Yet Another Vaucanson GUI

Florent D'Halluin

Technical Report n^o0822, May 2008
revision 1871

VAUCANSON is a finite state machine manipulation platform. Since it was started in 2002, the project has been attracting more and more users. In that regard, it requires an efficient user front end.

For non-expert users, automaton manipulation can be done via TAF-KIT, a set of command-line tools. A first GUI was developed in 2005. Its use was slow and complicated since it relied on TAF-KIT for every operation.

This new GUI, plugged directly into the core of the VAUCANSON library for efficiency, simplifies the automaton manipulation process and makes full use of the generic algorithms included in the library.

VAUCANSON est une plateforme de manipulation d'automates finis. Débuté en 2002, le projet attire de plus en plus d'utilisateurs. De ce fait, une interface utilisateur efficace est nécessaire.

Pour l'utilisateur non expert, la manipulation d'automates peut s'effectuer via TAF-KIT, une suite d'outils accessible en ligne de commande. Une première interface graphique a été esquissée en 2005, mais son fonctionnement est lent et compliqué car elle s'appuie sur TAF-KIT pour réaliser chaque opération.

Cette nouvelle interface graphique, branchée directement sur le cœur de la bibliothèque pour plus d'efficacité, simplifie la manipulation d'automates et rend accessible les algorithmes génériques de VAUCANSON.

Keywords
Vaucanson, Vaucanson GUI, User Interface, Qt, Design principles

Laboratoire de Recherche et Développement de l'Epita
14-16, rue Voltaire – F-94276 Le Kremlin-Bicêtre cedex – France
Tél. +33 1 53 14 59 47 – Fax. +33 1 53 14 59 22
`florent.d'halluin@lrde.epita.fr` – http://publis.lrde.epita.fr/200807-Seminar-DHalluin

Copying this document

Contents

Chapter 1

Introduction

YAVGUI

"Providing a generic tool for automaton manipulation to non-expert users."

1.1 Automaton manipulation libraries

Finite state machines are widely used in theoretical fields, but there exists few software automaton manipulation libraries. In 2006, the two best known librairies were the proprietary library FSM (Mohri et al., 2008) and the open-source library VAUCANSON (VAUCANSON Group, 2008a).

Both were aimed to handle multiple automaton types, and were refered to as *generic* manipulation libraries. However, FSM had much better performances than VAUCANSON, despite the recent improvements made to VAUCANSON (Lazzara, 2006).

Today, the actors have changed. FSM development was discontinued, and the library OPENFST (Riley et al., 2008) appeared. OPENFST is designed to work with weighted finite-state transducers and does not handle other automaton types.

VAUCANSON evolved as well. Once a generic manipulation *library*, it now aims to be a manipulation *platform*, complete with a package of command-line tools and a graphical user interface.

This evolution is an ongoing process. This report aims to show the motivations behind the new Vaucanson GUI, YAVGUI. It also presents design considerations and implementation results. Note that YAVGUI is a temporary name and that the tool presented here may be published under another name. Name suggestions can be sent to the VAUCANSON GROUP.

The reader is expected to be familiar with the C++ language and to have a basic knowledge of automata theory.

1.2 Acknowledgments

Jacques Sakarovitch and Sylvain Lombardy for their thoughts on automaton manipulation and their invaluable help with automata theory.
Alexandre Duret-Lutz for his practical help and pointers.
The members of the VAUCANSON GROUP for their constant feedback.

Chapter 2

Context & motivations

2.1 The VAUCANSON platform

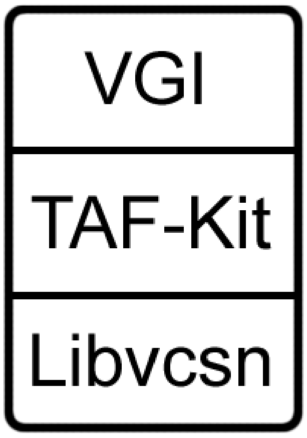

Figure 2.1: Structure of the VAUCANSON platform.

The VAUCANSON project is a multi-layered manipulation platform for finite state automata created in 2002. It has three sets of layers: the core C++ library, refered to as LIBVCSN, a binary package called TAF-KIT, a graphical interface, VGI, soon to be replaced by YAVGUI.

LIBVCSN

This C++ library contains two layers. The lower layer defines core structures for monoids, series and rational expressions, and the upper layer provides algorithms on those structures. LIBVCSN implements the Element design pattern through generic templated code (Cadilhac, 2006). It can therefore handle many types of automata.

Developpers use LIBVCSN to integrate automaton manipulation to a piece of software. Dynamic library files are instantiated for each automaton type used in the software. The libraries include instances of structures and algorithms for their given automaton type.

TAF-Kit

TAF-Kit is a set of command-line tools operating on the instantiations of the libvcsn for some of the most common automaton types, including boolean transducers. Before Yavgui, it was the most practical tool for non-expert Vaucanson users to create and edit automata, and to access the predefined algorithms included in libvcsn.

VGI

VGI is a graphical user interface written in Java by Louis-Noël Pouchet in 2005. It manipulates automata through TAF-Kit, provides visual feedback and offers algorithms for automatic placement of states and transitions on a 2-dimentional plane.

The development of VGI was discontinued in 2006. It remains accessible to the development team on `vcsn.enst.fr`. Yavgui will replace VGI in future releases of the Vaucanson platform.

2.2 The need for an efficient user interface

There are inherent issues to the classical structure of the Vaucanson platform. The following figure illustrate the platform's workflow:

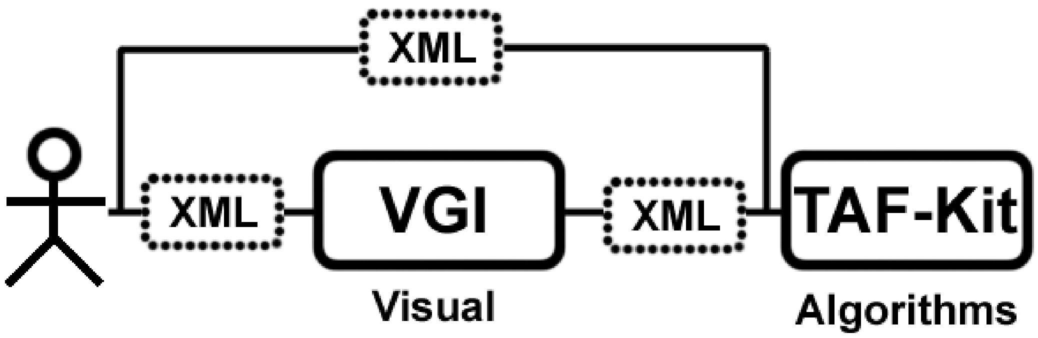

Figure 2.2: Structure of the Vaucanson platform.

Users have XML files describing automata. These files can be fed directly to TAF-Kit or go through VGI first. In VGI, automata are drawn on screen and at each algorithm execution, a new XML file is written and sent to TAF-Kit, which gives back a new XML file that goes back into VGI so the results can be displayed. The inherent problem to that is that 90% of the time spent in this chain of programs is spent generating and parsing XML files. Also, VGI's extent is limited to the automaton types that TAF-Kit can handle: VGI does not use the generic libvcsn library to its full potential.

The Vaucanson platform is also used by second year Epita students to illustrate the automata theory. They are not used to command-line tools and using TAF-Kit alone is a complicated and confusing process to some of them. The Vaucanson platform thus needs an efficient visual tool that meets the following points:

- TAF-Kit is not used as a bridge to libvcsn: the extra XML step is removed.

- LIBVCSN's genericity is exploited to its full potential.
- The resulting tool is simple, intuitive and visual.

2.3 Collaboration with the National Taiwan University

The VAUCANSON GROUP has ties with the National Taiwan University, Taipei. Professor Hsu-Chun Yen and his team offered to write a GUI for VAUCANSON that would answer the needs of the VAUCANSON GROUP. However, VAUCANSON is a large and complicated piece of code with no detailed documentation. YAVGUI is written in an effort to lay down reusable code for interacting with LIBVCSN and acts as a proof of concept in view of the development of a more complete GUI at the National Taiwan University.

2.4 Where YAVGUI fits in

The VAUCANSON GROUP took a joint decision so start a new GUI from scratch. It would be written in C++ and would not depend on TAF-KIT. It would be linked directly to LIBVCSN and would bring forward the genericity of LIBVCSN while retaining high performances. It would serve an educational purpose in such fields as the theory of languages, and provide reusable code to the National Taiwan University team, should they need any. In the VAUCANSON platform, YAVGUI is an alternative to VGI and TAF-KIT, as the following figure illustrates:

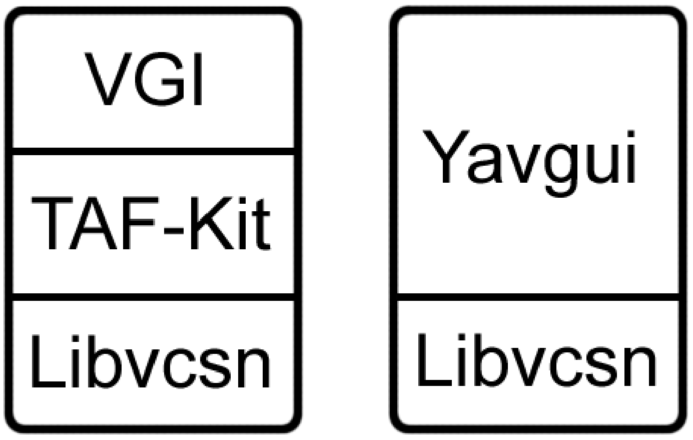

Figure 2.3: Structure of the VAUCANSON platform with YAVGUI.

Chapter 3

Design

This chapter concentrates on YAVGUI's inner workings as they were designed on paper. Implementation details and technical issues are explained in Chapter 4.

YAVGUI has two layers, seen in figure 3.1:

- The *front end* provides a graphical view of the automata and handles user interactions.
- The *core engine* is linked with LIBVCSN and allow access to its structures and algorithms

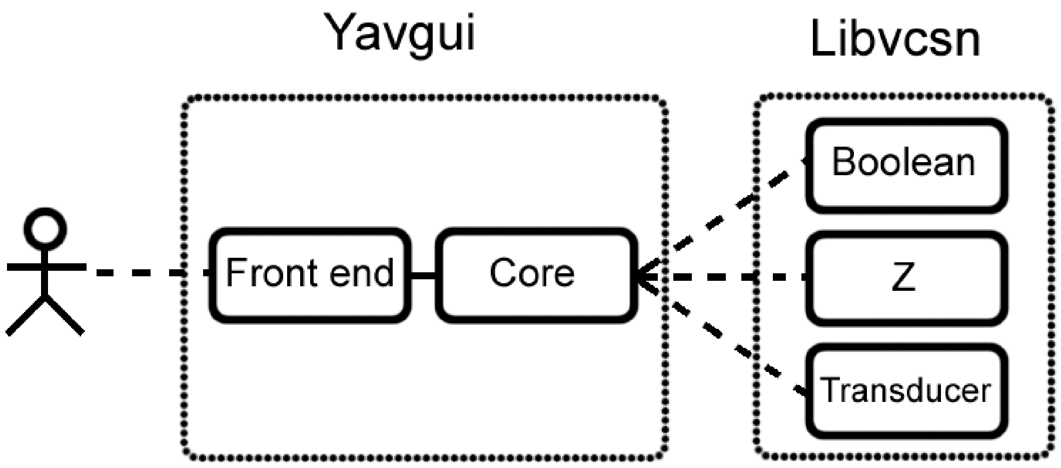

Figure 3.1: YAVGUI's Structure.

The **Visual front-end** section presents elements to take into account when designing a visual tool and describes the actual design of YAVGUI's visual component.
The **Core engine** section is an approach to the techniques used to manipulate generic libraries and to the solution chosen in YAVGUI.

3.1 Visual front-end

3.1.1 Elements of good GUI Design

A GUI for an automaton manipulation platform displays a visual repreentation of an automaton that includes states, state identifiers, transitions and transition labels. It allows the user to define and edit an automaton's structure, and to access the algorithms defined in the platform.

Careful consideration was given to YAVGUI's conception. Two questions arose:

- What makes a graphical interface powerful and attractive?
- How should the tool of choice for automaton manipulation be designed?

Good design is no longer an obscure science. Resources on the subject are readily available, such as *The Elements of Good Design* (Standard, 2008) and *First Principles of Interaction Design* (Tognazzini, 2008).

According to Ken Standard, the following two elements are of prime importance: **Consider the User** and **Start out Simple**. In the following, these principles are detailed with examples showing how they relate to automaton manipulation.

Consider the User

Users are the key to the expansion of a project. They are attracted by simplicity and ergonomy. When an inventive concept becomes a fully operational tool, developers must thrive to fit the tool to its users.

VAUCANSON aims to be a generic automaton manipulation platform suited both to developers and to C++ novices. While TAF-KIT does provide most of the tools needed to manipulate automata, it does not give the user immediate visual feedback and thus may seem *"too hard to use"* to a novice user.

A good GUI design gives the user an impression of simplicity and ease of use.

Start out Simple

Simplicity is expressed visually. A crowded visual field slows users down as they concentrate to avoid mistakes. Keeping the view clear and the features few and well-defined makes a user feel safe and increases his efficiency.

In GUI design, ergonomy is achieved through clever use of space. When using pointing devices, Fitts's law is a reliable guideline.

Fitts's law

Fitts's law is a model of human movement based on the following ergonomics principle:

> *"The time to acquire a target is a function of the distance to and size of the target."*(Tognazzini, 2008)

Simply put, large buttons can be reached faster than small ones, and long distances take longer to cover than short ones. The exception to this last rule is that screen corners and edges can be reached easily since the mouse pointer is pinned at the edge of the screen.

Fitts's law is straightforward and has many implications: In YAVGUI, the ability to zoom using the mouse wheel leads to tremendous gains of time compared to VGI since the user can make any target states larger with minimal mouse mouvement.

However, good layout design can be overshadowed by poor response times. The remainder of this chapter tackles the inner design of YAVGUI. Chapter 4 shows how the responsiveness is improved compared to VGI.

3.1.2 Actual visual design

The front-end is build around Trolltech's application framework Qt (Trolltech, 2008). Qt is multi-platform, flexible and well-documented, and was hence chosen over Fox or FLTK.

Qt allows object instances to be connected using *slots* and *signals*. *Signals* can be emitted from anywhere and are linked to *slots*, which behave like methods.

This process requires an additional pre-processing step from the Qt engine. Vaucanson uses GNU Autotools instead of Qt's *qmake*, which makes the integration tricky. It was simplified by the use of AutoTroll (Sigoure, 2008).

A visual preview is included below:

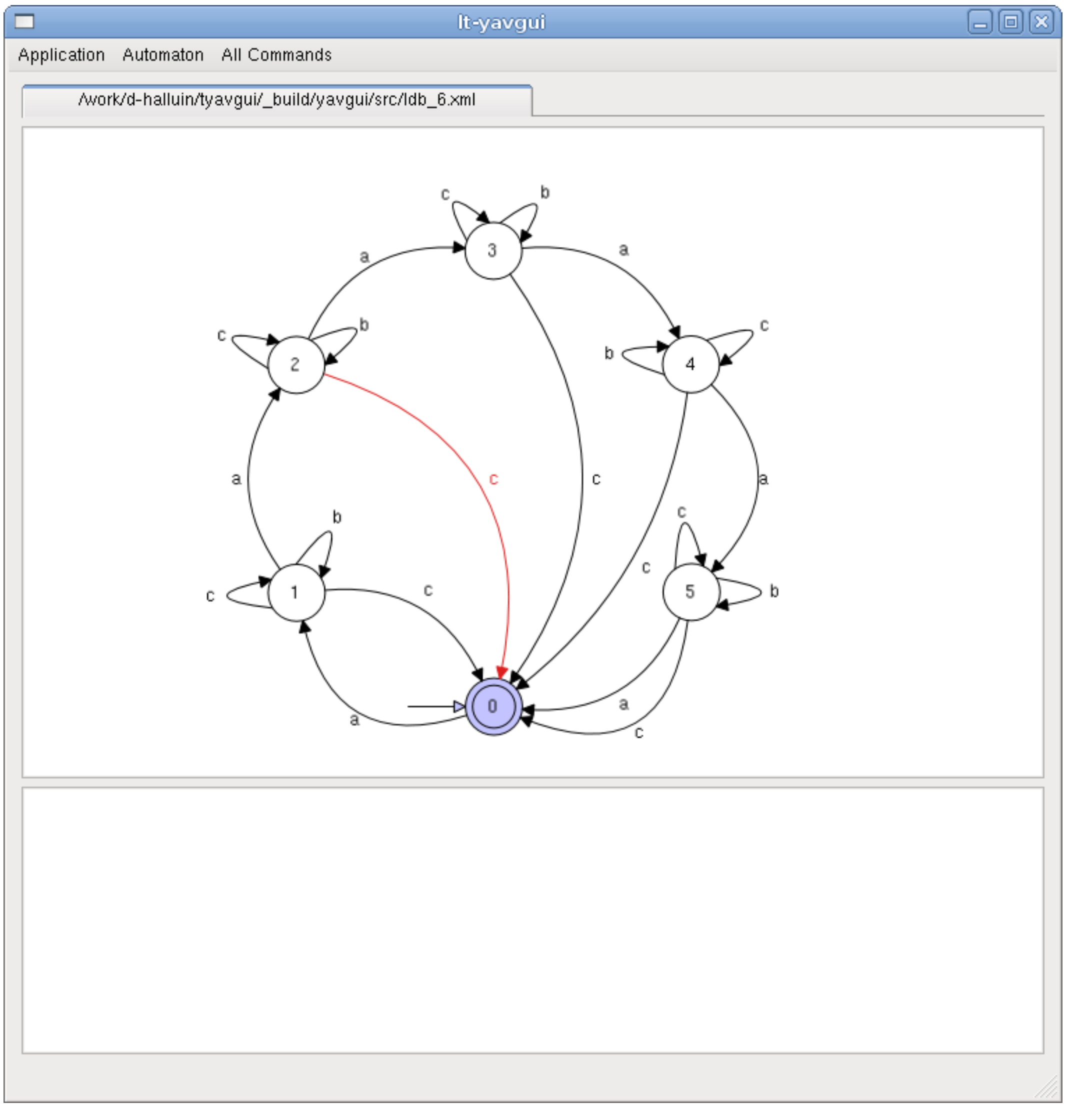

Figure 3.2: Overview of YAVGUI's visual. The mouse pointer, though not shown, was hovering over the transition highlighted in red.

- Automaton that are currently open appear on the top tab bar.
- Commands on the active automaton are accessed via the *Automaton* menu.
- States and transitions are highlighted as the mouse hovers over them.
- Highlighted states and transitions can be moved by dragging.
- Command results appear in the lower text area.

3.2 Core engine

YAVGUI handles many automaton types. For each automaton type, structures and algorithms are instantiated from LIBVCSN into the core engine.

This section describes the purpose and aims of the core engine and two design key points: The link between the core engine and LIBVCSN (*working with a generic library*) and the separation between the core engine and the front end (*static/dynamic bridge*).

3.2.1 Purpose and aims

The core engine provides an access to the inner types of LIBVCSN and the algorithms working on those types. It is meant to be separated from the front end so that it can be reused with minimal changes in other tools. This kind of separation has been approached in many ways in other LRDE projects, notably through the principle of static/dynamic bridge (Pouillard and Thivolle, 2006). While the current version of YAVGUI does not focus on this separation, such considerations will arise in the near future. The principle behind a static/dynamic bridge is explained in the last subsection of this chapter.

The main focus of the core engine is to manipulate types from a generic library while keeping the code as simple as possible. Two approaches are considered in the next subsection: a classical approach used in many LRDE projects, external polymorphism, as well as the practical approach used in YAVGUI that involves BOOST variants.

3.2.2 Working with a generic library

The Core engine manipulates many of LIBVCSN's types, *but has a unique, non-templated interface*. The classical way to implement this is to use external polymorphism, described next. In YAVGUI, a solution that seemed more immediately practical was used instead and is decribed afterwards.

The classical way: external polymorphism

An abstract class defines the interface through which generic classes will be manipulated. Each of the generic classes is wrapped in a concrete class inheriting from the interface class. Those concrete classes implement the methods of the interface according to the specificities of the generic class they hold. They each act as a proxy for the underlying generic classes.

This approach has several drawbacks in VAUCANSON's context:

- Having a new wrapping class for each of the types in LIBVCSN seems redundant and cumbersome.
- Due to the large number of generic algorithms available in LIBVCSN, the code of the concrete classes would be crowded by identical method definitions.

Actual solution: using BOOST variants

BOOST provides a `boost::variant` class that acts as a union container over complex data types such as classes. An instance of a `boost::variant` can thus hold any of the generic automaton class provided by LIBVCSN.

Access to the content of a `boost::variant` is done through a `boost::static_visitor`, which behaves like a functor. It possesses an overloaded `operator()` for all the types the `boost::variant` can hold. Commands (algorithms) on automata are each wrapped in a `boost::static_visitor` that can be applied to the variant holding the automaton instance.

The core engine has hence only one wrapping class that holds a variant over all of LIBVCSN's automaton types. Algorithms and automaton methods are each accessed via a visitor.

Since LIBVCSN is generic, those visitors have similar code for each of their `operator()` methods, which can thus be templated. This result in simple and easy-to-read code that can be simplified further by the use of macros. In the end, a command definition looks like this:

```
COMMAND_BASE(IsEmpty,
             FLAG_NAME("is_empty");
             FLAG_TOOLTIP("Test␣whether␣the␣automaton␣is␣empty.");
             FLAG_ALL;
             ,
             CODE_ALL(
               GET_CURRENT_INSTANCE(i1);
               Automaton& aut = AUTOMATON(i1, Automaton);
               std::string result =
                 vcsn::is_empty (aut) ? "true" : "false";
               SET_STATUS(i1, "Is␣empty:␣" << result);
               )
             );
```

The following figure illustrates the execution of a command in YAVGUI:

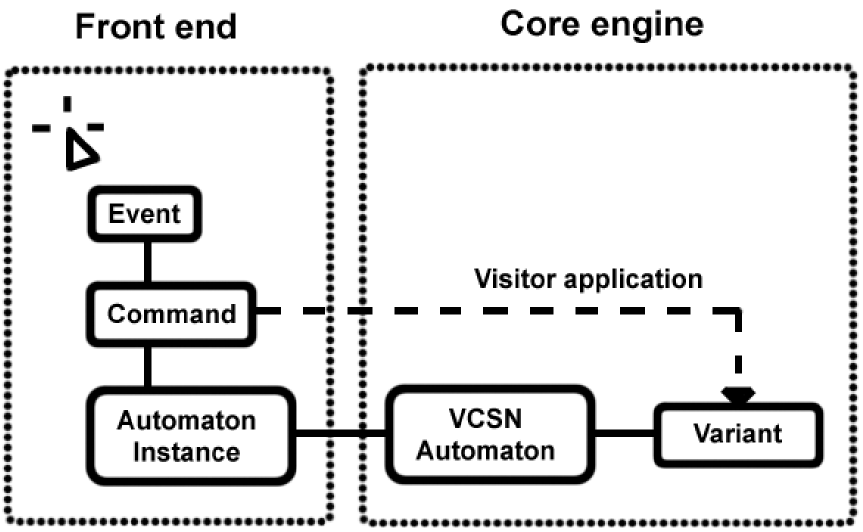

Figure 3.3: Execution of a command: An event is fired by the front end (mouse click), then linked to the corresponding command. The command takes hold of the currently active automaton instance, dives into the core engine wrapping class and applies itself on the variant.

3.2.3 Separating the core engine from the front-end: a static/dynamic bridge

The core engine encapsulate LIBVCSN in a static component that allows generic access to specific LIBVCSN classes, methods and functions. It also includes utilities that are absent from usual instanciations of LIBVCSN in dynamic libraries, such as the timer system described in appendix B. The principle of static/dynamic bridge refers to the process of making this component available from a dynamic language, such as `Ruby` or `Python`. This concern will arise as real-time type instanciation is considered (see Chapter 4, Future works). In YAVGUI's current version, the interface of this component is not precisely defined, and the front end and core engine remain strongly linked.

Chapter 4

Implementation

4.1 Technical issues

Writing a GUI from scratch did not come without its lot of tough spots.

There were two main complications:

- Due to the complexity of VAUCANSON core library, wrapping around it is a tricky process
- Since the inner structure of an automaton is traversed in order to construct its graphical representation, the front end has some degree of dependance on the core engine

4.1.1 Wrapping around VAUCANSON

Not all algorithms are instantiated in LIBVCSN dynamic library files. Some template instantiation must therefore be made while compiling the core engine, thus increasing compilation time.

Default dynamic library files are purposedly stripped from some algorithms defined in the C++ library.

A workaround would be to add the option to compile library files with full instantiation of all the symbols defined in the C++ library.

4.1.2 Dependance between Front end and Core engine

In order to allow runtime linking, the front end must be unaware of the link between the core engine and LIBVCSN. The calls to the core engine must be few and precisely identified, and must have a fixed prototype regardless of the automaton types instantiated in the core engine.

Due to time constraints, the interface of the core engine is not well-defined, and while the dissociation between the front end and the core engine exists on paper, the current implementation has introduced unwanted dependencies.

Since this does not impede the tool's operation, it will be tackled when runtime linking is considered.

4.2 Implementation keypoints

4.2.1 Commands

Commands are defined in the core engine as associations to generic LIBVCSN algorithms. Each command is a class that inheritates from `boost::static_visitor`.

That way, commands can be adapted to suit the specificities of each automaton type, while keeping the process completely transparent to the user.

The downside of this system is that it can make command definition fairly long and complicated even for the most basic algorithms. A set of macros facilitates this process and allows the commands to be tagged according to the automaton type they operate on.

This tagging process lets the front end sort commands automatically according to the automaton type they operate on. In the first version of YAVGUI, a menu item keeps a list of all the commands available on the currently active automaton updated at all times.

4.2.2 Graphical manipulation

- States and transitions can be moved around through the front end. According to the user's choice, these changes may or may not have an influence on the automaton's geometric structure as defined in LIBVCSN.
- Simple heuristic methods can automatically rearrange states and transitions.
- The view can be zoomed in and out using the mouse wheel. The current position of the mouse pointer is used as an anchor point.

4.2.3 Comparisons with the former interface

YAVGUI's interface is visually simpler than VGI's.

It offers more visual hints than VGI: States and transitions are highlighted when they can be modified, and the available commands have faster access time.

As expected, structure modification is significantly faster and can be done in real-time without noticable slow down with up to 20 states on screen.

In its preliminary version, YAVGUI has less control on automata than TAF-KIT or VGI, but the missing commands and structure operations will be added shortly.

Since the only available version of VGI is located on a remote machine (`vcsn.enst.fr`), is is impractical to make accurate benchmarks. The performance gap is however easily noticable, YAVGUI being multiple times as fast as VGI when executing algorithms.

Included next are screenshots from VGI and YAVGUI. The same automaton was opened the usual way in each of the tools, with automatic placement of states and transitions enabled.

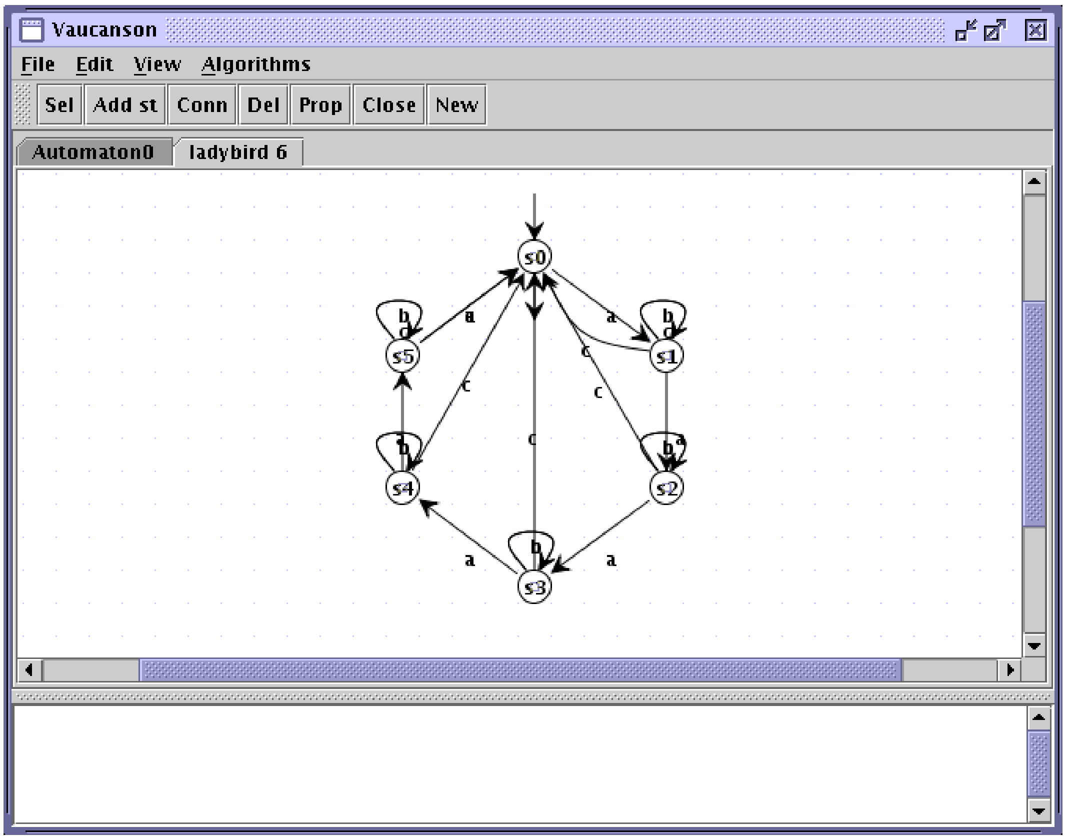

Figure 4.1: VGI's visual. Algorithms are accessed through the menu bar on top, while direct operations on the automaton's structure are activated via the buttons located below the menu bar. Text-based command results are displayed in the text field at the bottom.

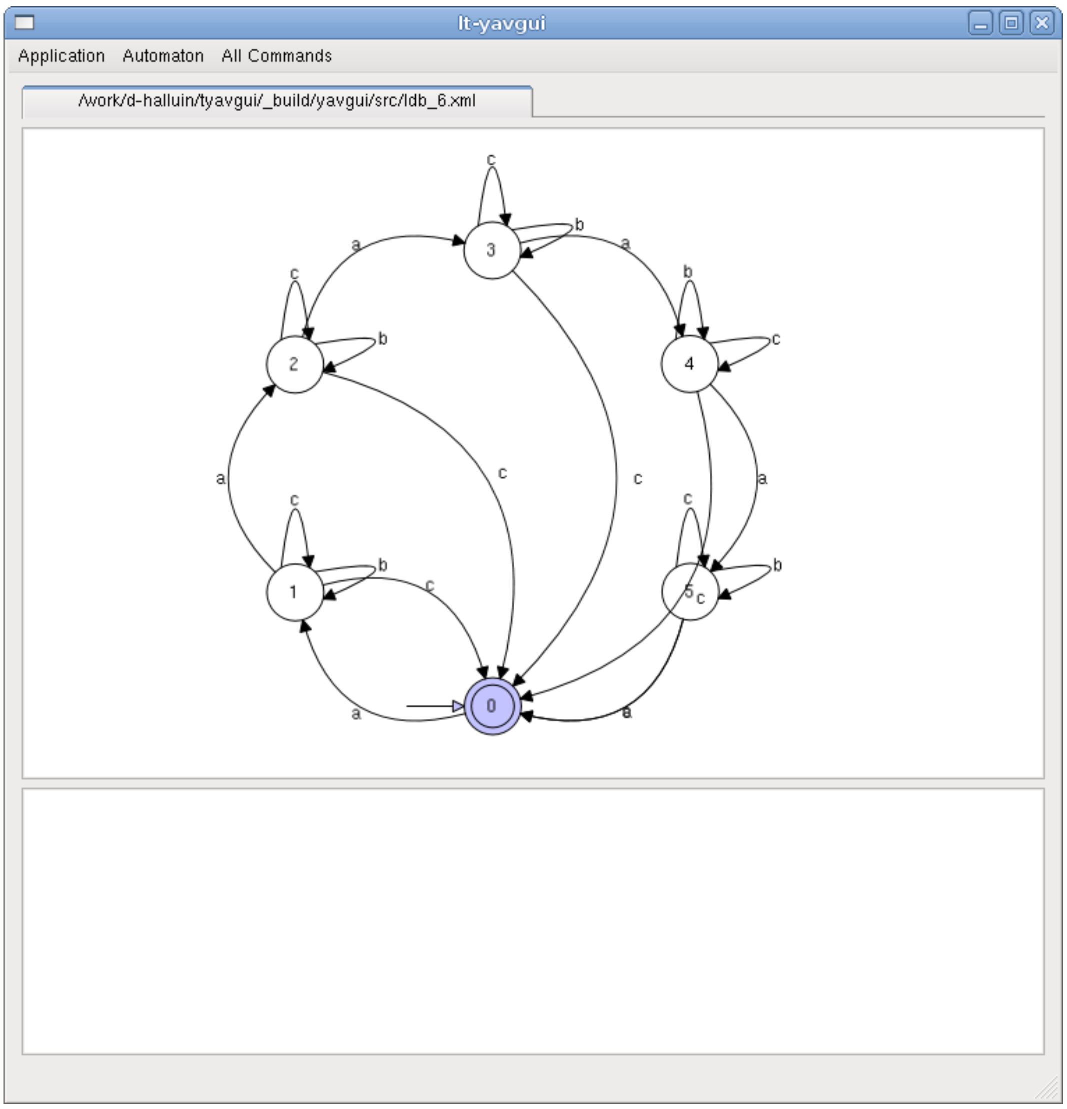

Figure 4.2: YAVGUI's visual. Algorithms are also accessed through the menu bar on top. Direct operations on the automaton's structure are however activated via intuitive, context-dependant mouse actions. Text-based command results are displayed in the text field at the bottom.

4.3 Future works

4.3.1 Finalization of the first version

At the time of this writing, two main features of YAVGUI have yet to be implemented: the complete set of commands corresponding to the algorithms available in LIBVCSN, and the structure manipulation functions that would allow an automaton to be defined from scratch. These features are to be added before the release of YAVGUI in VAUCANSON 1.3.

4.3.2 Directions to explore

This section introduces features that may be included in YAVGUI once a fully functional first version is released.

Step-by-step Algorithms

With a minor equipment of LIBVCSN's algorithm code, algorithms could be interrupted after every major step. Automata could be displayed in-between, highlighting newly added or modified states and transitions. This would give value to YAVGUI as a learning tool and help debug algorithms' code.

Real-time type instantiation

By dissociating the core engine from the front end and linking the core engine to the front end at runtime, users would be able to define new automaton types, instantiate those types and their algorithms, and provide immediate access to the new types inside YAVGUI without having to close the front-end.

Chapter 5

Conclusion & perspectives

5.1 Related works

YAVGUI is tightly tied to the work of the other members of the VAUCANSON GROUP:

- Florian Lesaint's work on a XML format for VAUCANSON is directly used by YAVGUI (Lesaint, 2008).
- Jérôme Galtier is currently developing on heuristics for state, transitions and label placement in a 2-dimentional plane
- Jimmy Ma and Vivien Delmon keep improving LIBVCSN, which is observable in all layers of VAUCANSON.
- The external polymorphism presented in chapter 3 is described in a report by Nicolas Pouillard and Damien Thivolle (Pouillard and Thivolle, 2006). This report also presents techniques to implement a static/dynamic bridge, which will come into play as real-time type instantiation is considered.

5.2 Conclusion

YAVGUI is a simple and easy-to-use GUI for VAUCANSON. It was developed for two major reasons: Firstly, it provides an immediate visual tool to Epita students, starting September 2008. Secondly, it acts as a proof of concept for the development of a more complete GUI by the National Taiwan University and lays down reusable code.

Above all, YAVGUI is an experimental tool on which design features can easily be implemented and tested. The key feature of YAVGUI's initial release is the interaction with a generic library, LIBVCSN. In the future, many more features may be experimented on, such as the step-by-step execution of algorithms.

With YAVGUI, the VAUCANSON platform now has a functionning graphical automaton manipulation tool. A first version of YAVGUI is to be released will VAUCANSON 1.3, which will be available from the VAUCANSON website (VAUCANSON Group, 2008a) by the end of July 2008.

Appendix A

Inside YAVGUI's code

This appendix is targeted at those working on YAVGUI. It explains the code organization and points out where to start.

The current version of this appendix was written on July 16th 2008. Due to important changes that are to take place in the following weeks, some of the information below may be outdated.

Getting YAVGUI

YAVGUI has its own branch in VAUCANSON's SVN repository.

Check out the source code

```
svn co https://svn.lrde.epita.fr/svn/vaucanson/branches/yavgui
cd yavgui
```

Or, if YAVGUI has already been merged into the trunk:

```
svn co https://svn.lrde.epita.fr/svn/vaucanson/trunk
cd trunk
```

Compile YAVGUI

```
./bootstrap
mkdir _build
cd _build
../configure --with-qt=<path to Qt4, usually /usr/share/qt4/bin>
make
cd yavgui/src
make
```

Run YAVGUI

```
./yavgui
```

General code organization

This section is meant to be updated as YAVGUI evolves.

- The source files are located in `./yavgui/src` once the above check-out step has been done.
- Each of the source files relates to one of YAVGUI's main components, the core engine or the front end.
- As a general rule, files prefixed by `g_` and `dialog_` relate to the front end, while files prefixed by `vcsn_`, `command_` and `commands_` relate to the core engine.
- The `boost::variant` holding all LIBVCSN types is defined in `vcsn_automaton_variant.hh`.
- An example of a `boost::static_visitor` is located in `g_state_adder.hh`.
- All commands are visitors defined using the macros located in `command_macros.hh`.

Appendix B

VAUCANSON's timer utility

VAUCANSON has a small utility for timing and profiling its code. This utility was designed to be used in VAUCANSON and the Tiger Compiler (LRDE, 2008). This appendix explains how to use said utility in VAUCANSON and other projects.

Features

- User-defined tasks via code equipment.
- Low-cost statistics gathering at run-time.
- Output in text or DOT format (to be processed by GraphViz).
- Several verbose settings available.

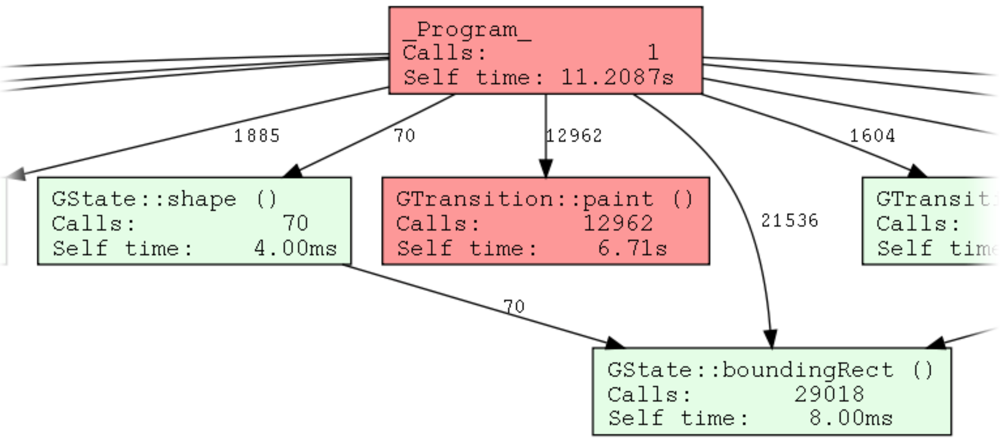

Figure B.1: Part of a DOT format output, processed.

Using the timer in VAUCANSON

The timer source code is located in `./include/vaucanson/misc/`. All the related files are prefixed by `timer`. The file `global_timer.hh` defines macros that facilitate the use of a global timer.

VAUCANSON's source code is already equipped with task definitions for most algorithms and I/O operations. Both TAF-KIT and YAVGUI include a global timer. In TAF-KIT, results can be displayed with a command-line option (see TAF-KIT usage). In YAVGUI, results are automatically generated and located in the file `timer.dot`, in the program's directory.

Advanced usage is described extensively in `timer.hh`.

Using the timer in other projects

Get the timer source code

```
cd include/vaucanson/misc
cp timer* global_timer.hh <project directory>
```

Documentation and usage is provided in `timer.hh`.

Appendix C

Bibliography

Cadilhac, Bigaignon, T. (2006). Remodeling Vaucanson. http://lrde.org/Publications/20060517-Seminar-Cadilhac-Bigaignon-Terrones.

Claveirole, T. (2004). Vaucanson overview. http://www.lrde.org/Publications/20041124-Seminar-Claveirole-VaucansonOverview-Report.

Friedman, E. and Maman, I. (2008). Boost variant. http://www.boost.org/doc/libs/1_35_0/doc/html/variant.html.

Lazzara, G. (2006). Automata and performances. http://www.lrde.org/Publications/200607-Seminar-Lazzara.

Lesaint, F. (2008). Fsmxml and its application in Vaucanson. http://lrde.org/Publications/200806-Seminar-Lesaint.

LRDE (2008). The Tiger Compiler Project. http://lrde.org/Tiger/WebHome.

Mohri, M., C. N. Pereira, F., and Riley, M. D. (2008). AT&T FSM library. http://www.research.att.com/~fsmtools/fsm/.

Pouillard, N. and Thivolle, D. (2006). Dynamization of C++ Static Libraries. Unpublished.

Riley, M., Schalkwyk, J., Skut, W., Allauzen, C., and Mohri, M. (2008). OpenFst. http://www.openfst.org/.

Sakarovitch, J. (2003). *Éléments de théorie des automates*. Vuibert informatique.

Sigoure, B. (2008). AutoTroll. http://www.tsunanet.net/autotroll/.

Standard, K. (2008). Elements of good GUI design. http://computerprogramming.suite101.com/article.cfm/elements_of_good_gui_design.

Vaucanson Group (2008a). Vaucanson home page. http://vaucanson.lrde.epita.fr/.

Vaucanson Group (2008b). Vaucanson documentation. http://www.lrde.epita.fr/dload/vaucanson/latest/ref/html/index.html.

Tognazzini, B. (2008). First principles of interaction design. http://www.asktog.com/basics/firstPrinciples.html.

Trolltech (2008). Qt reference documentation. http://doc.trolltech.com/4.3/index.html.