Warsaw University of Technology
Faculty of Mathematics and Computer Science

ΦNITE

APPLICATION FOR BUILDING FINITE-STATE MACHINE THAT IS EQUIVALENT TO A GIVEN REGULAR EXPRESSION AND FOR SIMULATING MACHINE'S EVALUATION OF A GIVEN WORD

Author:
Mateusz Bysiek

Supervisor:
dr Lucjan Stapp

Warsaw, 21 Mar 2013

Document metric

Project	ΦNITE	**Company**	WUT
Document name	technical analysis		
Document topics	technical analysis of the problem, details of used algorihms, interface specification		
Author	Mateusz Bysiek		
File	bysiekm-business-analysis.pdf		
Version no.	0.15 **Status** pre-alpha	**Opening date**	16 Mar 2013
Summary	Author of the document analyses the problem of designing an application for solving a well defined language-theory related problem, doing it from the developer perspective, i.e. providing definitions of expected application environment, developer environment, user interface requirements, and all used algorithms.		
Authorized by	dr Lucjan Stapp	**Last modification date**	21 Mar 2013

History of changes

Version	Date	Author	Description
0.10	16 Mar 2013	Mateusz Bysiek	creation of template for the document
0.11	16 Mar 2013	Mateusz Bysiek	added GUI mockup
0.15	20 Mar 2013	Mateusz Bysiek	added abstract

Contents

1 The runtime environment **4**
1.1 Hardware requirements . . . 4
1.2 Operating system . . . 4
1.3 Software present in the system . . . 4

2 Development process constraints **4**
2.1 Developing the application . . . 5
2.2 Source code documentation . . . 5

3 Used technologies **5**
3.1 The runtime . . . 5
3.2 The development . . . 5

4 GUI mockup **6**
4.1 Main menu . . . 6
4.2 Main area . . . 6
4.3 Status bar . . . 6

5 Classes **7**
5.1 RegularExpression . . . 7
5.2 PartialExpression . . . 7
5.3 FiniteStateMachine . . . 7

6 Algorithms **7**
6.1 Conversion of plain text into a RegularExpression object . . . 7
6.2 Conversion of RegularExpression object into FiniteStateMachine object . . . 7

Abstract

Write here!

1 The runtime environment

This section provides a set of requiremets with regard to the environment, in which the developed program will be run.

Definition

- *the application* is a synonym for Φnite, the term includes the runtime of Φnitewith the documentation, but not the source code.

1.1 Hardware requirements

The computer on which the application is run must conform with the minimum requirements that are defined for operating system and the components listed in further sections. The application itself must require at most 1GB of hard drive space and 1GB of RAM to work. The actual implementation does not have to use all the available space, but it must work within the boundaries.

1.2 Operating system

Application is expected to work on the following set of operating systems:

- Microsoft Windows 7 Professional x86
- Microsoft Windows 7 Professional x64
- Microsoft Windows 7 Ultimate x86
- Microsoft Windows 7 Ultimate x64

1.3 Software present in the system

Other than the operating system, application cannot be expected to work corretly unless all of the following components are present in the operating system:

- .NET Framework 4.0 Full Profile
- PDF (Portable Document Format) file viewer
- PDF-LaTeX creation toolkit that is able to create PDF files from LaTeX source code using command-line.

The application may seem to work correctly without some of these components being present, but it is undefined what will happen.

2 Development process constraints

This section provides a set of requiremets with regard to the environment, in which the application and its documentation will be developed.

Definition

- *the project* is term used for all of the listed: the runtime of Φnite with the documentation, the source code and required external libraries.

2.1 Developing the application

- Visual Studio 2012 Ultimate will be used
- C# will be used to implement the algorithms used to fulfil the main purpose of Φnite: for building finite-state machine that is equivalent to a given regular expression and for simulating machine's evaluation of a given word
- WPF (Windows Presentation Framework) will be used to create the user interface, via use of C# and XAML

2.2 Source code documentation

- All namespaces introduced in the development process have to be documented.
- All classes, interfaces and enumerated types have to be at least briefly described.
- All public properites and UI event handlers also have to be documented.
- The source code documentation will follow the standard C# documentation rules, because the Visual Studio has built-in support for creation of such documentation.
- Where the rules of C# documentation are not enough, the doxygen documentation format can be used: ex. to group classes into modules, to document namespaces, etc.
- Doxygen will be used to process the source code to create a set of web pages (which use HTML and CSS) with the complete documentation.

Write here!

3 Used technologies

3.1 The runtime

-

3.2 The development

The list of technologies for the runtime applies also to this section. Moreover, there are some extra components used exclusively for the development.

-

4 GUI mockup

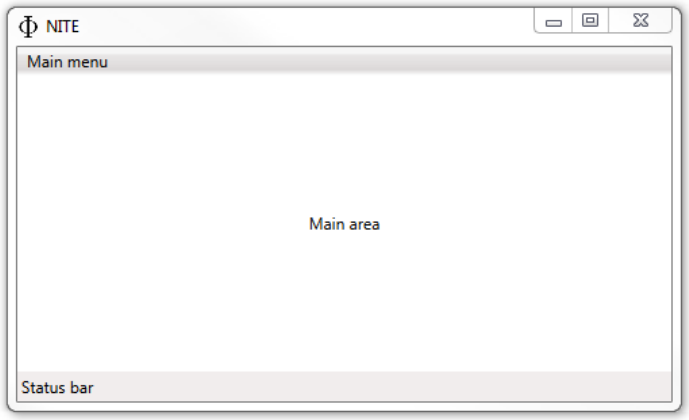

Figure 1: schema of the graphical user interface

4.1 Main menu

Main menu contains all general options that are required to be implemented, but are not connected with a specific step of calculation. Contents of the menu must not change during application operation, but its certain entries may be temporarily disabled during some phases of computation.

Main menu will contain, ex. exit option, example selection option.

4.2 Main area

Main area will contain frequently changing content, as it may have any of the following: a text field for entering a regular expression, button to proceed to the next step of computation, button to abort computation, button to display final result, a table with intermediate results, a final result, etc. Content of main area depends on context, i.e. previous actions of the user and current status of application.

4.3 Status bar

Status bar must contain one of three phrases at all times, unless there are equivalent indicators implemented (mentioned in each point):

- "busy" - if program computes the final or intermediate result.

  Equivalent of this is locking all elements of the GUI while the program is busy.

- "awaiting user interaction" - if program is not busy, but intends to be right after user gives some information that may help with further computation. The phrase may be changed to other with the same meaning. This status is intended only for computation phase, and only for those parts of computation phase that require user input.

  Equivalent of this status is displaying a new window. All information and input fields needed by the user to help the program must be in this new window. When it is closed, program resumes computation.

  Since the user feedback mechanism is optional (provided that it is of course substituted with working implementation), in certain implementation scenarios this status will never occur (or window will never be shown). If development team is able to prove that this feature would really never be used, this status indication does not have to be implemented.

- "ready" - in all other situations, for example: after start-up, after completing the computation.

  Equivalent of this status is a situation in which both previous mechanisms are implemented using second variant. If it is so, this status can be omitted.

Status bar may also temporarily contain some optional content that may help the user in completing the user interaction phase, provided that the main area is not a proper place for this content.

5 Classes

This section goes through the most important classes used in the project.

5.1 RegularExpression

```
class RegularExpression
{
string input;

List<string, Tag> taggedInput;

PartialExpression parsedInput;
}
```

5.2 PartialExpression

```
class PartialExpression
{
Role role;

PartialExpression parts;
}
```

5.3 FiniteStateMachine

```
class FiniteStateMachine
{
RegularExpression input;

RegularExpression initialState;

List<RegularExpression> states;

List<RegularExpression> finalStates;
}
```

6 Algorithms

6.1 Conversion of plain text into a RegularExpression object

6.2 Conversion of RegularExpression object into FiniteStateMachine object