Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
Electronic Notes in Theoretical Computer Science 224 (2009) 3–14
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Animalipse - An Eclipse Plugin for AnimalScript

Guido Rößling[1]

CS Department
TU Darmstadt
Darmstadt, Germany

Peter Schroeder[2]

CS Department
TU Darmstadt
Darmstadt, Germany

Abstract

The algorithm animation language AnimalScript, while highly expressive and versatile, is not easy to edit with no editor support. We have developed an Eclipse plugin for editing AnimalScript that includes a text editor, outline, and code assist. We expect that this plugin will make the editing process much easier and faster. The paper presents both technical aspects of the development and the resulting plugin.

Keywords: AnimalScript, Animal, Eclipse, Plugin, Animalipse

1 Introduction

Animal [5] is a versatile system for creating, modifying and presenting algorithm animations and visualizations (AV content). As far as we know, it is the only AV system that allows users to create AV content using all of the following approaches:

- *visually* using drag and drop in a novice-friendly graphical user interface [8],
- *textually* using the highly expressive AnimalScript language [7,9],
- employing a new Java-based generation API,
- using a set of external applications for generating context-specific animations for trees [10] as well as for graphs and graph algorithms [3,11],

[1] Email: roessling@acm.org
[2] Email: sevenclev@gmx.net

doi:10.1016/j.entcs.2008.12.044

- as well as using one of the currently more than 200 animation generators of the built-in *generator* framework [6]. Note that the number of generators does not necessarily indicate the number of algorithms covered, but rather the different "flavors" for the given algorithms, such as the choice of the programming language and the output language used for the presentation.

All generation approaches except for the first are directly or indirectly based on using AnimalScript, which is in the process of taking over the role of the preferred representation of Animal AV content from the built-in ASCII notation. The reasons for this development are the human-readable notation of AnimalScript, the ease with which it can be generated from programs and edited manually, and the expressiveness of the language. Since 2008, Animal also includes an integrated display of the BNF-based definition of the AnimalScript notation, as well as (since 2006) a small text editor for directly entering or modifying AnimalScript input and visualizing the results.

AnimalScript files contain one command per line, such as a definition of a new graphical object or a transformation of some objects. The animation is organized in steps, each of which can contain one or more commands. If multiple commands are used in a step, the step is surrounded by curly braces { }. Please see [9,7] for more information about AnimalScript.

Many of the other established AV systems also cover some of the generation approaches listed above. For example, JAWAA2 [1] and the GAIGS and JSamba [12] visualization engines used by JHAVÉ [4] also use a scripting language. JAWAA2 also offers a visual editor in its current release. JHAVÉ offers a set of content generators that are similar to the approaches offered in Animal's generator framework and can be run off the web. However, they focus on specifying algorithm parameters, and thus do not allow the definition of visual properties such as colors.

While AnimalScript can be edited easily using Animal's built-in editor or any arbitrary text editor, the comfort offered by this is somewhat lacking. The internal editor only offers rudimentary editing features; *cut, copy* and *paste* features are only supported by using the underlying operating system support. The editor does not offer a search facility, display of line numbers, indication of recognized syntactical or semantical errors, or syntax highlighting. Thus, editing a longer AnimalScript file is awkward and can become frustrating if the system indicates a parsing problem "in line 117". Despite (usually) precise information about the nature of the error, the lack of line numbers, search or "go to line" functions makes locating and fixing the error a tedious and less than enjoyable process.

We decided that his unsatisfying state needed addressing. Essentially, we saw three different approaches to provide better user support: improve the built-in editor to be comparable in comfort to the user's preferred text editor, create a new custom editor for AnimalScript content, or provide AnimalScript bindings for at least one commonly used text editor. It did not seem useful to invest much effort only to improve the built-in editor so that it would be comparable to, but still different from, a given user's preferred text editor. The same applied to creating a new custom editor. Therefore, we opted to provide AnimalScript bindings for at least

one commonly used text editor. We now had to decide which text editor to use.

The main target audience for ANIMAL and thus for ANIMALSCRIPT are students and teachers of Computer Science. We decided to use the text editor provided by the Eclipse IDE, as this IDE is used in many Computer Science courses. Thus, our target users may already be familiar with the basic features of the text editor.

The remainder of the paper is structured as follows. In Section 2, we will briefly summarize the features offered by the Eclipse IDE, focusing on plugins and text editors. Section 3 outlines the plugin for editing ANIMALSCRIPT code using the Eclipse IDE features. Section 4 shows usage examples to illustrate the support for ANIMALSCRIPT provided by the ANIMALIPSE plugin. Finally, Section 5 evaluates the plugin and presents areas for further research.

2 A Brief Overview of the Eclipse IDE

Eclipse [2] was presented by IBM in 2001 and turned into open source in 2004. Due to a large number of developers, the platform offers a huge selection of plugins and extensions for different needs, including a large selection of supported programming languages, version control system front-ends for CVS and Subversion, workflow and design components. Probably the best known plugin is the *Java Development Tools*, employed by many students, researchers, developers, and teachers world-wide for writing Java-based programs.

The main components of the Eclipse platform are the *workbench* responsible for the graphical user interface, including the maintenance of the Eclipse windows, and the *workspace*. The workspace is a separate file system that handles the creation, storage and editing of files, including files, directories and projects.

The graphical front-end of Eclipse contains the usual menus and dialogs as well as *editors* and *views*. Editors are used to modify resources - the most well-known is the Java Editor for editing Java class source. Views are responsible for presenting content. Eclipse already offers many different views such as the *Problems, Progress* or *Console* views. Figure 1 shows an annotated screenshot of the main components of Eclipse, as the will be described in the following paragraphs.

The *Outline* view provides a table of contents-like view of the editor contents. For a Java class, this view lists all methods and import statements; clicking on a line directly positions the text editor on the associated content line. Longer components in the text editor may also be folded to reduce visual clutter and make it easier to focus on the current area of work.

Eclipse *plugins* are Java programs that are loaded by the Eclipse runtime environment and added to the platform. They are connected to Eclipse using "extension points" provided by Eclipse, which describe different aspects of the Eclipse platform that can be extended by a given plugin [2]. The definition of the extension points used is stated in a XML-based "plugin manifest" that has to be provided together with each plugin.

One of the strengths of the Eclipse editors is the integration of helpful features. This includes syntax highlighting (using either color or font changes, or a com-

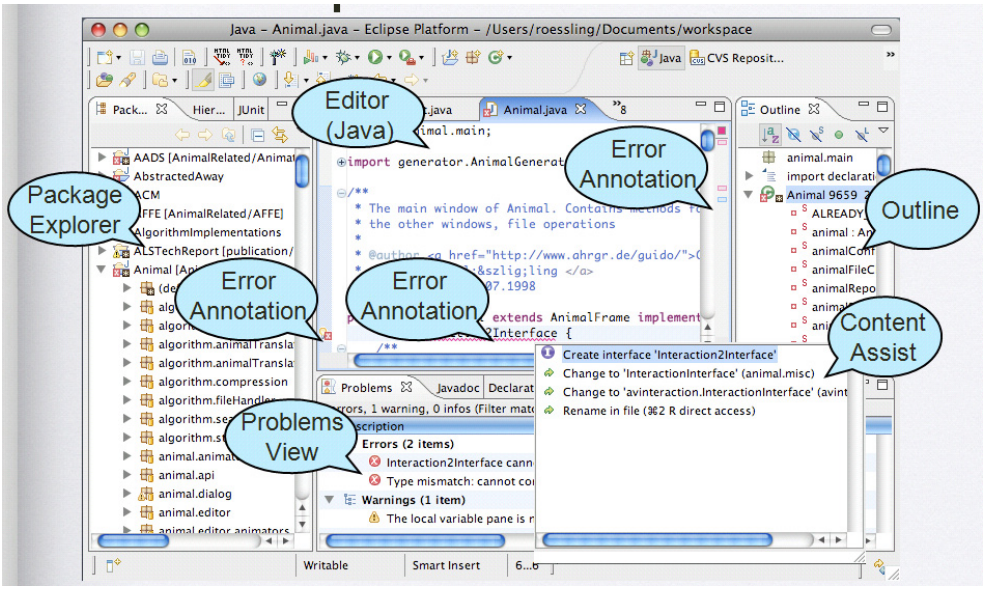

Fig. 1. An annotated screenshot of the main components of Eclipse

bination), the ability to directly jump to a given line in the editor, and marking (recognized) issues. The marking facilities called "annotations" in Eclipse can include a mark in the left or right margin of the text editor next to the affected line. An x-shaped cross in a red circle mark in the left margin is used to indicate syntactical (or other) errors, together with a red mark in the scroll bar to the right. Additionally, the annotated text is indicated, usually by a wavy red line, and an appropriate description is placed in the *Problems* view.

Finally, Eclipse editors can also support the user by *content assist*, often also called *code assist* or *code completion*. This feature lets the user choose from a pop-up list of constructs or completions possible at the current text caret position. It can be used both for the insertion of a single keyword or complete structure, such as an *if..else..* statement, and for the choice of a given method to be invoked, including the required invocation parameters. This type of support is only possible if the plugin is aware of the syntax of the underlying language or the set of classes and their methods, respectively. Both aspects of content assist can be very helpful and save time, especially for users who are new to the target language.

3 Animalipse: An Eclipse Plugin for AnimalScript

The Animalipse plugin was expected to provide the following functionality:

- support the creation and editing of AnimalScript as an Eclipse plugin;
- allow easy installation using the Eclipse plugin installation support;
- provide *cut, copy* and *paste* functionality, as well as *undo* and *redo*;
- allow the display of line numbers, animation step folding (showing only one line for a set of commands in the same animation step), automatic indentation of code

lines, and syntax highlighting;

- locate and mark errors in the AnimalScript file;
- display a useful overview in the Eclipse *Outline* view;
- and finally provide content assist for the AnimalScript command notation.

An AnimalScript file edited in the plugin shall also be directly runnable from the plugin, so that the user does not have to start the required Animal system externally.

The Animalipse plugin is based on an Eclipse text editor and thus directly inherits some of the requested functionality, such as the support for *cut, copy* and *paste*, as well as the easy installation typical for Eclipse plugins.

Figure 2 shows a screenshot of the plugin in action, annotated similarly to Figure 1. The individual elements of the plugin will be described in more detail in the following subsections. Additionally, subelements will be presented in larger screenshots in Section 4.

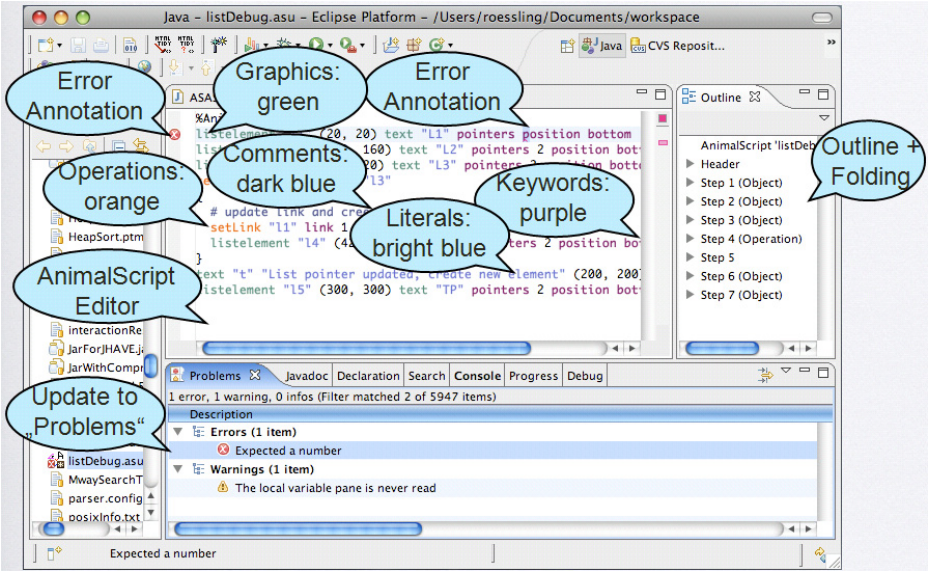

Fig. 2. Overview of the Animalipse Eclipse plugin with annotations

3.1 *Parsing* AnimalScript *Content*

Some of the features listed above, especially for marking errors, syntax highlighting and content assist, require that the editor “understands” what is being edited. As the editor may also need to request the same piece of information multiple times, we decided to implement a document object model (DOM) for AnimalScript (AS-DOM) to allow for faster and more expressive access to information about the currently selected element or current editing position. The essential structure of the AnimalScript-DOM consists of a root element, metadata about the AnimalScript contents, and a set of animation steps. The steps are placed in

ascending order, just as they would be for the animation. Each step can contain one or multiple animation commands.

The creation of a AnimalScript-DOM requires parsing the UTF-8 encoded AnimalScript contents from the text editor. By registering as an observer in the *IDocument* provided by the Eclipse editor classes, the parser can be informed automatically about changes in the code, and thus update its DOM. However, each key press triggers an update event, which would force the system to parse the complete script (again). Therefore, we decided to enforce a waiting interval of at least 2.5 seconds between two parsing iterations to prevent unnecessary continuous parsing of the editor contents. Of course, this interval can be adjusted by the end-user.

Animalipse does not directly use abstract syntax trees to support the parsing process, but parses elements on a line base. This is possible as AnimalScript mandates that each command will occupy exactly one line (and that each input line will contain exactly one command, if one ignores comments or the curly braces used to indicate steps). A command consists of a sequence of language elements separated by an arbitrary amount of whitespace. Each element can for example be numeric, a literal, or a keyword. The internal representation of the parsed elements is similar to an abstract syntax tree, called *ASAST* for AnimalScript-Abstract Syntax Tree. The definition of the tree is created when the plugin is started by parsing a BNF-like definition file, which makes a later adaptation of the AnimalScript language easy.

Listing 1 shows an excerpt of the BNF-like definition used for defining polygonal objects and especially squares. Non-terminal symbols are placed in <>. The colon following a non-terminal symbol separates the symbol from its definition. Thus, the *polygonTypes* non-terminal symbol can be refined to an instance of the *square, rect, triangle* or *polygon* non-terminal symbol. The `$idDescr` tag following the `''ID''` field provides meta-information for the content assist described in the next subsection.

According to the BNF, the definition of a square consists of the keyword *square*, followed by the square's ID in double quotes, a node definition and a natural number describing the top left corner of the square and its size. The other elements are placed in brackets to indicate that they are optional: the color of the square's outline, its depth, the fill options—whether the square is filled, and if so, with which color—, and the display options—whether the option is hidden or becomes visible only after a delay. Note that each optional element starts with a space separating the element from its predecessor. This space is also used for content assist, as described in the next subsection.

Listing 1: Example BNF for defining squares and rectangles for Animalipse

```
1 <polygonTypes>:
2 <square>|<rect>|<triangle>|<polygon>
3 <square>:
4 square "ID"$idDescr <nodeDefinition> <nat+>[ color <color>]
5   [ depth <nat>] [ <fillOptions>][ <displayOptions>]
```

```
6  <rect>:
7  <absoluteRectangle>|<relativeRectangle>
8  <absoluteRectangle>:
9  rectangle "ID"$idDescr <nodeDefinition> <nodeDefinition>
10   [ color <color>][ depth <nat>][ <fillOptions>]
11   [ <displayOptions>]
```

3.2 Content Assist

The content assist feature of ANIMALIPSE uses the content assist components provided by the Eclipse editors. Based on the internal representation of the script and the current position, the plugin creates a list of recommendations for content that could be used to complete the current selection. Compared to many other languages (including Java), this process is difficult for ANIMALSCRIPT content: the notation used by ANIMALSCRIPT contains many optional keywords or elements, making the number of possible completions at any given point comparatively high. After the list is populated, it is presented to the user, who is then prompted to choose one of the elements.

Similarly, locating syntax errors in a given ANIMALSCRIPT input file is also difficult due to the profusion of optional elements. In this case, the large number of branches possible at (almost) any given point in the parsing process leads to a number of "wrong errors": a command line is only syntactically incorrect if it does not match *any* of the possible syntactical rules. Or, to put it differently, if there is a way to parse a given line without a syntax error, the line is syntactically correct, and all parse errors when trying a different combination of optional elements have to be ignored. Additionally, the parser has to be able to detect when there is "too much" input in the current line: the command has been completely parsed, but contains additional elements that thus do not belong to the text.

Several nodes in the ANIMALSCRIPT abstract syntax tree can be annotated with context information, as shown in Listing 2. The context definition provides additional details about the context of a given leaf in the ANIMALSCRIPT abstract syntax tree. Line 1 shows the unique identifier of the regarded context element (*tupleOfTwoNaturalNumbers*), used for defining absolute coordinates. The dollar sign indicates that this is the element to be defined. Each definition line starts with the "at" character @ and can provide the following information:

@info elements provide a user-readable text that describes the element.

@display provides a user-readable rendition of the terminal output in the editor. This is for example necessary to indicate the position(s) of whitespace.

@cursorhint specifies how many character the cursor has to be shifted to the left after inserting the definition shown in *@display* into the editor. In this example, choosing the element will lead to the inclusion of the text `( X , Y )` including all spaces. The value 7 for the cursorhint places the cursor seven positions to the left of its position after the insertion, and thus places it on the first coordinate

inside the parentheses.

Listing 2: Context Definition Example

```
$tupleOfTwoNaturalNumbers
@info=A tuple of two natural numbers
@display=( X , Y )
@cursorhint=7
```

The combination of the presented features made the implementation of the parsing and content assist components of the AnimalipSE plugin far more difficult than we originally anticipated. However, we managed to overcome all obstacles and now have a full-fledged Eclipse plugin for AnimalScript.

3.3 Integration into Eclipse

The integration of the AnimalipSE plugin into Eclipse uses five different Eclipse extension points, as outlined in Table 1.

Plugin component	**Eclipse Extension Point**
AnimalScript editor	*org.eclipse.ui.editors*
Error marking	*org.eclipse.core.resources.markers*
Starting the animation	*org.eclipse.debug.ui.launchShortcuts*
New Animation Creation Wizard	*org.eclipse.ui.newWizards*
Plugin Preferences	*org.eclipse.ui.preferencesPage*

Table 1
Eclipse Extension Points used for the AnimalipSE plugin

The AnimalScript editor is based on the *org.eclipse.ui.IEditorPart* interface and extends the Eclipse *TextEditor* class. It handles AnimalScript files with the extension *.asu*. This editor is automatically started whenever the user opens or creates a file with this extension.

Other components of the plugin, such as code folding or the creation of the overview, similarly implement provided interfaces and extend existing Eclipse classes.

3.4 Installing the Plugin

To install the plugin, the user selects the *Help → Software Updates → Find and Install...* menu entry. After selecting "new features to install", a new remote site has to be created with the address `http://www.algoanim.info/Animal/download/Animalipse/` and confirmed by OK. After finishing the settings, a list of updates should appear and include AnimalipSE. After confirming all subsequent dialogs, the plugin will be installed and can be used after a restart of Eclipse.

4 Example Features of the Animalipse Eclipse Plugin

New AnimalScript files can be generated inside Animalipse by using the built-in creation wizard. The user first uses the *File* menu, toolbar or menu entry to create a new file of type AnimalScript. In the following dialog, the title of the animation, its width and height and the animation author and title can be specified. This information is then used to create the appropriate AnimalScript header and open the resulting file in the AnimalScript editor.

```
%Animal 2
listelement "l1" (20, 20) text "L1" pointers 2 position bottom color black ptr1 (
echo boundingBox: "l1"
echo bounds: "l1"
listelement "l2" (160, 160) text "L2" pointers 2 position bottom ptr2 (120, 120)
echo bounds: "l2"
listelement "l3" (300, 20) text "L3" pointers 2 position bottom ptr2 to "l2"
echo bounds: "l3"
setLink "l2" link 1 to "l3"
{
  # update link and creeate new object
  echo bounds: "l2"
  setLink "l1" link 1 to "l2"
  listelement "l4" (420, 40) text "L4" pointers 2 position bottom prev "l3"
}
text "t" "a" "Demo text of many characters" (200, 200)
listelement "l5" (300, 300) text "TP" pointers 2 position bottom prev "t"
```

Fig. 3. The Animalipse text editor with syntax highlighting

Figure 3 shows an example of the Animalipse text editor. Keywords used to define graphical objects are highlighted in green, operation keywords are shown in orange. All other keywords are shown in purple. Literal values, such as object names and Strings, are shown in blue. Color definitions are placed before a background of the chosen color. The original text is colored in a complementary color to be readable. Font definitions (not included in the example) are placed in italics. Finally, comments - introduced by the hash mark **#** - are shown in dark green. All color settings can be adjusted in the Animalipse plugin preferences.

```
%Animal 2
square "s" ( 10 , 30 ) 20 color black filled fillColor blue
square "s2" offset ( 10 , 20 ) from "s" W 30 color blue fillColor yellow
```

Fig. 4. Indication of a syntax error by the Animalipse plugin

Figure 4 shows a view of a small code snippet with a syntax error—the keyword "filled" of the optional *fillOptions* mentioned in Listing 1 is missing, as can be seen by comparing the two code lines. The syntax analysis has to be triggered manually by selecting the *Search for errors...* entry in the context menu of the editor. The incorrectly placed keyword *fillColor* is shown with a wavy red underline. Additionally, the marker in the left margin and the line marker in the scrollbar to the right indicate the error, while the filled red square at the top right shows the presence of (at least) one error. Additionally, but not shown in Figure 4, a short error description appears in the Eclipse *Problems* view.

The outline of the rather simple animation shown in Figure 3 is shown in Figure 5. Each step can be folded or unfolded to show more details. If multiple operations take place in the same animation step, they are shown on separate lines. Clicking on an entry positions the cursor on the appropriate line in the text editor.

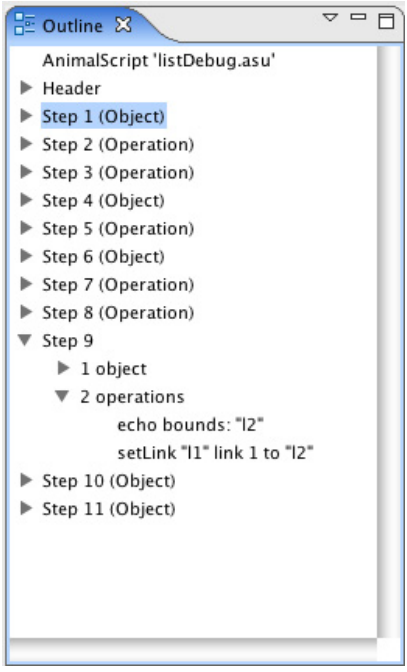

Fig. 5. The ANIMALIPSE outline, showing the structure of the animation

Content assist is provided whenever the user requests it explicitly by pressing the CTRL key together with the space key. Additionally, ANIMALIPSE offers content assist if a space character is entered. For this end, the space characters used in the BNF shown in Listing 1 are important, as the leading space is recognized as the possible start for one of the components. A small window pops up and offers the list of legal completions at this point, allowing the user to choose one or close the window and continue manually.

ANIMALSCRIPT content can be run directly using the context menu entry *Run → Load in* ANIMAL. This requires that the user has first told the plugin where the ANIMAL jar file can be found by going to *Window → Preferences...*, selecting the plugin from the list, and entering or browsing for the location of the ANIMAL jar.

5 Summary and Future Work

In this paper, we have presented the ANIMALIPSE Eclipse plugin for creating, editing and debugging ANIMALSCRIPT-based AV content. The plugin is very easy to integrate into an existing Eclipse installation and behaves similarly to other established plugins, such as the Java Development Tools plugin for Eclipse. Users should therefore find it easy to use the plugin to become more proficient with ANIMALSCRIPT.

The abstract syntax tree model used for the AnimalScript language allows extending the language's definition by modifying the BNF-based notation without touching the plugin code. However, a certain measure of caution has to be exerted when editing the file, to prevent the introduction of parsing errors. At the same time, the underlying notation could be exchanged by another language, such as JAWAA2 [1], by editing the notation file accordingly, or importing a different BNF file for the other chosen notation. The implementation of the underlying abstract syntax tree uses a set of elements that should be helpful for many other languages, such as support for integers and natural numbers, composite steps, keywords, literals and identifiers. If the non-terminal symbols used for the these elements are also used for the BNF definition of another language, it should be directly possible to support the other language inside the plugin. Please note that we could not test this during the limited amount of time the second author could spend on his Bachelor Thesis.

The other features described in this paper, such as syntax highlighting, searching for errors, and content assist, should also prove helpful. There are some minor issues with some of these components, which are due to the underlying language notation. For example, the choice lists for code assist can become very long, if the user requests assistance near the start of a command. This is due to the large number of optional keywords and components used throughout AnimalScript. While this makes programming in AnimalScript comfortable ("specify what you need and skip the rest"), it also leads to a large number of possible correct completions.

The error detection is not fully accurate; found errors will lead to parsing errors in Animal, but not all errors during loading in the animation in Animal may be detected by the plugin. For example, the user may request a certain transformation subtype, such as moving the nodes 4 and 5, on a structure that does not support this operation, for example a square. Syntactically, this command is correct, but it will lead to a semantic error when the execution is attempted. Therefore, the plugin cannot detect this type of error unless it were more tightly interwoven with Animal's internal parsing process - which would slow down the processing of files.

Possible extensions include highlighting the use of the currently selected identifier or showing context information about elements as a tool tip. Automatic formating of AnimalScript files, similarly to the feature offered by the Java editor, would also be helpful. Finally, it would be interesting to remodel Animal's content generators as Eclipse wizards or create an internal Animal view. However, these aspects require another full-time student working on them as a Bachelor Thesis.

If you are interested in trying out Animalipse, please follow the steps described in Section 3.4. Constructive feedback is appreciated! We would also like to cooperate with others who wish to implement Eclipse plugins for "their" AV notation.

References

[1] Akingbade, A., T. Finley, D. Jackson, P. Patel and S. H. Rodger, *JAWAA: Easy Web-Based Animation from CS 0 to Advanced CS Courses*, in: *Proceedings of the* 34^{th} *ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2003), Reno, Nevada* (2003), pp. 162–166.

[2] Beck, K. and E. Gamma, "Contributing to Eclipse. Principles, Patterns, and Plugins," Addison-Wesley

Longman, 2003.

[3] Naps, T. L., J. Lucas and G. Rößling, *VisualGraph - A Graph Class Designed for Both Undergraduate Students and Educators*, in: *Proceedings of the* 34^{th} *ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2003), Reno, Nevada* (2003), pp. 167–171.

[4] Naps, T. L. and G. Rößling, *JHAVÉ - more Visualizers (and Visualizations) Needed*, in: G. Rößling, editor, *Proceedings of the Fourth Program Visualization Workshop, Florence, Italy*, 2006, pp. 112–117.

[5] Rößling, G., "ANIMAL-FARM: An Extensible Framework for Algorithm Visualization," VDM Verlag Dr. Müller, 2008.

[6] Rößling, G. and T. Ackermann, *A Framework for Generating AV Content on-the-fly*, in: G. Rößling, editor, *Proceedings of the Fourth Program Visualization Workshop, Florence, Italy*, 2006, pp. 106–111.

[7] Rößling, G. and B. Freisleben, ANIMALSCRIPT*: An Extensible Scripting Language for Algorithm Animation*, Proceedings of the 32^{nd} ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2001), Charlotte, North Carolina (2001), pp. 70–74.

[8] Rößling, G. and B. Freisleben, ANIMAL*: A System for Supporting Multiple Roles in Algorithm Animation*, Journal of Visual Languages and Computing **13** (2002), pp. 341–354.

[9] Rößling, G., F. Gliesche, T. Jajeh and T. Widjaja, *Enhanced Expressiveness in Scripting Using* ANIMALSCRIPT V2, in: *Proceedings of the Third Program Visualization Workshop, University of Warwick, UK*, 2004, pp. 15–19.

[10] Rößling, G. and S. Schneider, *An Integrated and "Engaging" Package for Tree Animations*, in: G. Rößling, editor, *Proceedings of the Fourth Program Visualization Workshop, Florence, Italy*, 2006, pp. 23–28.

[11] Rößling, G., S. Schneider and S. Kulessa, *Easy, Fast, and Flexible Algorithm Animation Generation*, in: *Proceedings of the* 13^{th} *ACM SIGCSE/SIGCUE International Conference on Innovation and Technology in Computer Science Education (ITiCSE 2007), Dundee, Scotland* (2007), p. 357.

[12] Stasko, J., *Building Software Visualizations through Direct Manipulation and Demonstration*, in: J. Stasko, J. Domingue, M. H. Brown and B. A. Price, editors, *Software Visualization*, MIT Press, 1998 pp. 187–203.