Seminar Report

Lava

Michael Staudacher
michael.staudacher@student.uibk.ac.at

31 July 2015

Supervisor: Alexander Maringele

Abstract

This report gives an introduction into the experimental programming language Lava. The associated programming environment and the features of the language are shown. It contains an example to bring you the language nearer and the differences to other programming languages are described.

Contents

1 Introduction

Lava is an experimental, object-oriented programming language. It is an OpenSource project.[1] Details about the history and the vision of Lava are described in section 2. The associated programming environment LavaPE (Lava Programming Environment) uses a structure based editor instead of a text based one. The program-structure is created by clicking on Toolbuttons. Because of this, the user do not need to learn language specific syntax rules and the chance of producing syntax errors is minimized. Other mistakes, like variables with a wrong type, are prevented and mistakes like uninitialized or null value variables are directly shown to the user. Text entries are only for comments, constants and new identifiers necessary. Declarations are shown in a tree structure. Changes on these declarations are automatically put to all references. Design Patterns and Frameworks are supported through packages and interfaces. The Lava Programming Environment is shown at section 3 and the features of the language are described at section 4. How to create a program in Lava and an example of a program which generates the lyrics to the song "99 Bottles of Beer" as an output, is shown at section 5. The differences to well known programming languages and to other visual programming languages are mentioned at section 6. Then the current status is described at section 7 and a final summery of Lava is given at the Conclusion 8. [1]

2 History and Vision

The experimental language Lava was developed by Dr. Klaus D. Günther and his wife Irmtraut Günther, at Fraunhofer-Institut in Darmstadt (Germany). The early preview release 0.1 was released at April 2000. Lava wants to simplify and accelerate the software production and software maintenance processes. [2]

Writing a program character by character in a texteditor is outdated. Programs should be built without detailed knowledge about the syntax "in Lego Style"[2] with a structure based editor or out of available software components. Software components like classes are the highest level of granularity of the building blocks. This concept has to be simple, easy to learn and intuitive to be accepted by the software developers. Lava tries to rise the productivity, reduce the error rate and provide reusability of the software blocks.

The programs are created, edited and stored as "abstract syntax trees" and not as human readable text files. Only for better understanding and easier editing, the program is shown to the user in some kind of textual representation. [3]

3 Lava Programming Environment

Lava programs are "constructed" using point/click/drag/drop/cut/copy/paste and menu seletion operations, also for the executable parts of the programs. In the Lava Program-

[1] `https://sourceforge.net/projects/lavape/?source=navbar`

[2] `http://www.tecchannel.de/news/themen/business/407165/lava_programmieren_im_lego_stil/`

ming Environment (LavaPE), there are two dominating views, the ”declaration view” and the ”exec view”. [4]

3.1 Declaration view

The declaration view is used to create and edit Lava entities. It shows the program as a tree view. There are three operations toolbars with buttons which allow the user to create the program structure. Only the buttons which would provide a valid operation at the current situation are enabled. If you put the cursor on the button, a short description of the button is shown. The three toolbars are described in the following.

3.1.1 Tree operations toolbar

The tree operations toolbar (Figure 1) allows the user to navigate through the tree view, expand/open sub-trees, check the Lava program file, navigate through comments and errors and run the current program.

Figure 1: Tree operations toolbar

3.1.2 Declarations toolbar

The Declarations toolbar (Figure 2) allows the user to specify new ”include files” and declare packages, initiators, classes, implementations, component object specifications, component object implementations, sets of objects, enumerations, virutal types, member functions, member variables, enumeration items and make GUI services.

Figure 2: Declarations toolbar

3.1.3 Details toolbar

The Details toolbar (Figure 3) allows the user to open the properties of the Lava entities or the executable parts of it, add comments, navigate through declarations, implementations and references, override and insert things and editing GUI specifications.

Figure 3: Details toolbar

3.2 Exec view

The exec view shows executable Lava code in a so-called ”rich edit view” window as normal text.[3] Normal text editing is not possible in Lava, but the ”structure editing” mode allows you to insert, delete, replace, etc. only whole syntactic contructs. This is much less error-prone than text based editing.

Executable code can be created by clicking on several buttons in the exec view. The primary keywords like declare, set, if and much more can be chosen in a toolbar and contains ”placeholders” after beeing selected. Those placeholders are <expr>, <func>, <stm>, <var>, <type>, <set> ... for expressions (for example a variable), functions (for example a user created function or the messageBox function)... and must be replaced by the user. It is only possible to change these placeholders to valid values, for example with a combobox for variables.

3.3 GUIs

In Lava it is possible to generate a Graphical User Interface (GUI). The GUI can be edited graphically and the user sees the acutal GUI before running the program.

3.4 Generating HTML file

Lava supports the generating of HTML files which shows the program including the executable code in a structured form. This can easily be done by clicking on the menu entry ”File” and then on ”Generate single HTML file” or on ”Generate linked HTML files” to produce multiply, linked HTML files.

4 Features

Lava is intended to be a rapid application development (RAD) language. That means that it should be possible to produce Lava programs fast and to learn the language should be easy. The features of Lava are described in this section.

4.1 Point-and-Click

Lava programs are not written like other programs, they are produced by clicking on several buttons in Lego-like fashion. The executable parts of the program must also be created in this way. The Entities like classes and implementations are created in the “declaration view” and the executable parts are created in the “exec view”. Details about these two views are described at section 3.

4.2 Structure editor

A structure editor offers the user a lot of benefits compared to a text based editor. The most programming languages have their own syntax and a software developer which uses

[3] `http://lavape.sourceforge.net/doc/html/EditExec.htm`

the language have to learn the syntax of it. Lava does not even have a textual syntax. Only minimal text entries like comments, constants and new identifiers are allowed to the user. Everything else (class, function, variable, ...) is generated with the structure editor. It is not possible to change every part of the shown code, only complete syntactic units in the Lava structure editor can be selected and changed. Drag-and-Drop/Cut-Copy-Paste copy/move these units is only allowed at places where they are acceptable. If you need a variable, function or a type, all possible options for this case are displayed in a combo box.

Context-related errors are shown to the user at the earliest possible moment. After every editing, the edited part of the program must be valid, otherwise the user have to fix it before he can continue.

Lava automatically maintains all references if you rename a Lava entity or if you move it via drag-and-drop to an other place. If you add, remove or permute parameters of a function, the changes take effect at the whole program. Automatic maintenance is also used for changes of implementations, derived classes and packages.

Because of this, syntax errors like unbalanced parentheses, misspelled keywords or other context-free syntax errors are not possible.[4]

4.3 Detecting errors

Lava prevent or detect errors already at programming time in many cases and show them to the user as red colored text or in form of a messageBox. Mistakes in the program logic or infinite recursive loops could not be detected automatically.

The use of uninitialized or null value variables are staticly checked at programming time, because program crashes caused by those mistakes can be hard to find.[5]

Lava uses a storage management based on reference counts, because this releases storage occupied by objects at the earliest possible moment.[6]

The type cast concept of Lava provides a type-safe and consistent way without having to rely on run time type checks.[7]

4.4 Refactoring

Refactoring changes the internal structure of the code without changing the external behavior. Lava has a built-in refactoring support since the beginning.[8] The different kinds of refactoring are described in this subsection.

4.4.1 Renaming

If any property of a Lava entity (classes, functions, variables etc.) is changed, this takes effect to all references in the whole program. The internal identifiers are not changed,

[4] `http://lavape.sourceforge.net/doc/html/StructEdAdvant.htm`
[5] `http://lavape.sourceforge.net/doc/html/IniChecks.htm`
[6] `http://lavape.sourceforge.net/doc/html/PointersRefCounts.htm`
[7] `http://lavape.sourceforge.net/doc/html/TypeCasts.htm`
[8] `http://lavape.sourceforge.net/doc/html/Refactoring.htm`

just the names, which are shown to the user, change.

4.4.2 Moving/copying features

After moving or copying member variables and functions with drag-and-drop or cut/copy-/paste operations to an other place in the declaration tree, everything is restructured and refactored automatically if possible. Otherwise the user has to repair these in a meaningful way.

4.4.3 Moving/copying other Lava entities

Other Lava entities are moved or copied with the same drag-and-drop or cut/copy/paste operations. The names of moved or copied entities, which reflect the nesting of packages and classes, are automatically changed. The user do not have to rename something manually.

4.4.4 Changing formal parameters

If new formal parameters were added to a function, this changes were done to all invocations of this function at the whole program. The same is done when formal parameters are deleted, replaced or the order of them changed.

4.4.5 Introducing or removing accessor ("setter/getter") functions

Introducing or removing accessor ("setter/getter") functions for member variables of a class is supported by Lava. Default setter/getter functions are generated automatically by a class if it needs them. Changes are refactored automatically.

4.5 Integrated database

Lava is based on (bounded) quantifier logic and provides "exists", "foreach" and logical conjunctions (and, or, not, xor, if-then-else) and a set construction expression "select", that roughly corresponds to the SQL "select" expression. These can be used to express object and set queries, independent to the underlying data. Databases are accessible through specific interfaces. Therefore Lava does not need "embedded SQL" or similar implementations.[9]

5 Examples

This section explains how to create a simple program with the Lava Programming Environment and shows an example program which generates the lyrics to the song "99 Bottles of Beer" as an output.

[9] `http://lavape.sourceforge.net/doc/html/DatabaseIntegration.htm`

5.1 Creating a program

To create a program with Lava, the Lava Programming Environment (LavaPE) is necessary, which is available for Windows (32 Bit), Linux and Mac OS. It can be freely downloaded.[10] The LavaPE comes with a several samples which could help the user to get more common with Lava.

To create a new Program, click on ”File” in the menu and select ”New Lava file” and save it afterwards with a click on the save button or press ”Ctrl+S”.

At first the declaration view (3.1) is visable and you should create a main program by clicking on the ”New main program...” button (The third button from the left at Figure 1) and give it a name. With a double click on the ”Exec” item in the main program, the exec view (3.2) opens and the executable part can be created/edited.

At the exec view, we can click the ”Call a virtual function” button, write ”Hello World!” in the ”<expr>” placeholder and select ”messageBox” at the ”<func>” placeholder and then we have a hello world program like shown in Listing 1.

```
main HelloWorld

call "Hello World!".messageBox
```

Listing 1: HelloWorld

In the declaration view you can create a new class, a implementation of a class, a function, parameters and much more. At the exec view you can edit the executable parts called ”Exec”.

5.2 ”99 Bottles of Beer” example

The following example shows a program which generates the lyrics to the song ”99 Bottles of Beer” as an output.[11] It is a recursive solution because Lava abandons sequential loops.[12] Figure 4 shows the program in the declaration view (3.1). The main part of the program (Listing 2) just starts the “Show99BottlesOfBeerLyrics” function (Listing 3) with the amount of bottles (99 if we want to show the entire song) and this function have to do the whole work.

In all strophes except the last three, the function builts the text of the strophe, which contains the amount of beers which are left and the amount of beers which are left after take one down, shows the strophe to the user and calls itselfe recursively with one beer less than before.

The third and second last strophes do basically the same but with a slightly different text.

At the last strophe, when there are no beers left, the text of the last strophe is shown, but there are no more function calls necessary.

[10] http://sourceforge.net/projects/lavape/files/
[11] http://www.99-bottles-of-beer.net/
[12] http://lavape.sourceforge.net/doc/html/RepetComputSamples.htm

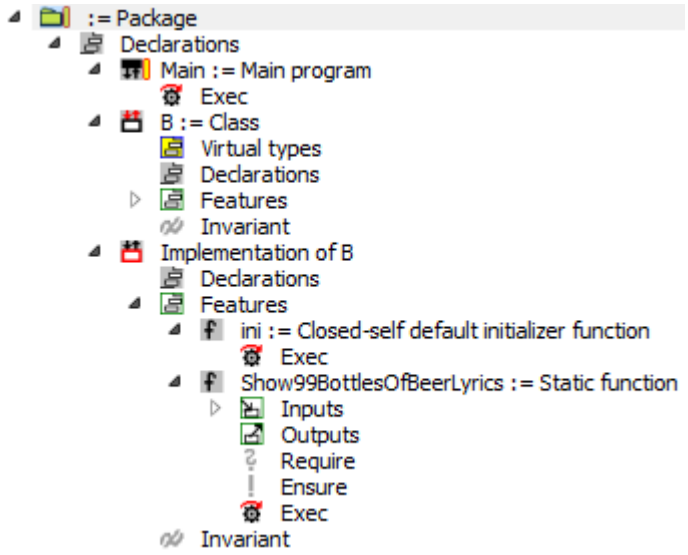

Figure 4: Tree view of the ”99 Bottles of Beer” program

The two lines of each strophe are produced seperatly because of the readability of the code and the output is down with a MessageBox, because this is the normal way in Lava.

```
main Main

call B::Show99BottlesOfBeerLyrics(99)
```

Listing 2: Main

```
function Show99BottlesOfBeerLyrics
inputs:
    Integer beers

declare
   String beersString, beersLeftString, line1, line2
do
   if
      beers > 2
   then
      call beers.string ==> parameter:beersString;
      call ( beers - 1 ).string ==> parameter:beersLeftString;
      set ( ( beersString
         + " bottles of beer on the wall, " )
         + beersString )
         + " bottles of beer.\n" ==> line1;
      set ( "Take one down and pass it around, "
```

```
            + beersLeftString )
            + " bottles of beer on the wall." ==> line2;
         call ( line1 + line2 ).messageBox;
         call B::Show99BottlesOfBeerLyrics(parameter:beers - 1)
      elsif
         beers = 2
      then
         set "2 bottles of beer on the wall,
            2 bottles of beer.\n"==> line1;
         set "Take one down and pass it around,
            1 bottle of beer on the wall." ==> line2;
         call ( line1 + line2 ).messageBox;
         call B::Show99BottlesOfBeerLyrics(parameter:1)
      elsif
         beers = 1
      then
         set "1 bottle of beer on the wall,
            1 bottle of beer.\n" ==> line1;
         set "Take one down and pass it around,
            no more bottles of beer on the wall." ==> line2;
         call ( line1 + line2 ).messageBox;
         call B::Show99BottlesOfBeerLyrics(parameter:0)
      else
         set "No more bottles of beer on the wall,
            no more bottles of beer.\n" ==> line1;
         set "Go to the store and buy some more,
            99 bottles of beer on the wall." ==> line2;
         call ( line1 + line2 ).messageBox
      #if
#declare
```

Listing 3: Show99BottlesOfBeerLyrics

6 Differences to other languages

There are some differences between Lava and other programming languages. Some of the differences are benefits and others are differences. Lava is not the only programming language which base on graphically input. There are others visual programming languages with similar concepts.

6.1 Differences to well known languages

Compared to well known programming languages like Java, C, Visual Basic and others, programs in Lava are built in a different way. The standard way to build a programm in

most programming languages is to write source code in a language specific syntax. In Lava, the programs are mainly build by clicking given buttons at the Lava programming environment. Details about the Lava Programming Environment are described at section 3. This way of creating programs should be more intuitive, easier to learn and faster than conventional programming. Programs which can be created with Lava can also be created with other programming languages, just the way of producing the program is different.

6.2 Differences to visual programming languages

Lava counts to the visual programming languages. Visual programming languages are programming languages which allows the user to create programs graphically (point,click,drag,drop...) instead of writing code. Most visual programming languages are concepted to teach programming, or to build simple programs without much knowledge about software developing.

Lava uses the visual programming because it is easy (specially for beginners) and also supports the creation of complex programs like in other non visual programming languages.

7 Current status

Version 0.9.4 is the latest version of the Lava Programming Environment, which was released in 2013. It is available for Windows (32 Bit), Linux and Mac OS. The early preview release was released in 2000 and most of the developement take place in the next few years. There were also a few papers released, which are linked on the Lava webiste[13], but non of them is newer than form 2003.

Dr. Klaus D. Günther and Irmtraut Günther are the only two developers of the programming language Lava and the programming environment LavaPE. They still have plans to extend Lava.[14]

Since the release of Lava, other programming languages and programming environments also had ideas to make software developement easier and faster. A good integrated developement environment (IDE) also shows the user syntax errors while programming. Autocompletion and quick fix functionalities also helps the user to reduce errors and work faster and refactoring features become more common.

8 Conclusion

Lava is just an experimental programming language. The main idea of Lava, to make software developement easier and faster is good, but that can not easily be done by replacing a text based editor with a structure based editor. A lot of clicking and selecting in practice takes more time than writing a little bit of code. On the one hand, everything

[13] `http://sourceforge.net/projects/lavape/files/Lava%20Papers/`
[14] `http://lavape.sourceforge.net/doc/html/FAQLavaFuture.htm`

should be simple and errors should be preventet automatically, on the other hand loops have to be build with recursive function calls, which enhance the risc of errors, specially for beginners. The error detection of the Lava Programming Environment is a good concept, but integrated developement environments can also show errors while programming time and even gives you suggestions to solve them. Lava is just a two persons project and the structure based editor is no benefit for experienced software developers, so I do not think that Lava would become widely used.

References

[1] K. Günther. Lava: Programmieren im lego-stil.

[2] K. Günther. Lego-style application programming, 2001.

[3] K. Günther. Lava: Bausteinbasiertes programmieren mit struktureditoren. *Objektspektrum*, (1):58–64, 2002.

[4] K. Günther and I. Günther. Lava - an experimental object-oriented rad language with virtual types and component integration support.