The distributed operation layer (DOL) is a framework that enables the (semi-) automatic mapping of applications onto the multiprocessor SHAPES architecture platform. The DOL consists of basically three parts:

- **DOL Application Programming Interface:** The DOL defines a set of computation and communication routines that enable the programming of distributed, parallel applications for the SHAPES platform. Using these routines, application programmers can write programs without having detailed knowledge about the underlying architecture. In fact, these routines are subject to further refinement in the hardware dependent software (HdS) layer.
- **DOL Functional Simulation:** To provide programmers a possibility to test their applications, a functional simulation framework has been developed. Besides functional verification of applications, this framework is used to obtain performance parameters at the application level.
- **DOL Mapping Optimization:** The goal of the DOL mapping optimization is to compute a set of optimal mappings of an application onto the SHAPES architecture platform. In a first step, XML based specification formats have been defined that allow to describe the application and the architecture at an abstract level. Still, all the information necessary to obtain accurate performance estimates is contained.

DOL Package

The DOL package (click to download) includes the following files. Refer to the Get Started section below for instructions how to set up the DOL framework on a computer.

- dol.jar archive and compile scripts for the DOL functional simulation
- Third-party Java libraries used by DOL
- Example applications (source code and process network XML)
- Example architecture and mapping XML files
- Documentation of DOL API, DOL functional simulation, XML schema semantics, and examples

DOL API Documentation

The two links below lead to the API documentation of the DOL. Since the DOL is written in Java and the comments are written in Javadoc style, the Javadoc documentation contains more information. The Doxygen documentation, however, contains graphical class diagrams of the DOL, which are not provided by Javadoc.

- Javadoc Documentation
- Doxygen Documentation

Get Started

This section provides a guide how to set up the DOL on a computer. This guide leads through the necessary steps, starting from the download of the DOL to the execution of an example

application using the DOL functional simulator. More detailed information is available in the tool guide which is included in the DOL package.

The DOL runs under UNIX/Linux as well as under Microsoft Windows. In particular, the DOL has been tested under Debian GNU/Linux 3.1 and under Microsoft Windows XP, Service Pack 2, (using Cygwin).

The requirements on both platforms are:

- C/C++ environment: compiler, linker
- Java environment: javac, java
- Build environment: make, Ant (version 1.6.5 or greater)
- SystemC environment (version 2.1 or greater)

If the above mentioned environments are in place, do the following to execute one of the application examples.

1. Download the DOL package here and copy it to the directory of your choice.
2. `$ unzip dol_ethz.zip`
3. Set the following three properties in build_zip.xml:
   - `<property name="systemc.inc" value="YYY/include"/>`
     Replace YYY by the path of your SystemC installation.
   - `<property name="systemc.lib" value="YYY/lib-linux/libsystemc.a"/>`
     Replace YYY by the path of your SystemC installation.
4. `$ ant -f build_zip.xml all`

   The output should look similar to the following one:

```
Buildfile: build_zip.xml

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:

config:
[echo] Create new dol.properties file.
[copy] Copying 1 file to D:\sandbox\dol\bin
[jar] Updating jar: D:\sandbox\dol\bin\dol.jar
[delete] Deleting: D:\sandbox\dol\bin\dol.properties

compile:
[echo] Create build directory tree.
[mkdir] Created dir: D:\sandbox\dol\build
[mkdir] Created dir: D:\sandbox\dol\build\bin
[mkdir] Created dir: D:\sandbox\dol\build\bin\main
```

```
[copy] Copying 1 file to D:\sandbox\dol\build\bin\main
[copy] Copying 1 file to D:\sandbox\dol\build\bin\main

updatebuildxml:

updatebuildxml1:

updatebuildxml2:

all:

BUILD SUCCESSFUL
Total time: 1 second
```

5. `$ cd build/bin/main`
6. `$ ant -f runexample.xml -Dnumber=1`

The output should look then similar to the following one:

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

showversion:

showantversion:
[echo] Use Apache Ant version 1.6.5 compiled on February 17 2006.

showjavaversion1:
[echo] Use Java version 1.5.0_06 (required version: 1.5.0 or higher).

showjavaversion2:

showjavacversion1:
[echo] Use Java version 1.5.0_06 (required version: 1.5.0 or higher).

showjavacversion2:

runexample:

prepare:
[echo] Create directory example1.
[mkdir] Created dir: /home/user/dol/build/bin/main/example1
[echo] Copy C source files.
[mkdir] Created dir: /home/user/dol/build/bin/main/example1/src
[copy] Copying 6 files to /home/user/dol/build/bin/main/example1/src

validate:
[echo] check XML compliance of example1_flattened.xml.
[java] /home/user/dol/examples/example1/example1.xml is valid.
```

```
flatten1:
[echo] Create flattened XML example1_flattened.xml.
[java] ....................................
[javac] Compiling 1 source file to /home/user/dol/build/bin/main/example1

flatten2:

dol1:
[echo] Run DOL.
[java] Read process network from XML file
[java] -- full filename:
file:/home/user/dol/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 SystemC package:
[java] -- Generation [Finished]


dol2:

systemc:
[echo] Make SystemC application.
[exec] g++ -g -O0 -I/home/user/base/resources/lib/systemC/include -Ilib
-Isc_wrappers -Iprocesses -c -o sc_application.o sc_application.cpp
[exec] g++ -g -O0 -I/home/user/base/resources/lib/systemC/include -Ilib
-Isc_wrappers -Iprocesses -c -o process.o lib/process.c
[exec] g++ -g -O0 -I/home/user/base/resources/lib/systemC/include -Ilib
-Isc_wrappers -Iprocesses -c -o generator_wrapper.o
sc_wrappers/generator_wrapper.cpp
[exec] g++ -g -O0 -I/home/user/base/resources/lib/systemC/include -Ilib
-Isc_wrappers -Iprocesses -c -o consumer_wrapper.o
sc_wrappers/consumer_wrapper.cpp
[exec] g++ -g -O0 -I/home/user/base/resources/lib/systemC/include -Ilib
-Isc_wrappers -Iprocesses -c -o square_wrapper.o sc_wrappers/square_wrapper.cpp
[exec] g++ -g -O0 -I/home/user/base/resources/lib/systemC/include -Ilib
-Isc_wrappers -Iprocesses -o sc_application sc_application.o process.o
```

```
generator_wrapper.o consumer_wrapper.o square_wrapper.o
/home/user/base/resources/lib/systemC/lib-linux/libsystemc.a
[echo] Run SystemC application.
[concat] consumer: 0.000000
[concat] consumer: 1.000000
[concat] consumer: 4.000000
[concat] consumer: 9.000000
[concat] consumer: 16.000000
[concat] consumer: 25.000000
[concat] consumer: 36.000000
[concat] consumer: 49.000000
[concat] consumer: 64.000000
[concat] consumer: 81.000000
[concat] consumer: 100.000000
[concat] consumer: 121.000000
[concat] consumer: 144.000000
[concat] consumer: 169.000000
[concat] consumer: 196.000000
[concat] consumer: 225.000000
[concat] consumer: 256.000000
[concat] consumer: 289.000000
[concat] consumer: 324.000000
[concat] consumer: 361.000000

BUILD SUCCESSFUL
Total time: 22 seconds
```

Done.