CASE Program
System Build Installation Guide

December 10, 2018

Background

This document describes the steps required to install and configure the components of the CASE system build tool-chain (see Figure below). The final section includes instructions on how to send an AADL model through the system build.

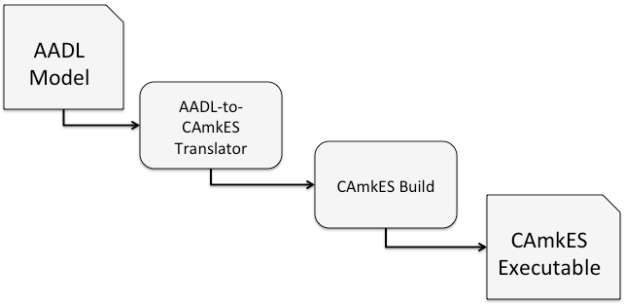

Figure 1: Basic Tool-chain Flow

AADL-to-CAmkES Translator

The functionality for the headless AADL-to-CAmkES translator is contained in two Git repositories:

- `https://github.com/sireum/osate-headless.git` -- contains scripts to invoke OSATE headlessly and translate an OSATE project into a distilled JSON form called AIR (*AADL Intermediate Representation*).
- `https://github.com/sireum/act-plugin.git` -- contains a Java/Scala-based tool that takes this JSON representation and any auxiliary C source files and translates to CAmkES.

To set up and run these tools:

1. Have the AADL model you want to translate ready in OSATE project form:
    - All AADL files in the project should be in the project's root directory, e.g., `/path/to/project`.
    - Know what system implementation you want to translate, e.g., `top.impl`, and what file it is in, e.g., `Main.aadl`.
    - If you have any auxiliary C source files referenced in your project, put them in one directory and know its location, e.g., `/path/to/auxCode`.

- Know the path where you want the translator to put the generated CAmkES, e.g., `/path/to/project_CAmkES`.

2. Clone the two Git repositories:

```
$ git clone https://github.com/sireum/osate-headless.git \
  osate-headless
$ git clone https://github.com/sireum/act-plugin.git act-plugin
```

3. Change to the ACT repository directory and compile the ACT tool.

```
$ cd act-plugin
$ bash bin/test.sh
```

This will build an executable JAR at `bin/act`.

4. Change to the OSATE translator repository directory and prepare the build.

```
$ bash prelude.sh
$ mkdir -p ~/.sireum/phantom
```

This will download a build tool called Mill, which performs the headless invocation of OSATE.

CAmkES Build Environment

CAmkES build dependencies recommends Ubuntu 18.04 since 16.04 and earlier will keep some packages (`CMake`, GCC, etc.) at older versions. This setup follows the suggestions of the Host Dependencies page (https://docs.sel4.systems/HostDependencies) linked on the seL4/CAmkES tutorial (https://docs.sel4.systems/GettingStarted#setting-up-your-machine).

1. On a fresh Ubuntu 18.04 installation you will need to enable the "universe" repository.
```
$ sudo add-apt-repository universe
```

2. The basic build package on Ubuntu is the build-essential package. To install run the following commands.
```
$ sudo apt-get update
$ sudo apt-get install build-essential
```

3. Add other base dependencies for building seL4 projects on Ubuntu.
```
$ sudo apt-get install cmake ccache ninja-build
```

```
$ sudo apt-get install python-dev python-pip python3-dev \
  python3-pip
$ sudo apt-get install libxml2-utils ncurses-dev
$ sudo apt-get install curl git doxygen
```

4. To build for ARM targets you will need a cross compiler. In addition, to run seL4 projects on a simulator you will need QEMU. Install these additional base dependencies.
```
$ sudo apt-get install gcc-arm-linux-gnueabi g++-arm-linux-gnueabi
$ sudo apt-get install gcc-aarch64-linux-gnu g++-aarch64-linux-gnu
$ sudo apt-get install qemu-system-arm qemu-system-x86
```

5. Optionally, you may install the hardware floating-point versions.
```
$ sudo apt-get install gcc-arm-linux-gnueabihf \
  g++-arm-linux-gnueabihf
```

6. Regardless of your Linux distribution, python dependencies are required to build seL4, the manual and its proofs.
```
$ pip install --user setuptools
$ pip install --user sel4-deps
```

7. To build a CAmkES based project on seL4, additional dependencies need to be installed on your host machine. Projects using CAmkES require Haskell and some extra python libraries in addition to the standard build tools. The following instructions cover the CAmkES build dependencies for Ubuntu/Debian. Ensure you have installed the dependencies listed in sections *sel4 Build Dependencies* and *Get Google's Repo* tool prior to building a CAmkES project.
```
$mkdir ~/bin
$ PATH=~/bin:$PATH
$ curl https://storage.googleapis.com/git-repo-downloads/repo > \
  ~/bin/repo
$ chmod a+x ~/bin/repo
$ mkdir google_repo_tool
$ cd google_repo_tool/
$ git config --global user.name "name"
$ git config --global user.email "name@email.com"
$ repo init -u https://android.googlesource.com/platform/manifest
```

8. The python dependencies required by the CAmkES build toolchain can be installed via pip.
```
$ pip install --user camkes-deps
```

9. The CAmkES build toolchain additionally requires Haskell. You can install the Haskell stack on your distribution by running the following commands.
```
$ curl -sSL https://get.haskellstack.org/ | sh
```

10. Next, install the following packages on your Ubuntu machine.
```
$ sudo apt-get install clang gdb
$ sudo apt-get install libssl-dev libclang-dev libcunit1-dev \
  libsqlite3-dev
$ sudo apt-get install qemu-kvm
```

11. The proofs in the L4.verified repository use Isabelle2017. To best way to make sure you have all the dependencies on your host machine is to follow the instructions listed in sections *Base Build Dependencies*. However you can avoid a full cross compiler setup. The dependencies for Isabelle you will need at least are listed here.
```
$ sudo apt-get install libwww-perl libxml2-dev libxslt-dev
$ sudo apt-get install mlton rsync
$ sudo apt-get install texlive-fonts-recommended \
  texlive-latex-extra texlive-metapost \
  texlive-bibtex-extra
```

12. To setup Isabelle, you will first need to pull the L4.verified project source using the *Google Repo Tool*. Use the following steps to setup Isabelle.
```
$ mkdir lv4_repo
$ cd lv4_repo/
$ repo init -u https://github.com/seL4/verification-manifest.git
$ repo sync
$ cd l4v
$ mkdir -p ~/.isabelle/etc
$ cp -i misc/etc/settings ~/.isabelle/etc/settings
$ ./isabelle/bin/isabelle components -a
$ ./isabelle/bin/isabelle jedit -bf
$ ./isabelle/bin/isabelle build -bv HOL-Word
```

13. Use repo to check `sel4test` out from GitHub. Its manifest is located in the *sel4test-manifest* repository.
```
$ mkdir seL4test
$ cd seL4test
$ repo init -u https://github.com/seL4/sel4test-manifest.git
$ repo sync
```

14. Configure an x86_64 build directory, with a simulation target to be run by Qemu. QEMU is a generic and open source machine emulator and virtualizer, and can emulate different architectures on different systems.

```
$ mkdir build-x86
$ cd build-x86
$ ../init-build.sh -DPLATFORM=x86_64 -DSIMULATION=TRUE
$ ninja
```

15. The build images are available in build-x86/images, and a script build-x86/simulation that will run QEMU with the correct arguments to run seL4test.

```
$ ./simulate
```

16. Now download CAmkES and run a simple application.

```
$ mkdir camkes-project
$ cd camkes-project
$ repo init -u https://github.com/seL4/camkes-manifest.git
$ repo sync
```

17. The following will configure, build, and run a simple example CAmkES system.

```
$ cd camkes-project
$ mkdir build
$ cd build
$ ../init-build.sh -DPLATFORM=sabre -DAARCH32=1 \
  DCAMKES_APP=adder  -DSIMULATION=1
$ ninja
$ ./simulate
```

Running the System Build

Consider the following steps.

1. First run the OSATE translator on your project.

```
$ ./mill-standalone cli -p /path/to/project -a Main.aadl \
  -o /path/to/output.json top.impl
```

The first time this is run, it will download a full installation of OSATE into the `~/.sireum/phantom` directory created in the first section of this document. The '`-p`' and '`-o`' options must contain absolute paths to the project and the output JSON file, respectively. Once the OSATE installation is loaded, subsequent invocations of the OSATE translator will not attempt to download a local copy of OSATE. To

load an updated version of OSATE, you must clear the contents of the `~/.sireum/phantom` directory.

2. Next run the executable ACT JAR on this output JSON file, to generate the CAmkES code.

```
$ ../act-plugin/bin/act -a /path/to/auxCode -o \
   /path/to/project_CAmkES/path/to/output.json
```