Building language interfaces with Vaucanswig

Author: Raphael Poss
Contact: raph@lrde.epita.fr
Date: January 2005
Version: Id

This document describes how to use Vaucanswig to produce interfaces with other languages.

Contents

Background

Vaucanswig is a set of SWIG wrapper definitions for the Vaucanson library.

SWIG takes Vaucanswig as input, and generates code to link between any supported scripting language and C++. In that sense, Vaucanswig is already "meta", because it ultimately supports several scripting languages. But still, even Vaucanswig itself is automatically generated, and this "meta-build" process is described in a separate document.

The document you are reading explains how to *use* Vaucanswig once it has been generated.

General idea

Once Vaucanswig has been generated, it is composed of *input files* to SWIG.

To use Vaucanson in a target scripting language, two steps are necessary:

1. Produce C++ sources for the interface (running SWIG).
2. Compile these sources.

Step 1 only requires Vaucanswig sources and a decent version of SWIG.
Step 2 requires the Vaucanson library and the extension libraries for the selected target language.

SWIG modules (MODULES)

Vaucanswig defines a number of SWIG modules.

The list of SWIG modules, hereinafter named MODULES, contains:

Name of module	Description
core	the core of vaucanswig.
K_context	for each K, the definition of the algebraic context K ("K" can be "usual", "numerical", "tropical" and so on)
K_automaton	definition of the Automaton and Expression types in context K
K_alg_A	for each algorithm A, the definition of the specific instance of A in context K. (A can be "complete", "standard", "product" and so on)
K_algorithms	a convienient wrapper for context K with "shortcuts" to all the algorithms instanciated for K.

Note that the name of SWIG modules are closely related to the namespace where the corresponding features can be found in the target scripting language.

Then, for each module M, two items are available:

Item	Description
src/vaucanswig_M.i	the dedicated SWIG source file
src/M.deps	(optional, may not exist) a file containing a list of modules that M is dependent upon. If the file is empty, two cases apply: • M is "core" - no dependency • M is not "core" - it depends on "core".

The first item is the most important. The second is only useful to create automated build processes which require dependency rules.

C++ sources specific to the target scripting language (T.S.L.)

Each TSL needs a different set of wrapper for the Vaucanswig modules.

For any given TSL, source files for the MODULES can be created by SWIG by running the following pseudo-algorithm:

```
$ for M in ${MODULES}; do
    ${SWIG} -noruntime -c++ -${TSL} \
       -I${VAUCANSWIGDIR}/src \
       -I${VAUCANSWIGDIR}/meta \
       -I${VAUCANSON_INCLUDES}  \
       ${VAUCANSWIGDIR}/src/vaucanswig_${M}.i
  done
```

Where:

- ${TSL} is the SWIG option pertaining to the language (python, java ...)
- ${VAUCANSWIGDIR} is the root directory of Vaucanswig.

- `${SWIG}` is the path to the SWIG binary.
- `${VAUCANSON_INCLUDES}` is the base directory of the Vaucanson library.

Compilation of the binaries for the target scripting language

The previous step creates a bunch of C++ source files of the form:

```
vaucanswig_${M}_wrap.cxx
```

They should be compiled with the C++ compiler supported by the TSL.
The C++ compilation should use the following flags:
`-DINTERNAL_CHECKS -DSTRICT -DEXCEPTION_TRAPS`: Use for more secure code in Vaucanson.
`-I${VAUCANSON_INCLUDES}`: Specify the location of the Vaucanson library headers.
`-I${VAUCANSWIGDIR}/src -I${VAUCANSWIGDIR}/meta`: Needed by Vaucanswig.
In addition, any "compatibility" flags required by Vaucanson for this particular C++ compiler should be used as well.

Automake support for Python as a TSL

According to the previous section, a Makefile.am file is generated in the subdirectory `python/`.
It contains four main parts:

- A header:

```
##
## Set INCLUDES for compilation of C++ code.
##

# FIXME: the python path is hardcoded, this is NOT good.
INCLUDES = -I/usr/include/python2.2 \
           -I$(srcdir)/../src -I$(srcdir)/../meta \
           -I$(top_srcdir)/include -I$(top_builddir)/include

##
## Set AM_... flags.
##

# According to spec.
AM_CPPFLAGS = -DINTERNAL_CHECKS -DSTRICT -DEXCEPTION_TRAPS
# We want lots of debugging information in the wrapper code.
AM_CXXFLAGS = $(CXXFLAGS_DEBUG)
# For Libtool, to generate dynamically loadable modules.
AM_LDFLAGS = -module -avoid-version
```

- The list of binary targets (the shared objects - DLL):

```
# for each MODULE:
pyexec_LTLIBRARIES += libvs_$(MODULE).la
```

- The list of Python source files:

```
# for each MODULE:
python_PYTHON += vaucanswig_$(MODULE).py
```

- Build specifications for binary targets:

```
# for each MODULE:
libvs_$(MODULE)_la_SOURCES = vaucanswig_$(MODULE)_wrap.cxx

# If the module is "core":
#    # This should be the only dependency against static, non-template
#    # Vaucanswig code. And make it a dependency to the SWIG runtime.
#    libvs_core_la_LIBADD = ../meta/libvv.la -lswigpy

# Else:
#    If src/$(MODULE).deps is empty:
#        libvs_$(MODULE)_la_LIBADD = libvs_core.la
#    Else:
#        for each DEPENDENCY in src/$(MODULE).deps do:
#          libvs_$(MODULE)_la_LIBADD += libvs_$(DEPENDENCY).la
```

Additionnaly, the following (not important) parts are generated for convenience purposes:

- Rules to rerun SWIG in case something changes in Vaucanswig:

```
vaucanswig_*_wrap.cxx vaucanswig_*.py: ../src/vaucanswig_*.i
     $(SWIG) -noruntime -c++ -python -I... \
                 -o vaucanswig_*_wrap.cxx \
                 ../src/vaucanswig_*.i
```

- Installation and uninstallation hooks.

Automake support for the TSL-independent code

In order to make things comply to the spirit of the Autotools, a convenience `Makefile.am` is generated in the `src/` directory.

It contains a definition of EXTRA_DIST with all the SWIG module source files, of the form: `vaucanswig_$(MODULE).i`