Tool Description

Wolfgang Haid and Kai Huang
Revision 947, January 24, 2008

This document describes the tools for processing an XML process network description and the associated C source code to generate an executable SystemC model. Fig. 1 gives an overview of the tool chain.

If you just want to get started with the DOL package (`dol_ethz.zip`), you can directly go to section 1 and read the background information given in the first part of this document later on. You may also visit http://www.tik.ee.ethz.ch/∼shapes for further information.

The application programmer provides an XML file according to the process network XML Schema definition `processnetwork.xsd`. Additionally, the application programmer provides a C source code file for each process. With the tool chain described in this section, these sources can be converted into an executable SystemC application.

The main tool of the tool chain, **dol**, is written in Java. Therefore, a Java compiler **javac** and the Java application launcher **java** are used in the tool chain. In the code, Java features are used that have been newly introduced in the Java Platform 5.0. Therefore, appropriate versions of **javac** and **java** are required (**javac** and **java** version 1.5 as included in the J2SE 5.0 JDK). For creating the SystemC application, the C/C++ compiler **g++** (version 3.3 or higher) is used. Besides these standard tools, the following libraries and tools are used in the tool chain:
As libraries,

- **jdom:** http://www.jdom.org/ (version 1.0),
- **xerces:** http://xerces.apache.org/xerces2-j/ (version 2.8.0), and
- **SystemC:** http://www.systemc.org/ (version 2.1) are used.

Note: The **jdom** and **xerces** libraries are already contained in the DOL distribution, so there is no need to download them.
As build tools,

- **ant:** http://ant.apache.org/ (version 1.6.5), and

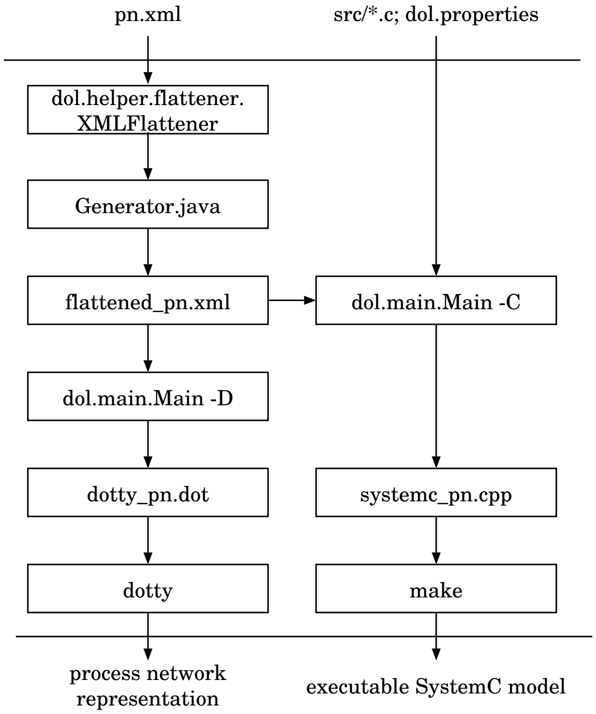

Figure 1: Tool chain.

- **make:** http://www.gnu.org/software/make/ (version 3.81) are used.

Additionally, **dotty** (http://hoagland.org/Dot.html) is used for visualizing process networks.

The following paragraphs describe the tool chain step-by-step. For each step, it is described *what* is done, followed by a short description *how* it is done. Note that the archive file `dol_ethz.zip` contains an **ant** build file and, alternatively, a bash script which perform these steps in an automated manner. (So, you might run the script first and then read what has actually be done.) The scripts are described at the end of this section.

Throughout the following description, it is assumed that the Java classpath is set correctly. That means that the `CLASSPATH` environment variable on the system contains the files `jdom.jar` and `xercesImpl.jar`. Moreover, `CLASSPATH` must include the `dol.jar` file. Note that the classpath separator is the colon ":" for Linux and Solaris platforms whereas it is the semicolon ";" for Windows. It is also possible to set the classpath using the –classpath commandline argument with **javac** and **java**. (This is done in the script mentioned above, for instance.)

The first step in the tool chain is the flattening of the process network description. During flattening, all children elements of <iterator> are instantiated by evaluating the corresponding <append> elements. The result is an XML file which still conforms to the XML Schema definition but does not make use of any <variable>, <function>, <iterator>, or <append> elements. In particular the Java class **XMLFlattener** is used for that purpose. For the specified XML file, **XMLFlattener** creates a Java class that can generate the flattened XML file. A typical sequence of commands to generate the flattened XML file is shown below. The second argument of **XMLFlattener** is the name of the class to generate which is stored in the corresponding `.java` file.

```
$ java dol.helper.flattener.XMLFlattener pn.xml Generator
$ javac Generator.java
$ java Generator.java > flattened_pn.xml
```

The flattened XML is further processed by **dol**. On the one hand, it is possible to generate a representation of the process network such that it can be displayed using the graph visualization tool **dotty**. On the other hand, the flattened XML is used by **dol** together with the C source code of the processes to generate an executable SystemC application. Before calling **dol**, make sure that in `dol.properties` all paths (for instance, the SystemC `include` directory path) are correctly set.
To generate a network description displayable by **dotty**, use **dol** with the `-D` flag. To specify the process network file, use the `-P` flag. The `-c` flag is used to enable a basic consistency check of the process network. In this check, **dol** will check whether each port is connected to some channel, for instance. A typical call of **dol** and **dotty** might be as follows:

```
$ java dol.main.Main -P flattened_pn.xml -D dotty_pn.dot -c
$ dotty dotty_pn.dot
```

To generate the source code for an executable SystemC application, use **dol** with the `-H` flag. Note that **dol** assumes that the source files are located in the subdirectory `src` of the working directory. **dol** will create a directory with the specified name containing all required source files for generating a SystemC application, including a `Makefile`. To compile and run the

SystemC application, simply use **make** and call the generated application **sc_application**.

```
$ cp $sourcefiles ./src
$ java dol.main.Main -P flattened_pn.xml -H systemc -c
$ cd systemc/src
$ make
$ ./sc_application
```

1 Automated Operation of the DOL Tool Chain

The DOL package comes with a couple of example applications for which the tool chain can be run automatically. Before doing so, make sure that the following tools are in place (refer to the previous section for links to obtain those tools):

1. C/C++ environment: compiler, linker
2. Java environment: javac, java
3. Build environment: make, Ant (version 1.6.5 or greater)
4. SystemC environment (version 2.1 or greater)

To run the DOL tool chain, simply walk through the following step-by-step instructions:

1. Copy the `dol_ethz.zip` archive to some directory, for instance `dol_ethz`:
   `~/dol_ethz>unzip dol_ethz.zip`

2. After unzipping the archive, a file `build_zip.xml` will be located in the working directory. Change the properties at the top of this file, that is, `systemc.inc`, `systemc.lib`, and `classpathdelimiter` as needed. You may also need to set the `javac.executable` property: Since an **ant** installation is always coupled with a specific java compiler — or a specific java compiler version — compilation may fail when using a "wrong" **ant**. In this case, you can specify a certain Java compiler in the `javac.executable` property and set the `use.external.javac` property to `"yes"`. An alternative approach might work on systems where more than one **ant** versions are installed. In that case, you might use a specific **ant** version associated with a more recent java version (for instance, use `ant-1.6.5` instead of `ant` when the standard `ant` is

   linked with `ant-1.5.4`).
   The properties `dol.path`, `xmlns`, and `xsischema` normally need not be changed.

3. Use **ant** (or a specific version of **ant**, see above) to generate the `src/dol.properties` file based on the settings and copy it to the `dol.jar`:
   ```
   ~/dol_ethz>ant -f build_zip.xml config
   ```

4. Use **ant** to set up the build directory structure `build/bin/main` and copy all the necessary resources to this directory:
   ```
   ~/dol_ethz>ant -f build_zip.xml compile
   ```

(4a.) Note: You may also use the `all` target to perform the previous two steps in one call:
```
~/dol_ethz>ant -f build_zip.xml all
```

5. Change to `build/bin/main` and run the example using the provided **ant** build file `runexample.xml`. `runexample.xml` will trigger the flattening, C code generation, compilation, and execution of the specified example.
   ```
   ~/dol_ethz>cd build/bin/main
   ~/dol_ethz/build/bin/main>ant -f runexample.xml
   -Dnumber=1
   ```

Alternatively, you may also use the script `runexample.sh`: `~/dol_ethz/build/bin/main>./runexample 1` An example output for **ant** is shown below.

```
$ ant -f runexample.xml -Dnumber=1
Buildfile: runexample.xml

showversion:

showantversion:
     [echo] Use Apache Ant version 1.6.5 compiled on June 2 2005.

showjavaversion1:
     [echo] Use Java version 1.5.0_07 (required version: 1.5.0 or higher).

showjavaversion2:

showjavacversion1:
     [echo] Use Java version 1.5.0_07 (required version: 1.5.0 or higher).

showjavacversion2:

runexample:
```

```

prepare:
     [echo] Create directory example1.
    [mkdir] Created dir: D:\shapes\dolPrototype\trunk\build\bin\main\example1
     [echo] Copy C source files.
    [mkdir] Created dir: D:\shapes\dolPrototype\trunk\build\bin\main\example1\src
     [copy] Copying 6 files to D:\shapes\dolPrototype\trunk\build\bin\main\
     example1\src

validate:
     [echo] check XML compliance of example1_flattened.xml.
     [java] D:\shapes\dolPrototype\trunk\examples/example1/example1.xml is valid
.

flatten1:
     [echo] Create flattened XML example1_flattened.xml.
     [java] ......................................
    [javac] Compiling 1 source file to D:\shapes\dolPrototype\trunk\build\bin\
    main\example1

flatten2:

dol1:
     [echo] Run DOL.
     [java] Read process network from XML file
     [java]  -- full filename: file:/D:/shapes/dolPrototype/trunk/build/bin/main
/example1/example1_flattened.xml
     [java]  -- Process network model from XML [Finished]

     [java] Consistency check:
     [java] Checking resource name ...
     [java] Checking channel ports ...
     [java] Checking Process connection ...
     [java] Checking channel connection ...
     [java] Checking instantiation ...
     [java]  -- Consistency check [Finished]

     [java] Generating ProcessNetwork in Dotty format:
     [java]  -- Generation [Finished]

     [java] Generating HdS package:
     [java] basename:
     [java] basename:
     [java] basename:
     [java] basename:
     [java]  -- Generation [Finished]


dol2:

systemc:
     [echo] Make SystemC application.
     [exec] g++ -g -O0 -DINCLUDE_PROFILER -I/cygdrive/c/tools/systemC/systemc-2.
1.v1/include -Ilib -Isc_wrappers -Iprocesses
-c -o sc_application.o sc_application.cpp
     [exec] g++ -g -O0 -DINCLUDE_PROFILER -I/cygdrive/c/tools/systemC/systemc-2.
```

```
1.v1/include -Ilib -Isc_wrappers -Iprocesses
-c -o process.o lib/process.c
      [exec] g++ -g -O0 -DINCLUDE_PROFILER -I/cygdrive/c/tools/systemC/systemc-2.
1.v1/include -Ilib -Isc_wrappers -Iprocesses
-c -o generator_wrapper.o sc_wrappers/generator_wrapper.cpp
      [exec] g++ -g -O0 -DINCLUDE_PROFILER -I/cygdrive/c/tools/systemC/systemc-2.
1.v1/include -Ilib -Isc_wrappers -Iprocesses
-c -o consumer_wrapper.o sc_wrappers/consumer_wrapper.cpp
      [exec] g++ -g -O0 -DINCLUDE_PROFILER -I/cygdrive/c/tools/systemC/systemc-2.
1.v1/include -Ilib -Isc_wrappers -Iprocesses
-c -o square_wrapper.o sc_wrappers/square_wrapper.cpp
      [exec] g++ -g -O0 -DINCLUDE_PROFILER -I/cygdrive/c/tools/systemC/systemc-2.
1.v1/include -Ilib -Isc_wrappers -Iprocesses -o sc_application sc_application.o
process.o generator_wrapper.o consumer_wrapper.o square_wrapper.o
/cygdrive/c/tools/systemC/systemc-2.1.v1/lib-cygwin/libsystemc.a
      [echo] Run SystemC application.
    [concat] consumer: 0.000000
    [concat] consumer: 1.000000
    ...
    [concat] consumer: 324.000000
    [concat] consumer: 361.000000

BUILD SUCCESSFUL
Total time: 19 seconds
```

A Package Contents

Below, the contents of the `dol_ethz.zip` file archive is shown.
dol_ethz.zip

	JDOM_LICENSE.txt, XERCES_LICENSE.txt, LICENSE.txt, NOTICE: software licenses README.txt: readme file Version.txt: version number build_zip.xml: ant build file
bin	dol.jar: DOL resources jdom.jar, xercesImpl.jar: third-party libraries
docs	ApplicationExamples.pdf ArchitectureExamples.pdf CodingGuide.pdf MappingExamples.pdf ToolGuide.pdf XMLSpecification.pdf
examples	example sources and run scripts
schema	XML Schema definitions