UNIVERSITY OF TARTU
Faculty of Mathematics and Computer Science
Institute of Computer Science

Reimo Rebane

An integrated development environment for the SecreC programming language

Bachelor's Thesis (6 ECTS)

Supervisor: Dan Bogdanov, MSc

Author: ..... "....." June 2010
Supervisor: ..... "....." June 2010

Allowed to defence
Professor: ..... "....." June 2010

TARTU 2010

Contents

Introduction

The purpose of this work is to simplify the creation process for applications developed with the SecreC language of the Sharemind framework. More precisely, we want to simplify the implementation of data mining algorithms in those applications. The solution is a software tool that is included with the framework. The tool should help the developer by providing a specialized environment containing a set of tools for working with the framework and its languages. An integrated development environment (IDE) will be created.

Currently, the process of creating data mining algorithms with the Sharemind framework is complicated, because the tool support is insufficient for working with the SecreC language. Implementing simple algorithms is not a problem but as the complexity of the algorithms rises, the need for debugging tools will likewise grow. The development environment should also provide the opportunity to bind together some of the Sharemind framework tools such as the SecreC compiler and the code analysis framework.

Section 1 contains a short description of the Sharemind framework and the motivation why the tool developed in this thesis is necessary. In Section 2, the requirements of the IDE are given to specify the scope of this project. Section 3 gives an overview of how the various components of the system interact with one another. Section 4 consists of the specification of the final functionality of the tool with detailed descriptions of various features. The Sharemind controller interface is specified in Section 5 with a detailed description of the relevant functionality of the controller library. Finally a user's feedback to the IDE is briefly described in Section 6.

1 Preliminaries

1.1 The Sharemind framework

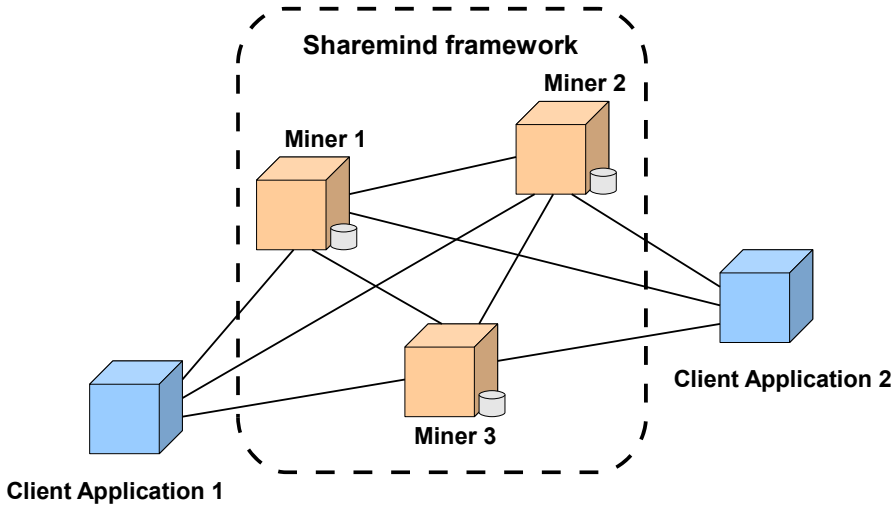

Figure 1: The Sharemind framework and controller applications

The Sharemind framework [BLW08] is a data processing system capable of performing computations on input data without compromising its privacy. The system is based on solid cryptographic foundations - secret sharing is used to preserve the privacy of the data during storage and secure multi-party computation allows the data to be securely processed. A working system consists of three miners that process and store the data and a controller providing input and coordinating the miners' work. The miners form the Sharemind virtual machine. Controller applications will implement the Sharemind controller library interface. The library interface is used to instruct the miners to run the compiled SecreC programs for performing various data mining tasks.

1.2 Existing Sharemind development tools

Applications developed with the Sharemind framework use its controller library as an interface to the framework. Two separate implementations of the library exist. The bounded version can insert data, request computations and read the results of the computations. This version of the library is meant for user applications. The unbounded version can, in addition to the previous functionality, get detailed

information about the miners' state. The latter library will be used in this work to implement the debugging functionality.

Two languages have been created for the framework - the Sharemind assembly language and the SecreC language.

The Sharemind assembly [Jag08] is a low level language that is directly executed by the Sharemind virtual machine [BLW08, BL10]. The language can be used to create applications for the framework but programming in a low level language as the Sharemind assembly can be time-consuming and error-prone. To address this issue, the SecreC language was created.

SecreC [Jag10] is a C-like programming language designed for developing privacy-preserving data mining algorithms for Sharemind applications. Currently the SecreC compiler generates Sharemind assembly code that can be run in the Sharemind virtual machine. It has been shown to be practical for a variety of data mining tasks [BJL09a, BJL09b].

1.3 Creating applications with Sharemind

Creating applications with the Sharemind framework mainly consists of two parts. Firstly, a client side application has to be developed which implements the Sharemind controller library for a communication channel with the miners. Additionally, a user interface is required for inserting new data to the miners' database, request computations on the existing data and for viewing the results of the computations.

Secondly, the data mining algorithms have to be implemented to fulfil the intended tasks on the data. The written algorithms have to be analyzed for private data leakages before execution. While Sharemind guarantees the security of the individual instruction level, leakages may occur on the algorithm level [Ris10].

1.4 Problems with current approach

The development of computation algorithms for applications would be significantly faster and easier with better development tools.

Debugging data mining algorithms is complicated and time-consuming because only some data is public, most of the computations are done with private data that has to be explicitly published. Therefore a debugging tool which would have access to all the data and could examine the current execution state, is needed.

Modern code development environments have multiple tools to aid the programmer. Examples of such tools are code highlighting which makes reading of the code more intuitive and auto-completion that completes variable and function names. Also an important part in many IDEs is the context sensitive help feature which enables the programmer to find currently relevant documentation by code

keywords or by index search. Many language specific environments have built-in language APIs.

If Sharemind is easier to develop for, it might find more use in real-world applications and lead us to information systems with significantly better privacy guarantees.

1.5 Our contribution

To ease some of the listed problems we have developed an integrated development environment in the scope of this thesis. Interfaces are supplied for working with the Sharemind virtual machine and the SecreC compiler. The environment should mainly support the writing of data mining algorithms in the SecreC language. This will be done by providing a code editor with syntax highlighting and code-completion to make the Sharemind framework languages more comprehensible. Also, tools for compiling the SecreC language into Sharemind assembly and an interface to run and debug the resulting code will be provided.

2 Requirements

2.1 Functional requirements

Project management

It is a standard feature in modern IDEs to have a project management facility for helping the developer group together the various pieces of source code for a specific project. Basic functionality will be provided that can be extended given demand by the users.

The user can create a project. The project has a name (a non-empty string) and consists of SecreC source code and compiled assembly language code files. The project information can be saved in an XML-based format.

The user can change the project name (to a non-empty string) and create new source code files that will automatically be added to the active project. Optional feature: if there is no active project, an empty project is created to contain the new source code file.

The user can add existing SecreC source code files (extension: .sc) and Sharemind assembly files (extension: .sa) to the project. Files can be removed from the project.

Code development

The programmer can select a SecreC source code file in the project for editing. It will be opened in an editor window. The editor window can perform syntax highlighting based on the current language, it has line numbers and works as any reasonably comfortable text editor.

The editor supports both SecreC and Sharemind assembly source code. It is possible to switch between SecreC code and the assembly code it is compiled into.

It is possible to find and replace text fragments within the scope of one file. There is an automatic code completion mechanism that supports completion of built-in function names.

Changes to the source code are tracked by the IDE so that the user could be alerted to save the code when the file is closed. The user should also be alerted when the code needs to be recompiled before execution.

If the user has a SecreC code file open, then it can be compiled into Sharemind assembly. There will be an output window that will display the output of the compiler, including possible errors. If compilation is successful, the resulting Sharemind assembly file will be added to the project. Compiling is a necessary step before the program can be run.

Running the code

When the user has an assembly source code file open, he or she can order the IDE to run it on the default miner configuration. The program will be uploaded to the miners and executed. The output from the program will be displayed in the output window. If an error occurs, the program is stopped and the debugger is automatically started. The current instruction is shown and the current contents of the stack and registers are displayed. The error message is displayed in the output window.

Executing SecreC code assumes that it is first compiled into Sharemind assembly. The assembly code is executed as before with a possible exception of invoking the SecreC debugger instead of the Sharemind assembly debugger. See the relevant section for details.

Debugging the code

When editing a Sharemind assembly script, the user can start it in debugging mode. In this mode, the user can perform a step-by-step execution of assembly code, stop the program on user-set breakpoints and examine the contents of the stack and registers. A special version of the Sharemind controller library will be used to control execution and examine the contents of the stack and the registers.

The requirement of debugging SecreC code directly will be considered during further development, after the completion of this thesis, due to the expected complexity of the feature. Given enough demand for such a feature, it will be implemented.

Help

The user can view relevant help documentation about the environment and the languages. In the help interface, the user can search and browse the contents of the documentation. The user can also activate the help interface in a context sensitive way so that the relevant page of the given keyword will be opened.

The contents of the documentation will not be fully created in the scope of this thesis. The help engine and some example pages are developed.

Configuration

The user can specify the location of the installation of the SecreC compiler in the system. Given that SecreC installations are not yet standardized, the user manual of the current version of Sharemind has to be used for specific requirements on how to find and run the compiler.

In order to run the resulting Sharemind assembly, the IDE needs to have the configuration of the miners. It should be possible to choose between several miner

configurations. The details of what needs to be in the configuration can again be found in the documentation of Sharemind.

2.2 Non-functional requirements

Platforms

The IDE should run on all platforms that are supported by both the Qt framework and the Sharemind framework. The following platforms should be supported:

- Linux (32 and 64-bit)
- Mac OS X (10.5, 10.6)
- Microsoft Windows (XP, Vista, 7)

Compatibility with the Sharemind framework

The IDE should be compatible with the up-to-date controller library and the SecreC compiler. If changes are made to the Sharemind framework affecting those components, the IDE should be modified accordingly.

3 Architecture

3.1 The technology

Multiple possibilities were taken into account for developing the tool. Firstly, writing a plugin for a widely used IDE was considered. It would have eased the development substantially because a lot of the required functionality would already be there. Eclipse was considered for the development environment because of its good plugin features. The idea was rejected in favor to other solutions because of the complexity involved in making a Java interface for the controller library that is written in C++.

The solution that seemed most adequate was to develop the IDE in C++. Qt was chosen as the core library because of its wide range of useful functionality and for its platform independency. Writing a code editor component in the scope of this thesis wouldn't have been reasonable and therefore an existing component was selected. At first, QCodeEdit was chosen as it is easy to install and had all the basic components needed. Later, this component was replaced with QScintilla because of its higher customization functionality and its wider range of features. QScintilla is a Qt port for a widely known code editing component called Scintilla. It provides syntax highlighting, code-completion, code folding, search and replace and basic text editor tools.

3.2 The structure

The IDE can be divided in the following components based on their roles:

- Project management
- Code development
- Tool support
- Code execution
- Code debugging
- Help module
- Configuration module

Figure 2 illustrates the various components of the IDE and their relationships with the Sharemind framework. The compilation of SecreC code is done by calling the SecreC compiler provided by the framework. The interface to the analysis tool will be a later addition to the IDE. The Sharemind controller library is used for communication with the miners. Both code execution and code debugging is done with the library.

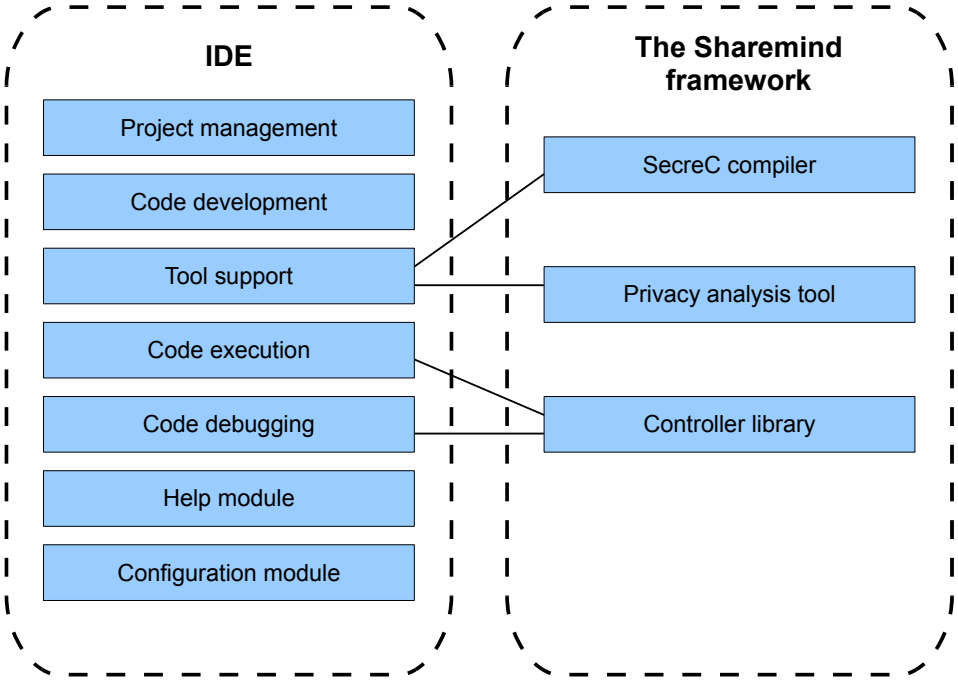

Figure 2: The Structure

3.3 Deployment scenario

In algorithm development, the IDE will be used to create the SecreC data mining algorithms for the application. The code execution and code debugging interface is used for testing the algorithms. In addition, the written SecreC code can be analyzed for data leakages that may occur on the algorithm level by the analysis tool provided by the Sharemind framework. When the algorithms have been tested and proven secure, they can be used by the client applications to perform computations on data inserted to the databases.

Figure 3 shows how the SecreC compiler of the Sharemind framework is used to build SecreC code into Sharemind assembly code. The built assembly scripts are uploaded to the miners by the IDE using the Sharemind controller library interface. The library interface is also used for running and debugging the uploaded scripts in the virtual machine. The results and debugging info are returned to the IDE.

Figure 4 shows how the data mining algorithms that have been uploaded to the miners are used by the client applications to perform computations on the inserted data. The communication with the Sharemind virtual machine is performed through the controller library. The results are returned to the client application.

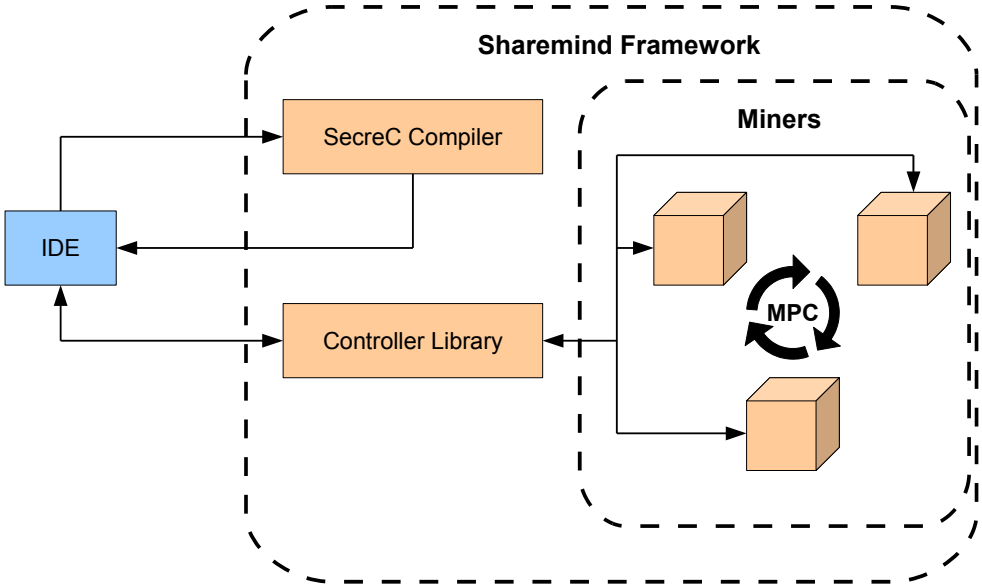

Figure 3: Creating and debugging algorithms with the IDE

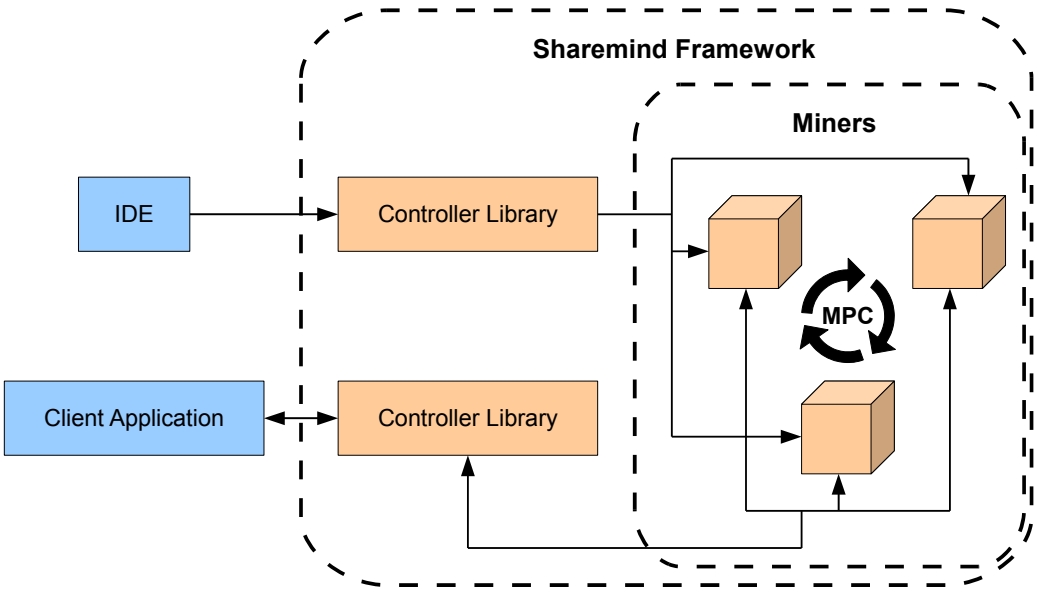

Figure 4: Deploying the algorithms in a user application

4 Functionality

This chapter describes the features of the IDE and how they are implemented.

4.1 Project management

The project manager consists of the project model which is a tree structure and the project view for an interface with the structure. There are three types of nodes in the project model: project, folder and file nodes. The project nodes are the root project items, the folder nodes represent virtual folder structures which sort files according to their extensions and the file nodes represent files. The structure is constructed from a XML based project file which can be stored on disk. The Qt XML module is used for writing and reading XML files.

Creating a new project

To create a new project the user opens the 'File' menu and selects 'New Project' from the menu. Alternatively, the user can right-click on the project view for a context menu and select 'New Project' from there. A new project entry will be created in the underlying project model and shown in the project view.

Opening an existing project

To open an existing project the user opens the 'File' menu and selects 'Open Project' from the menu. If preferred, the user can instead right-click on the project view for a context menu and select 'Open Project' from there. A file dialog will appear and after choosing a project file, a project entry will be created in the project model with the information read from the project file. If a project loaded from the selected project file already exists in the project manager, it will be removed before creating the new entry.

Saving a project

To save a project the user can right-click on a project node for a context menu and select 'Save Project' from there. If the project was previously loaded from a file, it will be saved there. Otherwise a file dialog will appear where the saved project file can be chosen. Alternatively, the user can choose 'Save Project As...' from the context menu which always asks the user for the saved project file.

The XML structure of the saved project files contains a 'project' tag with any number of children 'file' tags. The 'project' tag has two parameters of the obligatory 'name' parameter containing the project name shown in the project manager and the optional 'defaultBuildPath' parameter describing the default path for the compiled assembly files. The 'file' tag has also two parameters of the mandatory

'path' parameter containing the path to the file and the 'source'/'target' tags depending on the file type which link SecreC source files and Sharemind assembly target files.

Example project file:

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="Project" defaultBuildPath="/path/assembly">
    <file path="/path/file1.sc" />
    <file path="/path/file2.sa" />
    <file path="/path/file3.sc" target="/path/file4.sa" />
    <file path="/path/file4.sa" source="/path/file3.sc" />
</project>
```

4.2 Code development

The QScintilla library is used to implement syntax highlighting, code-completion, search and replace and some basic tools of the code editor. Compilation is done with the SecreC compiler provided by the Sharemind framework.

Syntax highlighting

Syntax highlighting is supported for the SecreC language and the Sharemind assembly language. The highlighting is done by comparing the code to sets of language specific keywords when a code file is opened. The keywords are supplied by a language specific API class to the appropriate lexer which processes the code.

Code-completion

The completion is done on two levels. Document-level completion is done by searching the document for previously used keywords. API level completion is done by comparing the entered keyword to a fixed set of language specific keywords. The keywords are supplied by a language specific API class to the appropriate lexer which forwards the keywords to the auto-completion call. The user can press the Ctrl + Space keys to activate the code-completion. When activated, a list of possible choices that the user can choose from will appear. The user can use the up and down keys on the keyboard to cycle through the possible choices. The keyword is completed using the Enter key.

Searching and replacing text

Basic text searching in the currently selected document is available forwards and backwards. Optionally, case sensitive, whole word only and regular expression

search strings are also supported. The strings that were found can be replaced with the inserted replace string one by one or all at once. The search and replace component is not a part of the QScintilla library but uses its functionality to implement the logic.

Compiling SecreC into Sharemind assembly

The compiler has to be configured before it can be run. The detailed procedure how to configure it is described in Section 4.6.

To compile a SecreC file into a Sharemind assembly file the user can right-click on the appropriate file in the project manager and select 'Build File'. If preferred, the user can instead click the 'Build' button from the tool bar if desired SecreC file is the currently open file. The IDE will locate and execute the compiler based on the given configuration. The compiler will try to compile the selected file and, if successful, an assembly file will be inserted into the project of the source file. If source file is not in any projects the assembly file will be inserted with no project. If errors occurred during compilation they will be shown in the 'Compiler' output tab. For each project, a default build path can be specified. Otherwise the built files will be created in the source file directory.

Code analysis tool

The interface to the SecreC code analysis framework will be added to this project in future releases.

4.3 Running the code

The Sharemind controller library interface is used to implement the script execution functionality. The specification of the interface is described in Section 5. A controller library instance will be created before a script is executed and the instance will be run in a separate thread.

Running Sharemind assembly code

To run a Sharemind assembly script the user can right-click on the file in the project manager and select 'Run Script' from there. Alternatively, the user clicks the 'Run' button on the tool bar to run the currently open Sharemind assembly file. The script will be uploaded to the miners and executed. Debugging will be initialized if an error occurred, otherwise the results for the script are returned and shown in the 'Results' tab of the output section of the IDE. Each result entry consists of a name, variable type and value.

Optionally, the user can specify runtime parameters before the execution of

the script. To set the parameters for a Sharemind assembly code file, the user can right-click on the appropriate file in the project manager and select 'Set Runtime Arguments' from the opened context menu. An options dialog will appear where the user can insert the required parameters in the table. Each runtime argument consists of a name, variable type and value. The arguments will be stored until the IDE exits. The parameters will be passed to the Sharemind controller prior to the execution of the script.

Running SecreC code

SecreC code is compiled into Sharemind assembly before execution. The built Sharemind assembly code will be run as described above.

4.4 Debugging the code

Initializing the debugger for the Sharemind assembly code

Debugging can be initialized in two ways. Firstly, if an error occurs when running a script, the controller will pause at the instruction executed at the time of the error. The error line in the script and the current state are returned to the IDE. The script will be opened, if it is not already open, and the relevant line will be located and marked.

Alternatively, the user sets breakpoints to the script before execution to start debugging. The breakpoints are set and removed by clicking on the line numbers margin area in the code editor. The running script will be paused at the breakpoints with the current line and state returned as before.

Debugging Sharemind assembly code

While in debugging mode and the current script is paused, three options are available: continuing, stepping and stopping of the script. Each are executed by clicking on the appropriate button on the tool bar.

Continuing the script resumes the execution of the script until an error, a breakpoint or the end of the script occurs. If the script is finished, the results of the script are returned. Otherwise, the current line and state are returned to the IDE.

Stepping the script executes the next instruction in the script and then pauses it. Results are returned if the end of the script is reached, otherwise the current line and state are returned to the IDE.

Stopping the script halts the current script. No result or state is returned.

Whenever the script is paused, the user can inspect the current stack and registers. The information is shown in the debugging output view that is only displayed

when debugging mode is active. The values on the stack are shown as a list. The registers are shown in a table where each register is represented with a name, a variable type and a value.

Debugging SecreC code

Debugging of SecreC code is currently not possible as the assembly code does not yet contain debugging information.

4.5 Help

The help interface of the IDE is implemented using the Qt Help module. Qt help files used by the module are compressed archives of the HTML based documentation with any necessary data, for example image and styling files, included. The help viewer implements the Qt Webkit module.

Activating the help module

To activate the help interface the user opens the 'Help' menu and selects 'Help' from there. If preferred, the user can instead press the F1 key on the keyboard for the same result. When activated, the help module compares the currently selected word in the code editor with a list of predefined keywords. If a keyword match is found, the help interface is opened on a page referenced by the keyword. If no match is found or there is no currently open editor, the help module will be shown normally with no additional pages opened.

Browsing help pages

The help interface is divided into two sections - the controls and the view. Using the controls the user can browse the contents in several ways.

The 'Contents' tab categorizes the information in a tree structure based on inheritance of the information. The user can double click on any entry to open the corresponding page.

The 'Index' tab contains the predefined keywords that can be filtered by the user by inserting a filter text to the input field. Double clicking on any keyword opens a page referenced by the keyword.

The 'Search' tab contains the search input field and the results view. The user can perform a search of the help contents by writing a query in the input field and clicking 'Search'. The results of the query are shown in the results view of the same tab. Optionally, the user can select use the advanced search options for a more specific query.

Any page opened is shown in the help view. The help view supports multiple

tabs containing simultaneously opened pages. The pages can be browsed like standard HTML pages with internal links.

4.6 Configuration

The configuration settings that need to be stored beyond the lifetime of a single IDE runtime will be saved using the QSettings class of the Qt Core module. The class provides persistent platform-independent application settings.

Configuration of the SecreC compiler

To configure the SecreC compiler the user opens the 'Tools' menu and selects 'Options'. When selecting the 'Compiler' tab from the opened options dialog the user can then modify the Sharemind framework path and the additional classpath. The Sharemind framework path will be used to search for the SecreC compiler. The compiler assumes that the ANTLR JAR file is in the additional classpath directory.

Configuration of the miners

To configure the miners the user opens the 'Tools' menu and select 'Options'. After selecting the 'Runtime' tab, the user can configure the addresses and ports of the miners. Alternatively, the user can select a controller configuration file by clicking on the file dialog button. If the selected configure file was successfully loaded the appropriate fields will be populated with the information read from the file. An error message will be displayed if the file could not be read or is in an unknown format.

5 Sharemind contoller interface

The Sharemind controller library interface functionality relevant to this project consists of functions related to assembly script execution. The library calls are sorted in two main categories of running the scripts and debugging the scripts. The instructions were used to implement the code running and debugging features.

5.1 Running scripts

General

```
bool runScript (const string& name,
        const vector<RegisterDescriptor>& params,
        vector<RegisterDescriptor>& results);
```

Description: Function for running a script. The script must be installed in all the miners.
Parameters:

- **name** - script name
- **params** - script parameters, containing the variables required by the script
- **results** - the returned results will contain results published by the script

Return: True, if the script is successfully finished, otherwise false

```
bool uploadScript (const string& localfile,
        const string& destfile);
```

Description: Function for uploading a script. The script will be uploaded to all the miners and can then be run.
Parameters:

- **localfile** - the local script file path
- **destfile** - the remote script file path where the script will be installed

Return: True on success, false on failure

5.2 Debugging scripts

General

```
bool debugScript (const string& name,
        const vector<RegisterDescriptor>& params,
        ScriptState& state);
```

Description: Function for debugging a script. Debugging will be initialized when a breakpoint or an error occurs during the execution of the script. The script must be installed in all the miners.
Parameters:

- **name** - script name
- **params** - script parameters, containing the variables required by the script
- **state** - the returned script state containing information about where and why the script was stopped

Return: True, if the script is paused or successfully finished, otherwise false

```
bool continueScript (ScriptState& state);
```

Description: Function for continuing the current script when it is paused. The script will be run until a breakpoint, an error or the end of the script occurs.
Parameters:

- **state** - the returned script state containing information about where and why the script was stopped

Return: True, if the script is paused or successfully finished, otherwise false

```
bool stepScript (ScriptState& state);
```

Description: Function for continuing the current script when it is paused. The script will be run for one instruction or until an error or the end of the script occurs.
Parameters:

- **state** - the returned script state containing information about where and why the script was stopped

Return: True, if the script is paused or successfully finished, otherwise false

```
bool resetScript ();
```

Description: Function for resetting the current script to the starting point.
Parameters: None
Return: True on success, false on failure

```
bool getScriptResults (
        vector<RegisterDescriptor>& results);
```

Description: Function for getting the results from the current script.

Parameters:
- **results** - the returned results will contain results published by the script

Return: True on success, false on failure

Breakpoints

```
bool addBreakpoint (uint32 linenumber);
```

Description: Function for adding a breakpoint to the current script.
Parameters:
- **linenumber** - the line number of the breakpoint

Return: True on success, false on failure

```
bool getBreakpoints (vector<uint32>& linenumbers);
```

Description: Function for retrieving the breakpoints for the current script
Parameters:
- **linenumbers** - the returned line numbers of the breakpoints

Return: True on success, false on failure

```
bool removeBreakpoint (uint32 linenumber);
```

Description: Function for removing a breakpoint from the current script
Parameters:
- **linenumber** - the line number of the breakpoint

Return: True on success, false on failure

```
bool resetBreakpoints ();
```

Description: Function for resetting the breakpoints for the current script
Parameters: None
Return: True on success, false on failure

Gathering data

```
bool getStackSize (uint32& count);
```

Description: Function for retrieving the stack size of the current script. The script has to be paused.
Parameters:
- **count** - the returned stack size count

Return: True on success, false on failure

```
bool getStackData (val_vector_t& data, uint32 start,
        uint32 count);
```

Description: Function for retrieving the the stack data of the current script. The script has to be paused.
Parameters:

- **data** - the returned data vector containing the values on the stack
- **start** - offset position on the stack for the returned data vector
- **count** - number of elements returned from the stack from the given offset position

Return: True on success, false on failure

```
bool getRegisterList (
        vector<RegisterDescriptor>& globalregs,
        vector<RegisterDescriptor>& localregs);
```

Description: Function for retrieving the list of active registers of the current script. The data of the registers will not be returned. The script has to be paused.
Parameters:

- **globalregs** - the returned global scope registers
- **localregs** - the returned local scope registers

Return: True on success, false on failure

```
bool getRegisterData (RegisterDescriptor &reg,
        uint32 start = 0, uint32 count = 0);
```

Description: Function for retrieving the data of a register of the current script. The script has to be paused.
Parameters:

- **reg** - the given register to be populated with data and returned
- **start** - offset position of the returned data vector, should be zero if data is not a vector
- **count** - number of elements returned from the data vector, should be zero if data is not a vector

Return: True on success, false on failure

```
bool getRegisterSize (RegisterDescriptor reg,
        uint32& vecsize);
```

Description: Function for retrieving the data size of a register of the current script.
Parameters:

- **reg** - the given register
- **vecsize** - the returned size of the data of the given register, it is not set when data is not a vector

Return: True on success, false on failure or when the data of the given register is not a vector

6 User feedback

A master's student of Stockholm KTH Royal Institute of Technology, Abu Hamed Mohammad Misbah Uddin, is privately searching anomalies in network logs for his master's thesis [Udd10]. For his work he has used the Sharemind framework, the SecreC language and the IDE developed in this thesis. The algorithms developed for his work were implemented in the SecreC language. We helped to set up the IDE on his computer and it was used for most of his programming work. He is defending his master's thesis in Spring 2010.

Uddin mentioned in his feedback letter that he was pleased with the IDE and although he experienced some unexpected crashes, the problems with it were infrequent. He also requested a feature for storing a list of recent projects which can then be easily opened. The request has been added to future functionality requirements and the stability of the IDE has been greatly improved in the later versions of the environment.

The letter containing the feedback from Uddin is included in Appendix 2.

Conclusion

In the scope of this thesis we have developed an integrated development environment. The IDE will support development of data mining applications for the Sharemind framework. It provides the developer with features like project management, code development, code execution and debugging, interfaces with other Sharemind tools and a help module. Syntax highlighting, code-completion and some other tools have been implemented in the code editor for the SecreC and the Sharemind assembly language.

The future work with the IDE will be to add additional features depending on the user requests and to fix reported bugs. Also it has to be modified as changes occur in the Sharemind framework.

The IDE has been successfully applied in the creation process of a data mining application by a master's student of Stockholm KTH Royal Institute of Technology.

Integreeritud arenduskeskkond SecreC programmeerimiskeelele

Bakalaureusetöö (6 EAP)

Reimo Rebane

Resümee

Sharemind on raamistik, mis võimaldab teha privaatsust säilitavaid arvutusi. Sharemind koosneb kolmest andmekaevandaja rakendusest ja juhtimisteegist. Andmekaevandajad, moodustades Sharemindi virtuaalmasina, teevad olemasolevate andmete peal privaatseid arvutusi. Juhtimisteegi abil saavad teised rakendused andmekaevandajate tööd kontrollida ning nende andmebaasidesse uusi andmeid sisestada.

Sharemindi raamistikule on loodud kaks programmeerimiskeelt: Sharemindi assemblerkeel ja SecreC keel. Sharemindi assemblerkeel on madala taseme keel, mida käivitatakse otse Sharemindi virtuaalmasinas. Kuigi ta lubab teha privaatsust säilitavaid arvutusi teostavaid rakendusi, on rakenduste tegmise protsess keeruline, sest madala taseme keeles programmeerimine on aeganõudev ning vigadele avatud. Selleks, et arendajad ei peaks assemblerkeeles programmeerima, loodi SecreC keel. SecreC on C-keele laadne programmeerimiskeel, mis võimaldab luua andmekaevandusalgoritme Sharemindi raamistiku rakendustele. SecreC kompileeritakse Sharemindi assemblerkeeleks, mis seejärel käivitatakse virtuaalmasinas.

Klientrakendused kasutavad juhtimisteeki andmekaevandaja rakendustega suhtlemiseks. Praegune Sharemindi klientrakenduste loomise protsess koosneb kasutajaliidese loomisest ja andmekaeve algoritmide realiseerimisest. Kasutajaliideses peab olema võimalus andmeid lisada, esitada päringuid olemasolevate andmete peal arvutuste tegemiseks ja arvutustulemusi kuvada. Järgnevalt tuleb luua andmekaevandusalgoritmid, mis realiseeritakse SecreC keeles. Loodud algoritme tuleb testida ning kontrollida, et lekkivaid andmeid ei oleks. Kuigi Sharemindi raamistik garanteerib andmete privaatsuse instruktsiooni tasemel, võivad lekked siiski esineda algoritmi tasemel.

Käesoleva töö eesmärgiks on lihtsustada rakendustele algormitmide realiseerimise protsessi SecreC programmeerimiskeeles. Meie panususena on loodud integreeritud arenduskeskkond, mis toetab programmeerijat, pakkudes talle mitmeid tööriistu, mis peaksid lihtsustama programmeerimist nii SecreC keeles kui ka Sharemindi assemblerkeeles. Samuti on arenduskeskonnas liides SecreC kompileerimiseks assemblerkeeleks, kasutades Sharemindi raamistiku SerceC kompilaatorit. Juhtimisteek lubab virtuaalmasinas assemblerkeelt käivitada ning siluda. Silumise töörežiimis saab uurida virtuaalmasina pinu ja registreid.

Arenduskeskond on programmeeritud C++ keeles ning kasutab Qt raamistiku suure osa funktsioonide realiseerimiseks. Qt raamistik pakub laialdast funktsionaalsust ning on platvormist sõltumatu. Samuti on kasutatud QScintilla teeki keskse koodiredaktori loomiseks. QScintilla on Qt raamistikule kohandatud versioon Scintilla teegist, mis on laialdaselt tuntud lähtekoodi redaktori teek.

References

[BJL09a] Dan Bogdanov, Roman Jagomägis, and Sven Laur. Privacy-Preserving Histogram Computation and Frequent Itemset Mining with Sharemind. Technical report, Cybernetica research report T-4-8, 2009.

[BJL09b] Dan Bogdanov, Roman Jagomägis, and Sven Laur. Sharemind: a practical toolkit for privacy-preserving data mining. *Submitted*, 2009.

[BL10] Dan Bogdanov and Sven Laur. The design of a privacy-preserving distributed virtual machine. In *Collection of AEOLUS theoretical findings*. University of Patras (EL), 2010.

[BLW08] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A Framework for Fast Privacy-Preserving Computations. In *ESORICS*, pages 192–206, 2008.

[Jag08] Roman Jagomägis. A programming language for creating privacy-preserving applications, 2008.

[Jag10] Roman Jagomägis. SecreC - a privacy-aware programming language. Master's thesis, University of Tartu, 2010.

[Ris10] Jaak Ristioja. An analysis framework for an imperative privacy-preserving programming language. Master's thesis, University of Tartu, 2010.

[Udd10] Abu Hamed Mohammad Misbah Uddin. Privacy Preserving Botnet Detection using Secure Multi-party Computation. Master's thesis, KTH Royal Institute of Technology / University of Tartu, 2010.

Appendix 1: The software

Introduction

This thesis is accompanied by software for demonstrating some of the functionality of the IDE created in this thesis. The source code and a binary version of the IDE are provided. For testing purposes, the miner applications, the SecreC compiler and some sample SecreC programs are also included. The sample programs supplied have been developed in a separate work [Jag10]. The binary versions of the applications are built for the Microsoft Windows platform, including the binary of the IDE. While they have been tested on several versions of Windows, the author cannot guarantee that all of the executables can be successfully run.

Installing the software

The `SecreCIDE-kit.zip` should be extracted in a desired folder to create the following subfolders.

1. `sample-programs`: Sample SecreC programs for testing purposes.
2. `secrec-compiler`: Contains the SecreC compiler code and binaries. The ANTLR library required by the compiler is also included.
3. `SecreCIDE`: Contains the SecreCIDE executable for Microsoft Windows.
4. `SecreCIDE-src`: Contains the source code and necessary files to build the SecreCIDE.
5. `sharemind-vm`: Contains the Sharemind virtual machine miner applications.

Setting up SecreCIDE

Start the SecreCIDE application in the `SecreCIDE/bin` folder. The following steps have to be taken to configure it.

1. Open the options dialog (`Tools->Options`).
2. Browse or enter the `secrec-compiler` path to 'Sharemind Path' input field.
3. Browse or enter the `secrec-compiler` path to the 'Additional Classpath' input field.
4. Click 'OK' to save the changes.

The IDE should now be able to compile and run SecreC programs.

Compiling SecreC programs with SecreCIDE

Take the following steps to compile a SecreC program. Example SecreC code files can be found in the `sample-programs` folder.

1. Open a SecreC code file (with the *.sc suffix) from the file menu (`File->Open`).
2. Build the SecreC code file by clicking on the build button on the tool bar.
3. The compiled file will added to the files list on the left side of the screen.

Running Sharemind assembly programs with SecreCIDE

The Sharemind virtual machine has to be running before attempting to execute the programs. To start the virtual machine run the `miners.bat` script in the `sharemind-vm` folder.

To run the built Sharemind assembly files take the following steps.

1. Open a built Sharemind assembly code file (with the *.sa suffix) from the file menu (`File->Open`).
2. Run the file by clicking on the run button on the tool bar.
3. The running log is shown in the 'Controller' output tab. If any results are returned they will be shown in the 'Results' output tab.

For a more specific case with runtime arguments take the following steps.

1. Open the `histogram.sc` file (`File->Open`) from the `sample-programs` folder.
2. Build the opened SecreC file.
3. Right-click on the built `histogram.sa` file and select 'Set Runtime Arguments'.
4. Three parameters have to be entered in the runtime arguments table. Additional parameter rows can be added by clicking on the button in the upper right corner.
   (a) Name: table, Type: public string, Value: answers100
   (b) Name: column, Type: public string, Value: threeanswers
   (c) Name: choices, Type: public int, Value: 3
5. The parameters can be saved by clicking on the 'OK' button.
6. Select the `histogram.sa` file by double-clicking on it.
7. Run the script by clicking on the run button on the tool bar.
8. The results of the script will be shown in the 'Results' output tab.

The histogram script with the given parameters returns the frequencies of the answers in an example survey with three possible choices and an answer set of 100 entries.

Debugging Sharemind assembly programs with SecreCIDE

The Sharemind virtual machine has to be running before attempting to debug the programs. To start the virtual machine run the `miners.bat` script in the `sharemind-vm` folder.

Take the following steps to start debugging a Sharemind assembly file.

1. Open a built Sharemind assembly code file (with the *.sa suffix) from the file menu (`File->Open`).
2. Click on the line number margin area to set a breakpoint on that line.
3. Run the file by clicking on the run button on the tool bar.
4. The script is run until the breakpoint is reached. The debug output section is displayed with the contents of the stack and registers.
5. Continue the script by clicking on the continue button on the tool bar.
6. The running log is shown in the 'Controller' output tab. If any results are returned they will be shown in the 'Results' output tab.

Appendix 2: A user feedback letter

from: Abu Hamed Mohammad Misbah Uddin
to: Reimo Rebane
date: Fri, May 21, 2010 at 11:48 AM
subject: Re: SecreC and SecreCIDE

Hi,
First of all, I am eternally grateful for SecreCIDE. I will answer your questions briefly. If you need more information you can further ask me.

> 1. What is Your work, what are You doing?

The title of my work is: Privacy Preserving Botnet Detection using Secure Multi-party Computation. But basically what I have done is secure set intersection using MPC. The key challenge was performance. A simple protocol took 1.5 hrs to give the correct result for 128 rows. Vectorization was mandatory. An O(n^2) vectorization was developed which gave the same result with the same set in 3 minutes. The current version has O(n*logn*logn) complexity, it gives result in 1.1 minutes.

> 2. How did You use the SecreC language, how the SecreCIDE?

All of my programming was implemented in SecreC. When I had to achieve O(n*logn*logn) complexity I used python for automatic code generation but its result was a SecreC code as well. I used SecreCIDE for most of the part.

> 3. What was good about them?

I can't imagine programming in Sharemind assembly. The internal functions of SecreC was a great help. SecreCIDE is a good IDE considering that it is still in beta phase.

> 4. What was bad about them?

There was a problem understanding to the syntax of Secrec during the first learning phase since it has no manual. Sometimes SecreCIDE crashed unexpectedly. But this problem was not frequent. It would be

nice if SecreCIDE could open with the projects loaded so that we don't need to load the projects every time we start SecreCIDE.

> 5. Any other comments?
For the moment, thank you for the IDE. I will try to give some more feedback about SecreCIDE later.

Sincerely,
Uddin