawing connections between data, images, text and graphics. It is a mozilla based visual programming environment written in Javascript, so it is possible to edit programs online. The main difference between these systems and the FIM approach is that 1) they use concrete data to explain algorithms, 2) they are not based on systematically defined language systems and 3) they do not provide practical libraries for algorithmic solutions. Although there are many projects to provide easy access to libraries of efficient and reliable algorithms (see, for example [16]), libraries items themselves are black boxes based on conventional text based programming languages.

In terms of on-line tools for solving mathematical problems, WebMATH [17] is a representative example of web sites that generate answers to specific questions and problems. WebMATH provides a real time answer immediately after a web user inputs his/her problem. Several mathematical expression editors were also developed in the past ten years. For example, MathEdit [18] is a browser-based visual mathematic expression editor, and GeometryEditor [19] is a Web-based interactive tool for drawing geometry graphics such as points, lines and polygons. These tools were developed in JavaScript and DOM to run in regular browsers, and are applied to E-Science systems.

In the field of end-user programming development, programming by example (PBE) [20] is a technique where the user demonstrates a sequence of operations in an interactive interface. The system acquires user actions and infers a generalized program.

Among other trends, importance of specialized programming languages is getting additional attention. An example of such languages is Kodu [21]. It is a free “visual programming language” (developed for Xbox360 and Windows PCs) where kids-oriented programming concepts are introduced by a video-game format.

3 Overview of Programming in Pictures

Our approach uses pictures and moving pictures to more directly represent features of computational algorithms and data structures. Within this approach some “data space” are traversed by a “front of computation” and necessary operations (formulas and functions) are performed during this traversal process. An algorithm is represented by a set of series of frames. A frame is to represent a view (a feature) of an algorithm. A series of frames is to represent a number of features in multiple views. A number of features presented by multimedia frames are assembled into a structure which is called cyberFilm. The series of frames are classified into four groups representing 1) algorithmic skeletons that show space data structures and temporal schemes of computational flows.
on these structures, 2) variables and formulas (operations) that are attached to the space-time points of the skeletons, 3) input/output operations that define the algorithmic interface with external world, and 4) a compact combination of main features of the groups 1)-3) organized in an integrated view. An appropriate series of corresponding frames is extracted and presented through the right editor/browser panels at the right time. Independent visual languages are used to represent the different features. As a result, the programming environment for making cyberFilms is a cluster of four mutually supplemented languages to define four different views of each software component. Each language uses its own set of self-explanatory pictures and super-characters. In these languages, open sets of visual constructs and super-characters are used to define the contents of the cyberFilm frames and their translation into executable codes. In addition to the constructs and super-characters, special background pictures and symbols, as well as auxiliary pictures are used to play a role of embedded clarity annotations for variables, operations, cyberFrames, cyberScenes and cyberFilm as a whole. A cyberScene is a group of frames which represents a unit of computational flows for formula attachment. For each cyberScene, a set of template programs which are organized by nested loops to represent the flows, is prepared. End-users do not create frames of algorithmic skeletons, but take and assemble them from the libraries.

So, the programming is reduced to defining size (and configuration) of the space structures, declaring variables on the structures and specifying operations on traversal nodes, including input/output operations. It is important to note that taking cyberFilms (cyberFrames) from the libraries and edit them to create new items are fundamental features of the approach, that can be considered, in a great part, as programming by examples [20].

4 System Architecture and Components

The programming environment is based on a client-server architecture as shown in Figure 1. In this section, major components in the client and server sides and an internal organization of the environment are presented.

4.1 Client Side

The client side is based on the following components. Administrator’s tools include system consoles to manage a database, special uploaders to store resources which end-users should not be involved (for example, skeleton frames and the corresponding template programs). The tools also include modelers consisting of a set of ad hoc programs and skeleton editors
to generate skeletons frames.

**Film Environment** is the main part of the client for end-users who work within web browsers. It has been designed to be an easy-to-access client where the users do not need to install any software other than Flash Player. The environment was developed in Actionscript 3.0 including CSS technologies. The browsers/editors provide functions which enable the users to manipulate visual constructs and symbols for programming, and the consoles/visualizers show compilation errors, runtime errors, executable codes and the corresponding computational results. The client communicates with the server side based on http requests and XML formats. A major task for the client environment is to parse the internal formats of cyberFilms and their contents presented in XML formats and then to convert them into frame-by-frame visual representations, and vice versa. The client is a rather light application where limited functions are implemented to provide interactive user interfaces.

**Stand-alone type application** is a coproduct in the development of the environment. AIR (Adobe Integrated Runtime) technology makes it possible to provide the conventional stand-alone application of the environment which has a special network connection to the server.

### 4.2 Server Side

The server side is based on vital functions related to component acquisition and retrieval from the database, code generation and execution. The key point of the architecture is that such functions which require much more computational resources are centralized in the server side. For example, appropriate visual symbols are extracted efficiently based on server resources as well as on valuable statistics accumulated on the database, for browsing and editing processes. The server side mainly includes the following components.

**WEB Service APIs** are to bridge between any approved client applications and server resources. The component acquisition related to structures, schemes, scenes, variables, formulas and cyberFilms as a whole, are accomplished through the first eight services as shown in Figure 1. Each service receives a request with some parameters from clients, performs queries based on the corresponding SQL, and then returns result sets to the invokers. On the other hand, Code Generation Manager receives a core data of a cyberFilm and then generates the corresponding executable code with Code Generator. These services are provided by Tomcat where arrived requests are executed by multithreads. Note that, the each service receives http requests and returns the corresponding XML documents. This means that the server has been designed so that different applications can access to the server resources.

**CyberFilm Database** is to store and manage all resources of the programming environment. The resources include open sets of algorithmic symbols and components related to language elements as well as user information. The database is organized by MySQL. Each algorithmic character/component has a unique ID and is managed within relational tables. Attributes of each entity and their relations are used for searching as well as browsing and editing operations as discussed in the next section.

**Code Generator** consists of several components for a number of phases. Based on the received core data, Template Composer sorts out necessary template programs from the database and then composes them into a target machine code for execution. Function Composer inserts descriptions of necessary functions to the target code. Formula Parser replaces formal keywords in the target code by the corresponding C++ expressions. The syntax of formulas has been defined in BNF and the corresponding parser has been implemented in the Formula Parser. These composers and parsers also provide error messages occurred during their own generation processes.

### 4.3 Internal Organization

Figure 2 shows a high-level schema of internal organization of the environment. The internal representation of a cyberFilm is principally a set of frames classified into four different groups where they are mutually supportive and even are considered as special annotations of each other. In fact, fundamental elements of a cyberFilm are consolidated into the integrated view as the core data, and contents of other views can be downloaded on demand using IDs of target elements. Important thing is that, in the client environment, any...
modifications of the core data reflect changes in other views automatically, and vice versa.

5 System Interactions and Interface

In this section, a typical usage scenario of the environment and general functions of the browser and the editor are presented. Generally, integrated views (programs) are shown in the main browser with supplemental on-demand pop-up browsers and editors of the corresponding elements.

5.1 Usage Scenario

Here is a typical usage scenario of FIM Environment:

1. The user accesses Web page of FIM Environment using a suitable browser, e.g. Firefox, then the application is delivered to the user’s browser.
2. The user should log into the environment.
3. The user downloads a cyberFilm or creates an empty cyberFilm as a new component. The user can download both his/her own components and any other public components created by others.
4. The user browses the cyberFilm.
5. When necessary, the user can make modifications to the cyberFilm to make it an original component.
6. The user can click “Run” button to generate an executable code and run it.
7. When necessary, the user can check compilation errors occurred during the generation process and debug the program in phases 4. and 5.
8. The user can observe and analyze the generated codes and the computational result.
9. The user can upload the modified component to the server to share it with others.
10. The user can export created frames as animations or pictures and save them in a local file system.

5.2 Browser

Figure 3 shows a screen shot of the main user-interface of FIM Environment. It consists of a main panel to browse and edit integrated views, some menu panels, an outline panel (optional) and a console panel. **Browser** of the integrated view corresponds to the view which holds several integrated views within a tab navigator where the user can select a target program. This view also includes a control panel to adjust the size of the program and visibility of program elements as well as to edit, run, upload and export the program.

---

**Figure 3: The main panel of FIM Environment**

---
elements. The control panel also includes buttons to edit, run, export and save the target program.

**Console** is to observe messages from the server side after “Run” button is clicked. The user can see 1) generated target codes, 2) necessary header programs, 3) compilation error messages from Code Generator, 4) compilation errors from the G++ compiler and 5) the standard output of the target code. In addition, special visualizers show computational results if the program includes functions to perform such output operations.

The key point of the browser is “embedded clarity” in the program. In the programming environment, different levels of embedded clarity are applied [22]. For example, although the user can program within rather compact format through the integrated views, when the user does not know the meaning of an icon for a cyberScene (in principle, the user does not need to memorize), the environment shows the corresponding animation frames on a pop-up panel as shown in Figure 4. So, different algorithmic features can be explained in the corresponding views in appropriate languages based on user’s demands. Another example is the expansion of elements in the variable declarations by displaying full names, applicability tasks and their semantics.

5.3 Editor

In the editor, the user can directly manipulate symbols and constructs. In fact, the browser of the integrated view is also a part of the editor. Basically, elements of the program can be edited by appropriate pop-up panels opened on demand and drag-and-drop operations with special selectors.

After the user downloads (or creates a new component), process of programming in pictures under the current version of FIM environment consists of the following iterative stages:

1. Specifying structure parameters. The user should input the size (and the configuration) of the structures.
2. When necessary, adding and deleting structures. A unique label should be assigned to each structure, when the new structure is defined.
3. Modifying, adding, deleting variables assigned to the corresponding structures. For each variable, its shape, data type and name should be defined.
4. Adding new schemes. A scheme is a temporal construct such as for-loop and while-loop constructs which can include other sections (schemes or scenes).
Figure 5: Some on-demand panels to edit the program elements around the integrated view

<table>
<thead>
<tr>
<th>Source element</th>
<th>Target element</th>
<th>Action performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>An icon in structure list panel</td>
<td>The header in the integrated view</td>
<td>Add a new structure</td>
</tr>
<tr>
<td>An icon in scheme list panel</td>
<td>The body or a section in the integrated view</td>
<td>Add a new scheme</td>
</tr>
<tr>
<td>An icon in scene list panel</td>
<td>A section in the integrated view</td>
<td>Add a new scene</td>
</tr>
<tr>
<td>An icon of a new formula</td>
<td>A terminal section in the integrated view</td>
<td>Add a new formula</td>
</tr>
<tr>
<td>An icon of a new variable</td>
<td>A structure area in the header</td>
<td>Add a new variable</td>
</tr>
<tr>
<td>A section in the body</td>
<td>Another section in the body</td>
<td>Move a section (scheme/scene) to the specified position</td>
</tr>
<tr>
<td>A section in the body</td>
<td>The trash box</td>
<td>Delete the section</td>
</tr>
</tbody>
</table>

Table 1: A list of possible drag-and-drop operations to edit a program

<table>
<thead>
<tr>
<th>Source (Click)</th>
<th>Panel</th>
<th>Action performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter area in the header</td>
<td>Parameter Editor</td>
<td>Specify parameters for the corresponding structure</td>
</tr>
<tr>
<td>Variable area in the header</td>
<td>Variable Editor</td>
<td>Specify a shape, data type and name as well as semantics of corresponding variables</td>
</tr>
<tr>
<td>Formula area in a scene</td>
<td>Formula Editor</td>
<td>Specify formulas and functions for the activity involved nodes in the corresponding cyberScene</td>
</tr>
<tr>
<td>Edit button in the menu</td>
<td>Skeleton Editor</td>
<td>Insert structures, schemes and scenes</td>
</tr>
</tbody>
</table>

Table 2: A list of possible operations within the corresponding editor panel
5. Adding a new scene. A scene can be inserted into a specified position in a section. The label of the target structure in which the new scene invokes, should be specified.

6. Moving, deleting a section. A section can be moved to a specified position in another section.

7. Writing, adding, deleting formulas in corresponding scenes.

The above mentioned operations can be performed by drag-and-drop operations or typing within the corresponding special pop-up panels. Table 1 and Table 2 show details of these operations. Figure 5 shows some panels to edit the program. One of the panel supports inserting icons related to structures, schemes and scenes into a target section in the integrated view by drag-and-drop operations. Another panel of Formula editor allows the user to click (or touch) icons representing formula elements to insert them into the position the cursor indicates.

One of key points of the editors is related to their functions to keep compatibility of the program elements and to suggest targets elements to be selected. The users can select an item from only compliant candidates obtained from the server when they specify (select) the icons and the formulas. For example, the user never assigns an in-order traversal scheme to a 2d-grid structure, and never assigns a collective function to an individual operation. In addition, for selecting items (e.g. scene icons and function symbols) from a number of candidates, the editor lists up these items in the appropriate order based on frequency and statistics accumulated on the database.

6 Scalability of Library Components

The principle of the scalability states that the environment should ideally support a huge number of algorithms (components) and problem domain without limitation. In our programming environment, although visual constructs and their operations are rather simple, the number of supported icons and super-characters are open sets. New icons and super-characters are developed when the necessity arises to support the representations of new algorithms. Open sets of programs that the end-users created within the environment, will also be future assets. The huge number of icons and super-characters is not a major problem for searching and editing operations because of the intelligent assistances for symbol selecting and keeping compatibilities.

Many algorithms related to trees, graphs, computational geometry and other fields have already been implemented and stored in the database. However, the development process is continued. For example, the skeletons including computational schemes on moving particles are considered as a next step.

7 Conclusion

A tool that enables programming in pictures based on an approach of Filmification of Methods (FIM) has been presented. FIM employs pictures and moving pictures to specify and program computational algorithms. Several interfaces including browsers, editors as well as sub-systems deployed on the server for organizing a visual programming environment have been implemented within the tool. The implementation allows end-users to access information resources through web browsers and create their own software components. This enables the users to retrieve library components from database, browse and edit them with intelligent supports, generate executable codes in C++, execute them to observe computational results, and upload and share the components. The key point of the environment supported is that it employs appropriate visual languages to browse and edit corresponding algorithmic features of the components as well as embedded clarity annotations to facilitate the processes of understanding and programming. Furthermore, the tool allows to export the visual programs and computational results in several formats, including Adobe Flash animations, PNG or JPEG images. Some aspects of the scalability of the environment and library contents have also been shown.

References:


