Service Oriented Movie-based Programming Environment

Ruth Cortez, Hayato Tan and Alexander Vazhenin
University of Aizu, Department of Computer Science and Engineering. Fukushima, Japan
Email: (d8132103, m5171203, vazhenin)@u-aizu.ac.jp

Dmitry Vazhenin
Simulatio Corp
Yokohama, Japan
Email: dmitry@simulatio.jp

Abstract—Movie-based (MB) programming environment is a visual programming platform for creating algorithms and programs applying animation frames with computational and sensible operations. It facilitates the user’s understanding and debugging of a program by coloring correlated areas of structures, introducing appropriate computational formulas as well as specifying algorithmic and logical behavior of those areas. The goal of the presented work is adapting the current system to Service-oriented Architecture in order to organize a sharable storage and server-based execution of MB-programs and algorithms. Therefore, the MB-programs/algorithms could be shared and reused by different developers. In this paper, features of MB-programming paradigm and the corresponding software are presented, which are used to enhance capabilities for data/knowledge and execution management. In the framework of Virtual-MVC design pattern, the MB-management architecture, component’s design steps and a prototype of service-oriented MB-programming environment are discussed, including concepts and examples of usage.

I. INTRODUCTION

Trends in Software development shows that it is acquiring importance to provide programming languages for non specialist people, even at early ages such is the case of the Scratch programming language from MIT [1]. The users can concentrate on creating interactive programs by joining together a set of blocks. The programs can be shared among a community of users, enhancing the programming learning process. Under traditional textual programming, understanding program syntax and semantics, as well as debugging code is a complicated task. The difficulties that come along are due to 1) the one-dimensional approach of the traditional textual programming languages, where multi-dimensional structures are encoded into one-dimensional strings according to some intricate syntax, as well as 2) the complexity of combining mathematical formulations with human language. It is necessary to express the program as text-based source code, and use a compiler or interpreter to transform this text into a semantic readable representation [2], [3], [4]. In contrast to traditional textual programming languages, Visual Programming Languages (VPL) essentially remove this layer of abstraction and allow the programmers to directly observe and manipulate the complex software structures. Two-dimensional displays for programs, such as icons, forms, and diagrammatic-based programming language supported by graphical user interfaces, have positively advanced program understanding [4], [5] and collaborative work under distributed environments.

GASPARD (Graphical Array Specification for Parallel and Distributed computing) is an example of visual programming environment devoted to the development of parallel applications based on task and data parallelism [6]. The system has a simple programming interface for task specification, which are instantiated by plugging them into a slot. The user can design a program by using components from an Elementary Transformation Library, including familiar signal processing operations like fast fourier transformation, integration, sum, dot products, etc. Each of these operations takes arrays as input, and return output arrays. Those operations may also be parameterized, for example, by the size of the arrays. Alligator is a Web-based Distributed Visual Programming Environment supporting the data-flow programming language [7]. The programming process involves constructing the computational scheme from available blocks by choosing any block in a library. Accordingly, the visual program is represented as a special data-flow graph without explicit links. Graphically, blocks appear to be fitted one to another similar to LEGO® building blocks. Actually, this approach is very close to the idea proposed by Tanimoto named Data Factory [8], which is also an iconic-based data-flow visual language. The Alligator system is a client-server application composed of three kinds of machines: the main server, clients, and execution servers where the actual execution of some blocks occurs.

A programming platform called Lawnmower is another example of a web-based learning environment for architecture students [9]. It is oriented to learn programming, especially when the final code is required to be entered in textual form. It provides automatic translation of visual programs into textual code. Students can visually observe the similarities and differences between flow-based and block-based programming styles, as well as understand transitioning to text-based programming. Clients are interacting with the system via their browsers on a webpage, which utilizes AJAX to refresh itself, based on user inputs and server actions. The rendering of the VPL is controlled by JavaScript in the browser. Storing and sharing of code components, as well as collaboration among the users is performed using a message-oriented database. A VPL named Blockly Editor was released by Google [10], [11], with the goal of supporting developers to build software without typing a character by piecing together small graphical objects in much the same way you’d piece together Legos, the platform is similar to Scratch [1], but aims to facilitate support novice programmers to develop workable scripts for web-based applications. In Blockly, each visual object includes also a code object that can be a variable, a counter, or a
conditional resources. The user can also compose his/her own functions or final application by joining other objects together. From Google’s site, the user can translate Blockly applications into existing languages, including Javascript, Dart, and Python. The discussed systems are examples of flow-based and block-based Visual Programming styles based mostly on client-server distributed model.

In this work we propose the extension of a VPL to share resources under a distributed environment applying a Service-Oriented Architecture (SOA) design pattern [12], [13], to support distributed storage, sharing and reusing of algorithms knowledge representation and software components. The processing logic of the shared resources can be utilized to construct new applications based on services reusability. The application we focus on is called Movie-based Multimedia Environment for Programming and Algorithms Design (MMEPAD). It aims to provide visual representations of mathematical notations such as matrices, which require a high-level language constructs, otherwise complex in text-based programming environments. It simplifies the coding task for users with basic programming skills, but familiar with mathematical notations. The language constructs contains special symbols and images with embedded semantic descriptions. The innovative approach is on introducing dynamical features to the visual algorithmic constructs in a movie-based film representation. In MMEPAD, each step of an algorithm corresponds to a problem solution step. The user can create/debug the code using algorithmic/movie objects called metaframes, which contains information about the visual structures and their parameters such as variable/structure names, activities on areas, as well as a control-flow, and computational formulas that should be implemented during each algorithmic step on node elements within these areas. The nature of the metaframe information is similar to Metadata in Meta-programming, allowing to automatically generate executable code for a target computational platform, as well as to produce a corresponding series of animation frames. The generated code and frames are adequate to each other. The Movie-Program film (MP-film) is a collection of metaframes specifying a whole algorithm, both movie animation and executable program can synchronously be generated, executed and debugged. This technology supports parallel code with relatively good quality, that could be later used for the OpenMP programming platform [14].

The contribution of this paper is on the extension of a MMEPAD stand-alone application and prototyping of Business to Business connection of the original Virtual-Model-View-Controller (Virtual-MVC) design pattern [15]. The aim of Virtual-MVC is demarcate the presentation (View) and the implementation (Model) logic by inducing an Integration layer (Controller), to encapsulate Non-Functional logic such as security, reliability, scalability, and message routing; enabling the separation of Integration Logic from the Functional Logic (Client Application) and Implementation Logic (Service). With this approach, the View and Model become loosely coupled. The controller enables the View to produce high-level queries that are parsed and analyzed by the controller in order to compose the required services. The Model is represented as a service inventory organized as a set of domains that we call Engines. An Engine contains endpoint services comprising processing logic designed as loosely coupled functionality. The set of MB-computational services use the available computational resources to support the execution of MB-programs on several parallel platforms. The rest of this paper is organized as follows. In Section 2, the principles of MB-programming are explained including the process to create a program/movie. Section 3, describes approaches to adapt MB-programming to a distributed environment, by applying the Virtual-MVC design pattern. Component’s design steps are described in section 4. Finally the conclusions and future work are presented in section 5.

II. MOVIE-BASED PROGRAMMING ENVIRONMENT

The idea behind Movie-based programming is to create and support high-level language constructs using special symbols and images with semantic support to model dynamical features of the algorithms. This visual representation of algorithms is in demonstrating their features in the form of a movie, in which is possible to specify a behavior in time to dynamic elements such as film, frames, control lines, structures and structure elements as well as computational formulas. It is also in facilitating not only the programming task but also the debugging process. In this section, we summarize briefly the main principles and functions of this VPL [6], [9].

A. Basic Concepts

Movie-based programming paradigm follows the analogy of a movie film. Typically, one frame corresponds to a logically completed part of an algorithm, for example, one step of an iterative method. According to its nature, we distinguish two types of frames. The animation frame is an image that shows dynamically a step in the algorithm to improve the user’s perception. According to matrix structures, each frame highlights and flashes elements of parameterized matrix structures, defining operations or formulas. Different operations can be coded by different colors. Special Control Lines (I1, J1 in Fig. 1) are used to reference these areas of activities. Control lines can change their placement inside the matrix during frame transitions. The second type is the execution frame, that represents a source code snippet. Both types of frames are produced by a basic component called Metaframe because of the presence of these dualistic features. A Film is a collection of metaframes. It is possible to have a film collection in which each film is independent from other films, but films can also be nested. Any film is able to produce animation and execution frames, and contains a collection of spatial structures, a set of variables declaration, and metaframes. The metaframes are composed by a set of rules and parameters to produce the executable code from the algorithmic film. The metaframe’s rules and parameters are defined with a set of traversal schemes for the activation of structure nodes by attaching operations (Computational formulas) on them. A set of Control-Flow formulas (CF-formulas) are applied in the frame generation process to specify a transitional assignment of film parameters from one frame to other [14]. There are two main types of metaframes: single and episode. Single frame is to specify a single step of an algorithm, and episode is to define multiple algorithmic steps, where each of them has the same computational formulas, but the original configuration of node activities.
B. Movie-based program design stages

The general scheme of the MB-program design includes the following four stages: Create a MP-film prototype. It defines only a distribution of computations over metaframes, by assigning activities on structure elements over a time interval. The user can design his/her MP-film by creating/editing his/her own metaframes, or by importing them from external films. CF-formulas need to be introduced as frame-to-frame transitional assignment. Examples of CF-formulas are shown in Fig.1.

Create the complete MP-film according to the concrete application requirements. For each metaframe, it is necessary to specify the computational formulas on active structure nodes in a prototype film. For debugging procedures, these formulas are extended by special breakpoint operators and the corresponding references for monitoring structure’s data, as well as checking the correctness of the index expressions that define structure elements.

Generate, Execute and Run-time Debugging. The internal representation of a metaframe allows generating a movie animation and the corresponding executable code, based on a set of source code templates. The MB-program is implemented using a special monitor tool that controls the program execution and visualization of the matrix data. A Run-time debugging monitor uses a special interface panel to display the status of a program during its execution, as well as the value of the data in a particular step. The panel has a control line and a matrix structure visualization layers. This panel contains the information about a metaframe number and frame position inside of metaframe, to point the user to a place of a MB-program where the stop is occurred. When a breakpoint has been reached, the program is paused and the data visualization panel is invoked reflecting the most recent state of the data structure specified by the Visualization Invoker operator, which should be specified before any breakpoint in a formula.

Export the algorithmic movie and executable program to the target machine. The code generator produces the final executable program based on the MB-constructions, according to the target machine requirements.

C. Movie-Based Programming Environment (MMEPAD) architecture and reusability features

The main components of the stand-alone architecture of MMEPAD are shown in Fig 2. An MP-film is associated with an internal semantic representation or metadata, this metadata is stored in XML files called MP-templates. A code generator component processes the MP-templates generating the executable code and a movie animation based on a set of source code templates. The templates store snippets of code in different programming languages semantically associated with the visual objects [14]. With the current features, MMEPAD facilitates the development of complex algorithms and parallelization of tasks. The executable code produced in C language can be incorporated into external applications such as OpenMP platform, therefore reducing the development effort in such complex algorithms. However, MMEPAD is a standalone application and does not support collaboration and sharing of resources on a centralized repository, which could greatly enhance the reusability of MB-programs for collaborative work in such complex tasks. Therefore, we propose to create a repository organizing MP-films in the form of a library (knowledge base), where the developers can share MP-templates and have access to a common library and reuse the algorithms by importing them into their own MP-films, and in future expose them as a service which can be integrated on external applications.
III. SERVICE-ORIENTED MOVIE-BASED PROGRAMMING ENVIRONMENT

A. Virtual-MVC Design Pattern

The MVC design pattern was initially proposed to increase the reusability of the code under the Object Oriented paradigm [16]. Virtual-MVC is a compound pattern which extends the traditional Model-View-Controller (MVC) for Service Oriented Architecture, in which the processing logic of the components is designed as a service. The view layer encapsulates functional requirements of an application while the model holds the business logic. The adaptation from the MVC is on removing the direct link between the view and the model, by reorganizing it within the controller introducing the Enterprise Service Bus (ESB), and Dependency Injection patterns [15]. The ESB is a flexible connectivity infrastructure for integration based on Web services, and which main functions are messaging, intelligent routing, transformation and orchestration [17]. Therefore, the controller is designed to handle the complexity of heterogeneous applications, and seamless integration among disparate services.

In the Virtual-MVC pattern, the components can be distributed on independent servers. The view is hosted as a Web Server containing Web based interfaces and graphic components such as CSS, JavaScript, multimedia objects etc. to construct the User interface, and manage the view logic. The view layer of Virtual-MVC has no knowledge of the underlying logic in the model, issuing a high level request by filling up an itinerary request form defined and managed by the controller. This form contains all the information of the required functionalities that the view layer will provide as input data payload. The decoupling from the model allows the view to be developed autonomously, facilitating the GUI design and development task [15]. At the view layer, external applications can be connected to the controller via Internet protocols such as SOAP and HTTP.

The controller performs non-functional and integration logic tasks, supporting QoS properties related to the integration such as security, performance, and availability, works by negotiating the Service Level Agreement (SLA) among the service consumer and provider. The integration task involves mainly Data transformation, Protocol transformation, Mediation (Message Routing), and Service Composition, for which the architecture of the controller is designed. The details of the architecture is discussed in [15]. It acts as a centralized container that mediates a request received from the view to find and invoke the appropriate service in the model avoiding direct reference from the view, which means enabling higher privacy of the business logic. The service mapping information is registered in the ESB and placed in the Service Registry and Repository. By using this service mapping, the ESB composes endpoint services that are necessary to execute the requests. The services are deployed and maintained in the model layer. A Service Virtualization component mediates the corresponding services by using a Service Mediation Pattern. The advantage of our approach is that it does not require the Universal Description Discovery and Integration (UDDI) registry since the view layer has no knowledge of the endpoint services, increasing interoperability and reusability of the business logic. The model is normalized into a database layer and a business logic layer. The Data Access Object (DAO) pattern is used to transfer the state of the data from the database layer to the business access layer. The business logic is organized following the Service Inventory pattern to standardize the blueprint of the endpoint services in the model [12]. In the service inventory, we introduce an Engine sub-structure, to organize logically the services according to specific domains of application. Fig. 3 presents the new MMEPAD architecture based on Virtual-
MVC. In the model, two engines that will provide the business logic. The core engine is called Movie-based Engine (MB-Engine), that contains the services corresponding to the Movie-based environment. The second engine to be used is called Wiki-Miner Engine, containing services to get contextual help from Wikipedia content enriching algorithms descriptions.

B. Designing Movie-based Components under Virtual-MVC

We are extending MMEPAD functionalities under the Virtual-MVC pattern to support reuse of MP-templates containing the MB-programs and the executable code under a distributed environment, as explained in Fig 3. The stand-alone application is extended with the functions to import (download) and export (upload) the executable code and MP-templates to the library in the model. The users can also browse the library and add new categories. Once the MP-templates and executable code are stored in the model, they can be viewed independently of MMEPAD environment under a Web-based application. The independent version, supports library browsing functionalities (See Fig. 4). When the user selects an algorithm from the library, the application shows the description, enhanced with automatic glossary, the display of the corresponding algorithm movie, and an execution monitor screen to show data values color coded. The user has the option to download the algorithm or create a new entry in the library. At the model the MP-films are organized within a library structure in the database, as well as the corresponding executable code. The business logic to process the MP-film library, and the execution functions are organized under the MB-Engine as endpoint services, which are registered in a service metadata table (See Fig. 5). In previous research work we developed an engine to extract and process Wikipedia content named Wiki-Miner Engine [18]. We are reusing these services to provide contextual help for algorithm descriptions, displaying a dynamic glossary from concepts in Wikipedia. This is helpful to describe mathematical terminology, formulas, etc. The development effort to reuse the Wiki-Miner services is at the view layer by selecting the necessary services, and at the controller layer we create the request and response messages in XSLT for message exchange. At the view layer, we develop a component embedded in MMEPAD to connect with the Controller via HTTP.
IV. COMPONENTS DEVELOPMENT AND DESIGN ISSUES

A. Service Definitions

The service endpoints are deployed in the model as a java class and the service definition is registered in the Metadata schema Fig. 5. Initially, we are defining six services for the extended functionality, and they are described as follows: loadLibrary retrieves the Library structure that is shown to the user for the algorithm selection. loadMpFilm retrieves the selected algorithm MP-movie animation and executable code. saveMpFilm stores in the database the MP-film templates and the executable code. uploadAlgorithmInput updates a data file containing the input for the algorithm execution. executeAlgorithm will process the input registered in the data file and execute the algorithm. Finally, updateLibrary is used to handle modifications to the Library structure. Services in the model follows a template, to help the developer to concentrate on designing the business logic process the request and generate the response parameters associated to that service. Fig. 6 shows a code snippet of a service endpoint template. In this example we use the service loadLibrary. The service endpoint is a class that implements the interface registered in the Service Definition Metadata Schema. The input parameters are provided by the corresponding RequestValueObject, which has been deployed in the model. The developer can customize the code corresponding with the processing logic, and finally add the result parameters to the object declared as a ResponseValueObject.

B. Components Workflow scenario

The Virtual-MVC pattern supports application’s development based on services composition, handled through itinerary based messaging at the controller layer. The components workflow sequence is described in Fig. 7, showing a simplified example for the MP-Library browser request to download a MP-film. Initially, the view issues a request with an input payload that includes the service ID and a set of input parameters. The service ID corresponds to the appropriate service registered in the Metadata definition table. The controller identifies the service contract and process the XSLT message, routing the requests to the appropriate value object, service endpoint, interface, endpoint data, and the corresponding function. The mapped data is in string format and it is inflated into the Data Transfer Object of the service model, which is deserialized by the controller into the model data format. After the completion of the service request in the model, the controller deserializes the Response Value Objects corresponding to that service endpoint, rendering a response message in XML which is delivered to the view producing the HTML presentation, as well as the necessary css, javascript files, and images.

V. CONCLUSION

In this research we have discussed the extension of the MMEPAD stand-alone application under Virtual-MVC Design Pattern, to share MP-films algorithms under a distributed environment. The advantage of designing processing logic in the Movie-Based Engine, is that the services can be composed and reused by different applications and combined with services in other engines, as in the case presented for Wiki-Miner. The executable code of the MP-film uploaded in the model can be reused for processing the corresponding calculations and exposed as individual applications that can be accessed outside of MMEPAD as well. The variety of local and remote computational resources can be also implemented as a set.
of Engines. From the user’s point of view, it is a subset of endpoint services that encompass functionalities of an API and realize processing that is specific to an application. This makes possible to provide the flexibility of available computational resources, supporting the execution of MP-programs on various programming platforms. Additionally, the integration with Wiki-Miner Engine helps the users to perform a semi-automatic generation of glossaries for algorithm descriptions.

REFERENCES


