IMPLEMENTATION OF EXTENSIBLE FLOWCHARTING SOFTWARE USING MICROSOFT DSL TOOLS

Mikas Binkis, Tomas Blazauskas

Kaunas University of Technology, Department of Software Engineering, Studentu str. 50-101A,
Kaunas, Lithuania, binkis.mikas@gmail.com, blaztoma@gmail.com

Abstract. Currently there are commercial and freeware tools that allow users to specify algorithms using visual flowcharting software. These tools are widely used for many purposes – from visual programming to specification of common actions, yet they usually can't be extended with additional elements and have limited execution capabilities. Our goal is to choose a flexible implementation platform and create extensible executable flowchart system that can be used to teach students programming. In this article we present a flowchart metamodel created with Microsoft DSL Tools – a graphical designer for domain models with a set of code generators.

Keywords: flowcharting, domain specific languages, Microsoft DSL Tools, visual programming, MDA, extensible flowcharts.

1 Introduction

Flowchart is graphical representation of a process or the step-by-step solution of a problem, using suitably annotated geometric figures connected by flowlines for the purpose of designing or documenting a process or program [1]. Because of the rather simple and comprehensible notation, flowcharts are widely adopted as one of the means of teaching algorithms and programming. Although computer science uses a variety of flowcharts, new software, dedicated to flowcharting, have offered extensibility. Some programs have combined flowcharting capabilities with programming and allow users to transform their diagrams into specific programming code.

Microsoft Visual Studio add-on Microsoft DSL Tools is an interesting case, since it allows to graphically specify the metamodel of a diagram (or in this case – flowchart) and create a model editor. This means that it is possible to modify flowchart models by adding custom elements, which extends the usability of flowcharts in almost every imaginable field. Another advantage of the DSL approach is that the modelling environment can constrain and validate a created model for the domain's semantics, something that is not possible with UML profiles [2].

The model editor, created with DSL, can be used to transform graphical notation into any type of programming code from simple XML notation to a working program, thus creating an executable flowchart solution. This gives an advantage over commonly used flowcharting software, that usually offers only limited code transformation options and does not provide other than default executable environment.

In this article we will briefly review some of the existing flowcharting software, analyse main properties of domain specific languages and propose our prototype flowchart engine, based on Microsoft DSL Tools for Visual Studio 2008.

2 Background

2.1 Flowcharts in learning process

The human ability to realize graphic representations faster than textual representations led to the idea of using graphical artifacts to describe the behaviour of algorithms to learners that have been identified as algorithm visualization [3]. Visual programs based on flowcharts allow students to visualize how programs work and develop algorithms in a more intuitive fashion. The flow model greatly reduces syntactic complexity, allowing students to focus on solving the problem instead of finding missing semicolons [4].

Flowcharts complement other learning methodologies. One of the examples is Algorithm Visualization using Serious Games (AVuSG) – an algorithm learning and visualization approach that uses serious computer games to teach algorithms [5]. Open source, simple structured flowcharts, presented in widely acceptable standard formats may also contribute to the expansion of collective creativity. It is an approach of creative activity that emerges from the collaboration and contribution of many individuals so that new forms of innovative and expressive art forms are produced collectively by individuals connected by the network [6]. This way flowcharts may be used as a learning medium to transfer and exchange both simple and complex algorithms or scenarios for various frameworks, that utilize flowcharts.
2.2 Existing flowcharting solutions

RAPTOR is an open source iconic programming environment, designed specifically to help students visualize classes and methods and limit syntactic complexity. RAPTOR programs are created visually using a combination of UML and flowcharts. The resulting programs can be executed visually within the environment and converted to Java [7].

The “Iconic Programmer” is an interactive tool that allows programs to be developed in the form of flowcharts through a graphical and menu-based interface. When complete (or at any point during development), the flowchart programs can be executed by stepping through the flowchart components one at a time. Each of these components represents a sequence, a branch, or a loop, so their execution is a completely accurate depiction of how a structured program operates. To solidify the concept that flowcharts are real programs, the developed flowcharts can also be converted into Java or Turing (present capability), or and other high-level language (easily extendable) [8]. Ionic Programmer supports input / output, selection, looping, and code generation, but does not support subprograms.

The SFC (Structured Flow Chart) Editor is a graphical algorithm development tool for both beginning and advanced programmers. The SFC Editor differs from other flowchart creation software because its focus is on the design of flowcharts for structured programs; using a building block approach, the graphical components of a flowchart are automatically connected and structured pseudo-code is simultaneously generated for each flowchart. While SFC was originally designed as a tool for beginning to intermediate programmers, it has been used by students in upper level classes and by professional system designers [9].

Visual Logic [10] provides a minimal-syntax introduction to essential programming concepts including variables, input, assignment, output, conditions, loops, procedures, arrays and files. Like other flowchart editors, Visual Logic supports visual execution and stepping through the elements of a diagram. The language contains some built-in functions from Visual Basic, yet it does not support the creation of classes.

2.3 Domain specific languages

Domain specific language (DSL) is a language designed to be useful for a delimited set of tasks, in contrast to general-purpose languages that are supposed to be useful for much more generic tasks, crossing multiple application domains. [11]. A key benefit of using a DSL is the isolation of accidental complexities typically required in the implementation phase (i.e., the solution space) such that a programmer can focus on the key abstractions of the problem space [12]. Other benefits of DSLs include [13]:

- DSLs allow solutions to be expressed in the idiom and at the level of abstraction of the problem domain. Consequently, domain experts themselves can understand, validate, modify, and often even develop DSL programs.
- DSL programs are concise, self-documenting to a large extent, and can be reused for different purposes [14].
- DSLs enhance productivity, reliability, maintainability [15, 16], and portability [17].
- DSLs embody domain knowledge, and thus enable the conservation and reuse of this knowledge.
- DSLs allow validation and optimization at the domain level [18, 19, 20].
- DSLs improve testability following approaches such as [21].

A graphical domain-specific language must include the following features [22]:

- Notation – a domain-specific language must have a reasonably small set of elements that can be easily defined and extended to represent domain-specific constructs.
- Domain Model – a domain-specific language must combine the set of elements and the relationships between them into a coherent grammar. It must also define whether combinations of elements and relationships are valid.
- Artifact Generation – one of the main purposes of a domain-specific language is to generate an artifact, for example, source code, an XML file, or some other usable data.
- Serialization – a domain-specific language must be persisted in some form that can be edited, saved, closed, and reloaded.

A domain-specific language is defined by its domain model. The domain model includes the domain classes and domain relationships that form the basis of the domain-specific language. The domain model is not the same as a model. The domain model is the design-time representation of the domain-specific language, while the model is the run-time instantiation of the domain-specific language. Domain classes are used to create the various elements in the domain, and domain relationships are the links between the elements. They are the design-time representation of the elements and links that will be instantiated by the users of the design-specific language when they create their models [23].
Despite the benefits offered by DSLs, there are several limitations that hamper widespread adoption. Many DSLs are missing even basic tools such as debuggers, testing engines, and profilers. The lack of tool support can lead to leaky abstractions and frustration on the part of the DSL user [24].

We have chosen Domain Specific Language Tools for Microsoft Visual Studio 2008 for our research, since it provides robust development environment, adequate debugging mechanisms, good extensibility options (it is possible to add custom elements to the flowchart model), .NET framework support and sufficient documentation.

There is also an approach of domain specific language creation by using UML profiles [25]. Instead of heavyweight metamodeling, the developer can create a full-featured DSL, based on a UML profile and its customization. One of the tools, implementing this approach, is MagicDraw. Although it’s possible to validate models and generate code from them, surveys show [25] that the tool lacks customization flexibility (e.g. creation of a new symbol, which has nothing in common with existing UML symbols, inability of changing default UML metamodel values etc.).

3 Prototype flowcharting tool

3.1 Proposed solution

The main goal of our work was to create a customizable visual flowcharting tool with capabilities of generating flowchart definition code that could be utilized by other software (Figure 1). We have chosen a simple flowchart version with most common flow diagram symbols and created a metamodel (detailed in section 3.2) using Microsoft DSL Tools. The metamodel was compiled into an IDE that can be used to draw flowcharts and convert them into customizable XML code. The XML code is used by Adobe Flex application, which visually represents the flow algorithm.

Figure 1. From metamodel to code

It is important to note that Microsoft DSL automatically creates XML code from both metamodels and models (via implemented serialization functions), yet the generation lacks flexibility – usually even little changes in XML structure requires considerable modifications of the generator. That’s why we chose a much faster and easier solution and created our own XML generator (detailed in section 3.3).

3.2 The metamodel of flowchart

Microsoft DSL Tools for Visual Studio IDE can be used to create virtually any type of diagram metamodel by specifying it’s object hierarchy, relationships between classes and representation of model objects. The main elements of the metamodel are domain classes (with optional domain properties), that can be connected among themselves with three types of relationships [26]:

- Inheritance – a relationship between a base class and a derived class. Displayed as a line that has a hollow arrow that points from the derived class to the base class.
- Embedding – a containment relationship. Displayed as a solid line in the diagram.
- Reference – a relationship between two domain classes that do not have an embedding relationship. Displayed as a dashed line on the diagram.

Every domain relationship has roles (source / target) and multiplicity (specifies how many elements can have the same role in a domain relationship).

As it is shown in Figure 2, our flowchart model consists of the most common flowchart symbols – “Start”, “End”, “Decision”, “Action”, “Input”, “Output” and “Subdiagram”. Every symbol has identification number (provided by IDE) and a name. Symbols, that require user input in a model, have property “Value”, which, if necessary, can be validated by custom criteria. Connection between diagram elements is called “FlowConnection” and has a property “Condition”, used in conjugation with “Decision” symbol.
3.3 Code generator

The generator of code is based on text template mechanism, provided by IDE, and therefore enables to transform model into any type of programming code. We have chosen XML, as it is most versatile and is best suited to achieve our goals. It is also important to note, that code generator is not limited by any output language, since by using Microsoft Visual Studio text templates it is possible to transform the model into any type of programming code or formal notation like GraphML. While generated IDE with the mentioned code translator might seem like a all-round platform independent solution, the IDE itself is mainly restricted to operation in Windows family operating systems. Some sources indicate, that this may be only a temporary inconvenience, as Microsoft has recently acquired Teamprise, a division of SourceGear that built tools to give developers access to Visual Studio 2008 Team Foundation Server from systems running Linux, Mac OS X and Unix [27].

To illustrate our solution, we present a simple example of a program (Figure 3), that asks for a numerical input, checks if the given number is less than 3, increases the number if it’s not (simple illustration of a cycle) and outputs the number. Because of the text amount limitations in the article, we present only the generated XML code. Every element is specified by name (caption) and identification code. Additional attributes, such as Type and Color have been added to adapt the code that is used by an animated model representation application (detailed in section 3.4). We are planning to dedicate our upcoming articles to more detailed examples that will include more elaborate cases.
Figure 3. Example of a simple program, in our flowcharting environment

<Flowdiagram Caption="Diagram1" Id="4f5af4e7-4fe3-5a7i-018ca-48b3n7c01a32">
  <Start Caption="Start1" Id="a0014f1d-36a4-4dd6-97bf-f35c010a81ca" Type="Circle" Color="Black">
    <Target Caption="Input1" Id="6a63a117-e160-432b-99d2-18a35ae9c01b" />
  </Start>
  <Input Caption="Input1" Id="6a63a117-e160-432b-99d2-18a35ae9c01b" Type="ReverseParallelogram" Color="Green">
    <InputValue> a </InputValue>
    <Source Caption="Start1" Id="a0014f1d-36a4-4dd6-97bf-f35c010a81ca" />
    <Target Caption="Decision1" Id="9893ffde-e0a6-48d5-8871-7c54ff62738d" />
  </Input>
  <Action Caption="Action1" Id="8f4ff65-12cb-4909-8906-599e1f60cc48" Type="Rectangle" Color="Green">
    <ActionValue> a = a+1 </ActionValue>
    <Source Caption="Decision1" Id="9893ffde-e0a6-48d5-8871-7c54ff62738d" />
    <Target Caption="Decision1" Id="9893ffde-e0a6-48d5-8871-7c54ff62738d" />
  </Action>
  <Decision Caption="Decision1" Id="9893ffde-e0a6-48d5-8871-7c54ff62738d" Type="Diamond" Color="Yellow">
    <DecisionValue> a&lt;3 </DecisionValue>
    <Source Caption="Input1" Id="6a63a117-e160-432b-99d2-18a35ae9c01b" />
    <Source Caption="Action1" Id="8f4ff65-12cb-4909-8906-599e1f60cc48" Case="False" />
    <Target Caption="Action1" Id="8f4ff65-12cb-4909-8906-599e1f60cc48" Case="True" />
  </Decision>
  <Output Caption="Output1" Id="4ff36dba-aa1e-4b39-b656-a0586b47fe38" Type="Parallelogram" Color="Blue">
    <OutputValue> a </OutputValue>
    <Source Caption="Decision1" Id="9893ffde-e0a6-48d5-8871-7c54ff62738d" />
    <Target Caption="End1" Id="a3e22c3f-c8fc-47ea-b8ab-705c4623484b" />
  </Output>
  <End Caption="End1" Id="a3e22c3f-c8fc-47ea-b8ab-705c4623484b" Type="Circle" Color="White">
    <Source Caption="Output1" Id="4ff36dba-aa1e-4b39-b656-a0586b47fe38" />
  </End>
</Flowdiagram>
3.4 Executable model representation application

Our flowchart visualization software is implemented in Adobe Flex. This development framework was chosen because of its focus on GUI creation as well as its multiplatform support. It is accessible from any web browser with Adobe Flash Player plug-in installed. We also use Degrafa framework which allows to easily create scalable vector graphics elements.

This software is dedicated to provide visualization and debugging features. This allow students to watch algorithm flow by executing each diagram block element. The software also allows to change the flow of the algorithm as well as block element contents. So, the students can investigate quickly various "what-if" scenarios without leaving visualization software environment.

All flowcharts which are uploaded by users reside in a database located on server. Upon request server-side PHP script serves XML code which is converted into ActionScript object and presented in a browser window. We use separate website CMS (Content Management System) component to manage submitted flowcharts.

4 Conclusions and future works

Visual programs based on flowcharts are an affective aid in teaching of programming, because students focus on solving the problem, not concentrating on syntax errors. Current flowcharting software does provide considerable functionality for algorithm visualization and visual execution, but offers almost no extendibility of the flowchart model and too little model transformation and run-time execution capabilities. By using Microsoft DSL Tools, we have created an extendable flowchart model and a prototype IDE, which generates customizable XML code. We also applied the generated code for our executable model representation application, implemented in Adobe Flex.

In the nearest future we are planning to extend our flowchart model, so that it would be possible to convey more complex algorithms, and write additional model transformation plug-ins to study adaptability of the generated code for various frameworks. We are also planning to conduct a study by using the improved and completed flowcharting environment to teach students the principles of programming and assess the results in the upcoming article.

References


