Project in software engineering ECCO KeFaX (ECCO Linux KErnel FeAture eXtraction) - documentation

Harald A. Weiner
JKU
Linz, Austria
Email: harald.weiner@jku.at

***Abstract*—This project aims to use an exisiting C code parser to walk through the produced abstract syntax tree (AST) to provide an importer to the *ECCO* (*Extraction and Composition for Clone-and-Own*) tool. The goal is to offer case studies for the feature model exploration implementations in *ECCO*. Various approaches have been explored and finally Eclipse MoDisco in combination with the Xtext plug-in have been chosen to reverse-engineer C programs and import them into the EMF (Eclipse Modeling Framework).**

I. Introduction

This project has been developed as part of my project in software engineering for the master course computer science. The tool *ECCO* (*Extraction and Composition for Clone-and-Own*)[1] is developed at the ISSE (Institute for Software Systems Engineering [2]) at Johannes Kepler University in Linz, Austria and can be found at [3]. It maps commonalities and differences of existing variants of a portfolio to a feature set. To provide a real-life case study, the Linux Kernel has been chosen as a demonstration example for the *ECCO* tool (the Linux kernel is a pretty well-known example / case study and a project with huge impacts in industry and research). At the moment, plain C and C++ programs are not supported by *ECCO* yet. Therefore, an importer should be written which is able to use the output of C parsers to extract the relevant information for *ECCO*. The KeFaX project should provide an importer for the Linux kernel to the ECCO tool. As a result, this project is supposed to parse the Linux .config file, set-up the minimal infrastructure of source code for the modules and parse the source code. At the end, this project should create an "input tree" data structure. Various approaches exist which have been evaluated for their suitability to the given task. This paper will present the steps taken so far.

May 09, 2016

II. Getting started

This project is provided as a set of Eclipse plug-ins. Depending on if you would like to use the project or just want to explore the source code there are different requirements. Both development and runtime execution have been tested under *Eclipse Modeling Luna SR2* and *Eclipse Modeling Mars.2 Release (4.5.2)* on AMD64 architecture with a Linux operating system and at least 8 GB RAM.

Warning: This is a prototype / proof-of-concept and is not intended to be used in production environments!!! It may contain some serious bugs, security issues or design flaws which might lead to data loss or data corruption. You have been warned ;-).

A. How-To use

1) Ensure that you have Git installed and that the git executable is in your current $PATH variable.
2) Download Eclipse Modeling IDE from
   - either https://eclipse.org/downloads/[4]
   - or https://eclipse.org/downloads/packages/release/luna/sr2[5]
3) Unzip and open the Eclipse IDE
4) Install Eclipse Modisco (Help→Install new software, select the predefined software site *Modeling package updates for Eclipse Mars*[6] or *Modeling package updates for Eclipse Luna*[7] and install either *Modisco/Modisco SDK (incubation) 0.13.2.201601200708* or *Modeling/-Modisco SDK (Incubation) 0.12.2.201501021045* (depending on your Eclipse version).
5) Install NeoEMF by opening the install new software dialog again and add https://timeraider4u.github.io/NeoEMF/[8] as NeoEMF update site. Install
   - *Base/NeoEMF Persistence framework*
   - *Backends/NeoEMF Blueprints adapter*
   - and *Backends/NeoEMF Blueprints implementation*

   each with version 0.0.1.2016040202
6) Install *org.xtext.antlr.generator* by adding https://timeraider4u.github.io/org.xtext.antlr.generator/[9] as an update site and selecting the feature *org.xtext.antlr.generator/org.xtext.antlr.generator.feature* with version 3.2.1.201604141818.
7) Install the modified version of *Xtext* by adding https://timeraider4u.github.io/xtext/[10] as an update site and select *Xtext/Xtext Complete SDK 2.9.0.v201604150031*
8) Install KeFax by adding https://timeraider4u.github.io/kefax/[11] as an update site and select *at.jku.weiner.kefax/at.jku.weiner.kefax* with version 0.1.0.201605080110

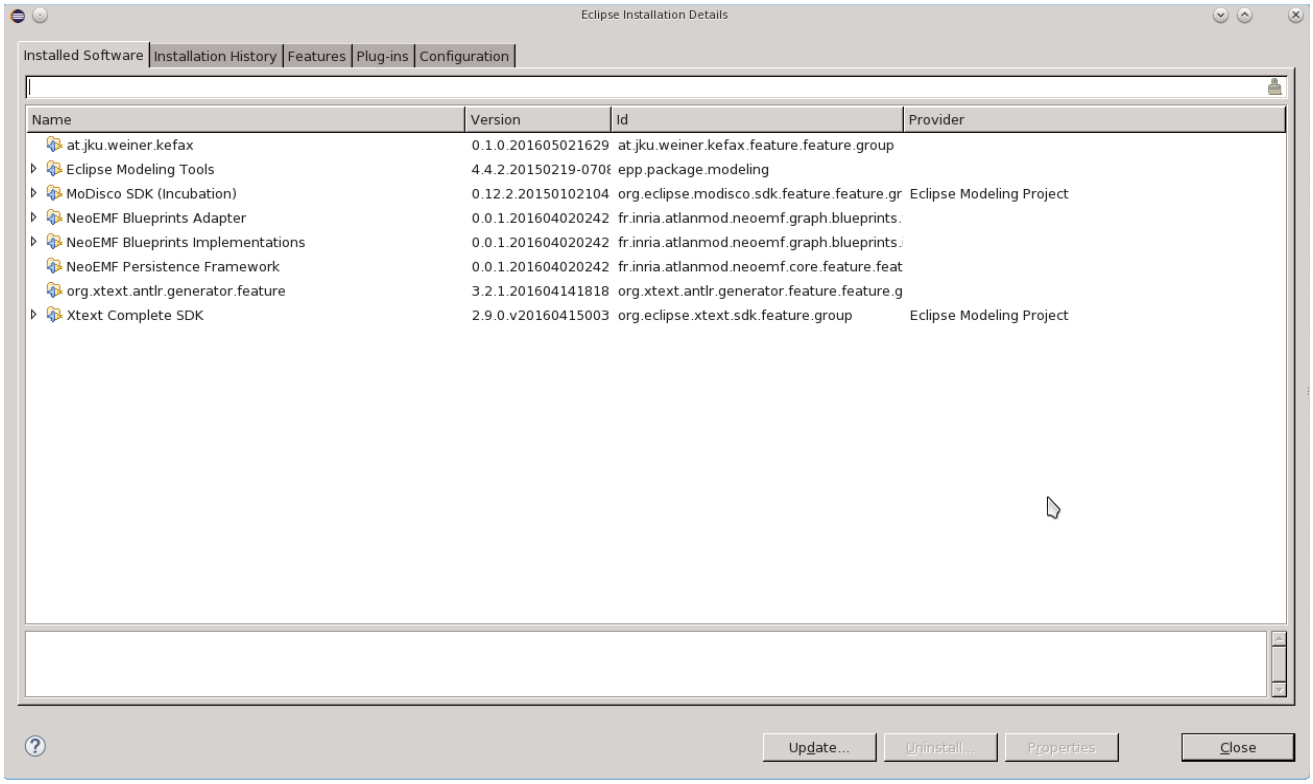

Fig. 1. Installation details

9) Your installed software (Help→About Eclipse→Installation Details) should now look similiar to the screenshot shown in figure 1:
10) Edit the eclipse.ini file. It should contain the following configuration:

Listing 1. part of the eclipse.ini

```
--launcher.XXMaxPermSize
512m
--launcher.defaultAction
openFile
--launcher.appendVmargs
-vmargs
-Dosgi.requiredJavaVersion=1.7
-XX:MaxPermSize=512m
-Xms512m
-Xmx2800m
```

11) Restart Eclipse
12) Run by selecting menu items from *KeFaX* menu (shown in figure 2) either
    - KeFax →Run KeFax demonstration A
    - or KeFax →Run KeFax demonstration B

This will now download the Linux source code with git, generating a minimal working configuration file, execute a kernel compilation to obtain the compile options for all source files, generate a *features.txt* file in the destination project *kefax-linux-working*, copy the minimal required source and header

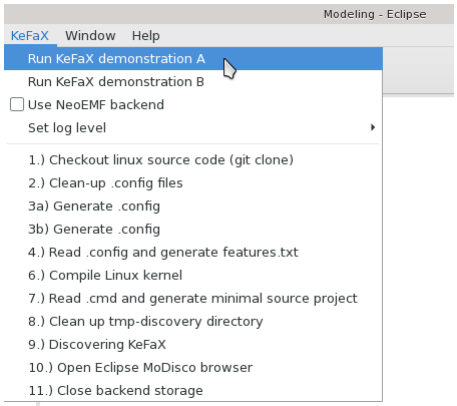

Fig. 2. Using KeFaX after installation

files to the *kefax-linux-working project* and start discovering the *kefax-linux-working project*. Once pre-processing and parsing is done, KeFax will open the *MoDisco EMF browser* which shows the reverse-engineered Linux kernel C source model.

Demonstration mode A and demonstration mode B just differ by one feature: B has *CONFIG_UNIX98_PTYS* set to yes will while is not set for A at all. This is either done in step *3a) Generate .config* or in step *3b) Generate .config* of the *KeFaX* menu.

The resulting EMF model file(s) can than be found in the folder *tmp-discover* of the *kefax-linux-working* project.

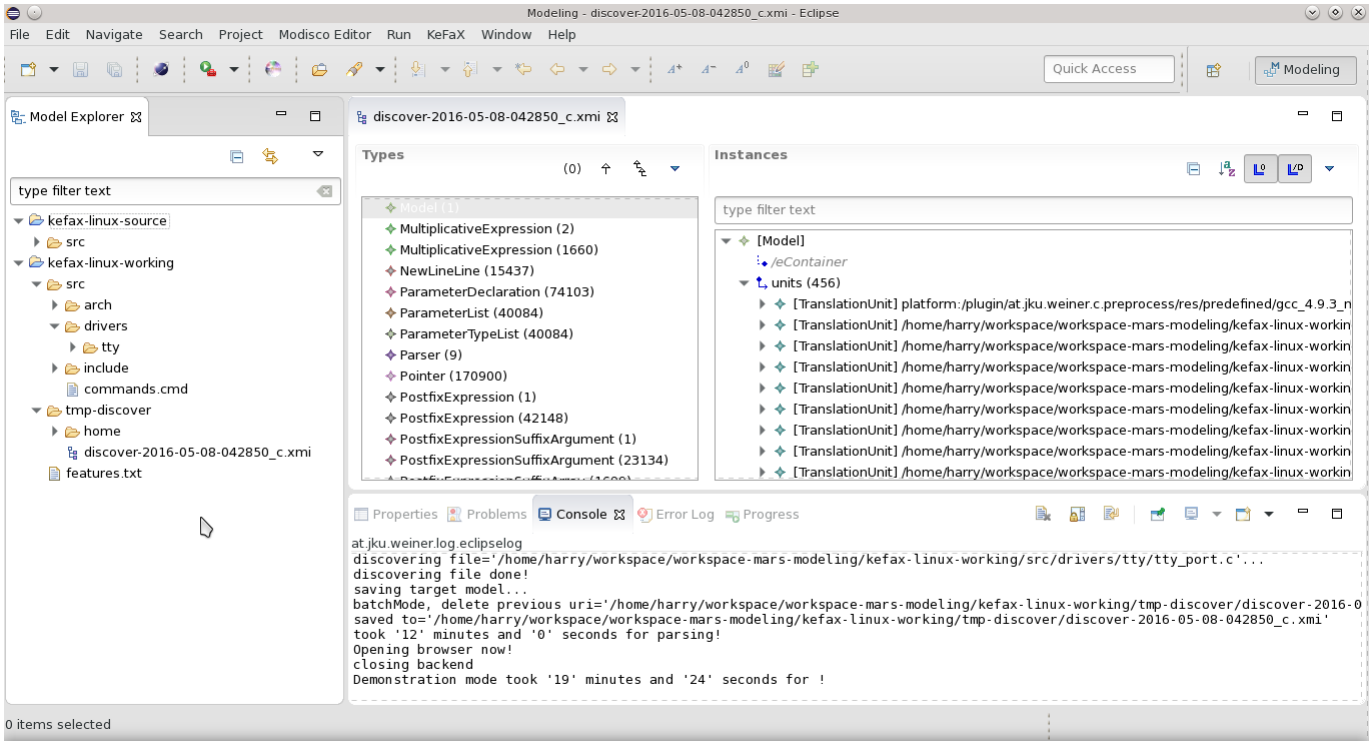

Fig. 3. Result after running KeFaX

You might also adjust the log-level or run the individual commands step-wise to see what they are doing in detail.

Warning: Do not run with log-level set to *trace*. Log-level *trace* is only meant to be used for debugging very nasty bugs (e.g., preprocessor macro expansion). It will take almost forever to execute the preprocessor due to printing the detailed log to the console. Use at your own risk... You have been warned ;-).

B. How-To develop

The whole project is distributed under Eclipse Public License - v 1.0 (see http://www.eclipse.org/legal/epl-v10.html[12]) unless otherwise stated. The source code can be obtained with git from https://github.com/timeraider4u/kefax[13]

1) `git clone https://github.com/timeraider4u/kefax`
2) Execute steps 2 – 7 from How-To use II-A
3) Add https://timeraider4u.github.io/kefax/[11] as an update site, just like in step 8 of How-To use, but instead of installing *kefax* select *at.jku.weiner.xtexttest/at.jku.weiner.xtexttest version 0.1.0.201605080110*
4) Restart Eclipse
5) Import project into workspace (File $rightarrow$ Import $rightarrow$ General $rightarrow$ Existing projects into workspace) and select the workspace folder inside the local *kefax* git repository as the root directory. Select all projects and start the import process.
6) 6. If there are any errors/failures shown after importing you may try to execute Project $rightarrow$ Clean $rightarrow$ Clean all projects. This will remove temporary Xtext/Xtend files and enforce a global rebuild.
7) The code structuring will be explained later in this paper.
8) KeFaX uses Maven (and Github Travis) for continuous integration: A local Maven 3.0 build can be started by navigating to the local kefax git repository root and executing
9) `mvn clean install`

   . This will also execute all JUnit tests.
10) Feel free to start a pull request or report an issue on the Github page [13].
    - The *master* branch is used for development
    - The *gh-pages* branch is used to store the Eclipse update site.
11) Also take a look at the *README.md* file and execute git pull from time to time to keep in touch with the latest changes.

III. Linux kernel configuration/build

The The Linux kernel is developed by programmers from all around the world and can be obtained at https://www.kernel.org/[14]. A mirrored version of the Linux kernel can be found at https://github.com/torvalds/linux[15]. The kernel is developed, compiled and installed by using Unix-like tools.

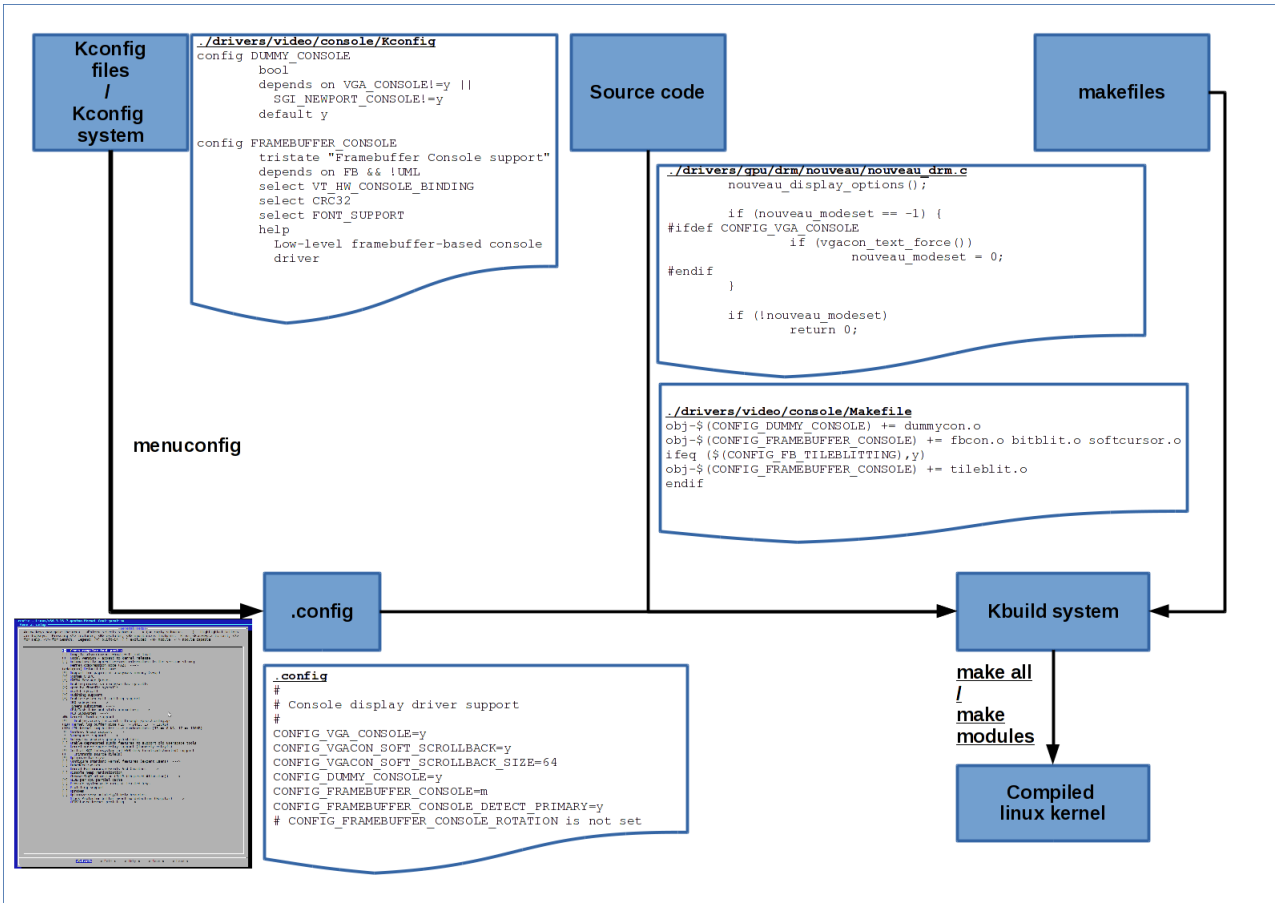

Fig. 4. Overview over the Linux kernel configuration and build process

The main documentation is provided as plain text files while the kernel itself is programmed in the C programming language. Although the kernel itself is developed in C and Assembler, some of the included helper tools are written in C++, in Bash shell scripts, or even in Python (e.g., see *tools/perf/python/*) or in Perl (e.g., see *tools/perf/perl*). The Linux kernel also uses its own ecosystem of "programming languages" for configuration and building of the binary objects which are somehow "domain-specific languages", which have historically grown over time.

The picture 4 shows an overview of the dependencies between the different "DSLs". An overview of the kernel compilation and building can also be found at http://www.linuxjournal.com/article/6568[16]

A. .config

First, the Kconfig system specifies which features depend on each other or can not be combined together. Therefore, kconfig serves as some kind of non-formalized feature model. Here is an example:

Listing 2. ./drivers/video/console/Kconfig

```
config DUMMY_CONSOLE
        bool
        depends on VGA_CONSOLE!=y ||
          SGI_NEWPORT_CONSOLE!=y
        default y

config FRAMEBUFFER_CONSOLE
        tristate "Framebuffer Console
            support"
        depends on FB && !UML
        select VT_HW_CONSOLE_BINDING
        select CRC32
        select FONT_SUPPORT
        help
          Low-level framebuffer-based
              console driver
```

A detailed description of the kconfig system can be found at https://www.kernel.org/doc/Documentation/kbuild/kconfig-language.txt[17]

The variability model of the Linux kernel is, e.g., described in She et al. "The Variability Model of The Linux Kernel"[18], in Sincero et al. "The Linux Kernel Configurator as a Feature Modeling Tool"[19] and in Tartler et al. "Dead or alive: finding zombie features in the linux kernel "[20].

Although these papers describe older kernel versions their main conclusions are still valid.

B. Generating a .config file

The users can then use one of the configuration tools which get delivered with the Linux kernel, e.g. *make oldconfig* when you already have a running kernel and you want to only update to a new kernel version or *make menuconfig*, etc. These applications use the kconfig files as input to determine which additional features must be selected or hide options which are not available due to conflicts. The configuration is then saved into a *.config* file at the root directory of the Linux kernel source code.

The following text listing is an example for such a generated configuration file:

Listing 3. .config file snippet

```
#
# Console display driver support
#
CONFIG_VGA_CONSOLE=y
CONFIG_VGACON_SOFT_SCROLLBACK=y
CONFIG_VGACON_SOFT_SCROLLBACK_SIZE=64
CONFIG_DUMMY_CONSOLE=y
CONFIG_FRAMEBUFFER_CONSOLE=m
CONFIG_FRAMEBUFFER_CONSOLE_DETECT_PRIMARY
    =y
# CONFIG_FRAMEBUFFER_CONSOLE_ROTATION is
    not set
```

The following screenshot in figure 5 shows the *menuconfig* program, a graphical configuration tool which can be used on the terminal:

C. Kbuild system

Whenever the kernel, its modules or any initramfs should be built the kbuild system is invoked. Some valid targets for building are

- make all
- make kernel
- make modules
- make vmlinux
- make initramfs

The makefile in the root directory of the linux source code then invokes the kbuild which generates / collects all necessary makefiles and processes them step-wise. Most of them just contain plain GNU make instructions but some contain additional bash script magic.

An example kbuild makefile is the following:

Listing 4. Snippet from ./drivers/video/console/Makefile

```
obj-$(CONFIG_DUMMY_CONSOLE) += dummycon.o
obj-$(CONFIG_FRAMEBUFFER_CONSOLE) +=
    fbcon.o bitblit.o softcursor.o
ifeq ($(CONFIG_FB_TILEBLITTING),y)
obj-$(CONFIG_FRAMEBUFFER_CONSOLE) +=
    tileblit.o
endif
```

The commands in the makefile then instruct the C compiler which source files to compile and which command line options to use. The selected features of the *.config* file are provided as pre-processor macro definitions. But also include directories are enlisted.

The following source snippet shows how the macro definitions are used in the C source code:

Listing 5. Snippet from ./drivers/gpu/drm/nouveau/nouveau_drm.c

```
        nouveau_display_options();
        if (nouveau_modeset == -1) {
#ifdef CONFIG_VGA_CONSOLE
                if (vgacon_text_force())
                        nouveau_modeset =
                            0;
#endif
        }
        if (!nouveau_modeset)
                return 0;
```

The results of the compilation are binary object files which can be installed to */boot* and/or */lib/modules/<linux-version>* with *make install*.

More information about kbuild itself can be found at https://www.kernel.org/doc/Documentation/kbuild/makefiles.txt[21] and http://hemaprathaban.blogspot.co.at/2013/06/makefile-and-kconfig.html[22]

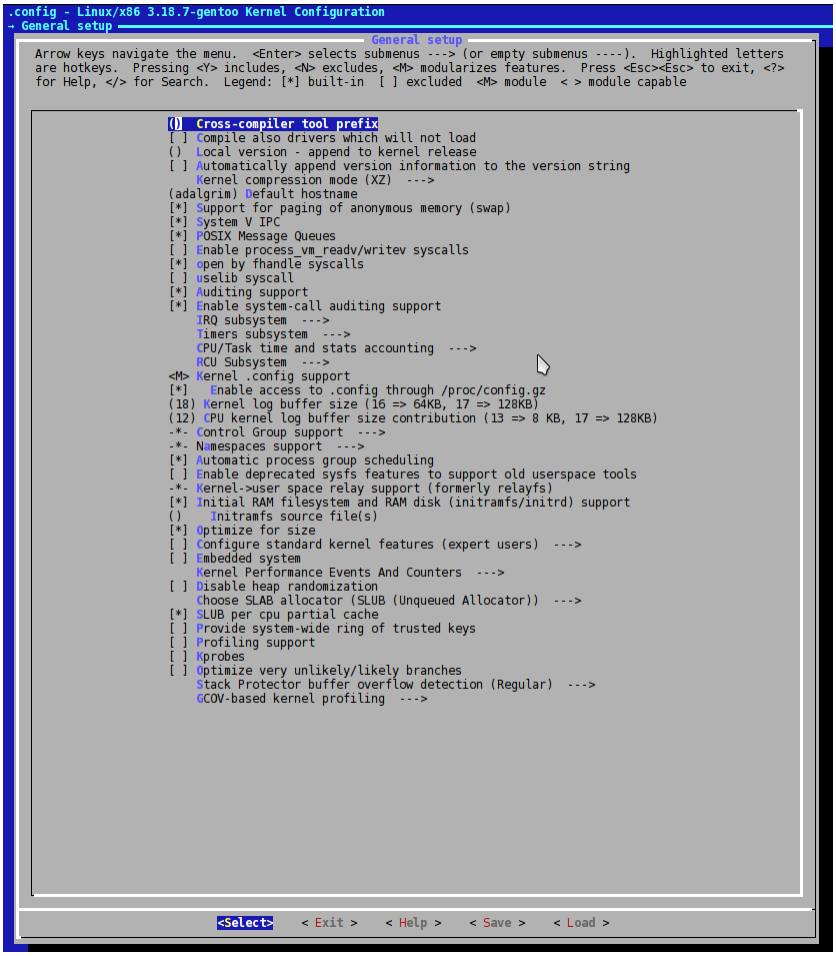

Fig. 5. Configuring the kernel with menuconfig, e.g., over a ssh connection

IV. Required code structure for ECCO

Ecco requires a structure similiar to the file tree shown in figure 6 to be able to parse the product

The *feature.txt* should list all enabled features, one per line starting with a unique name separated with a semicolon from the feature's description and a new-line character at the end. The file structure should only contain the pruned source code (so no folders / source files for disabled features). The source files should be parsed by some parser and be presented to *ECCO* as a tree data structure.

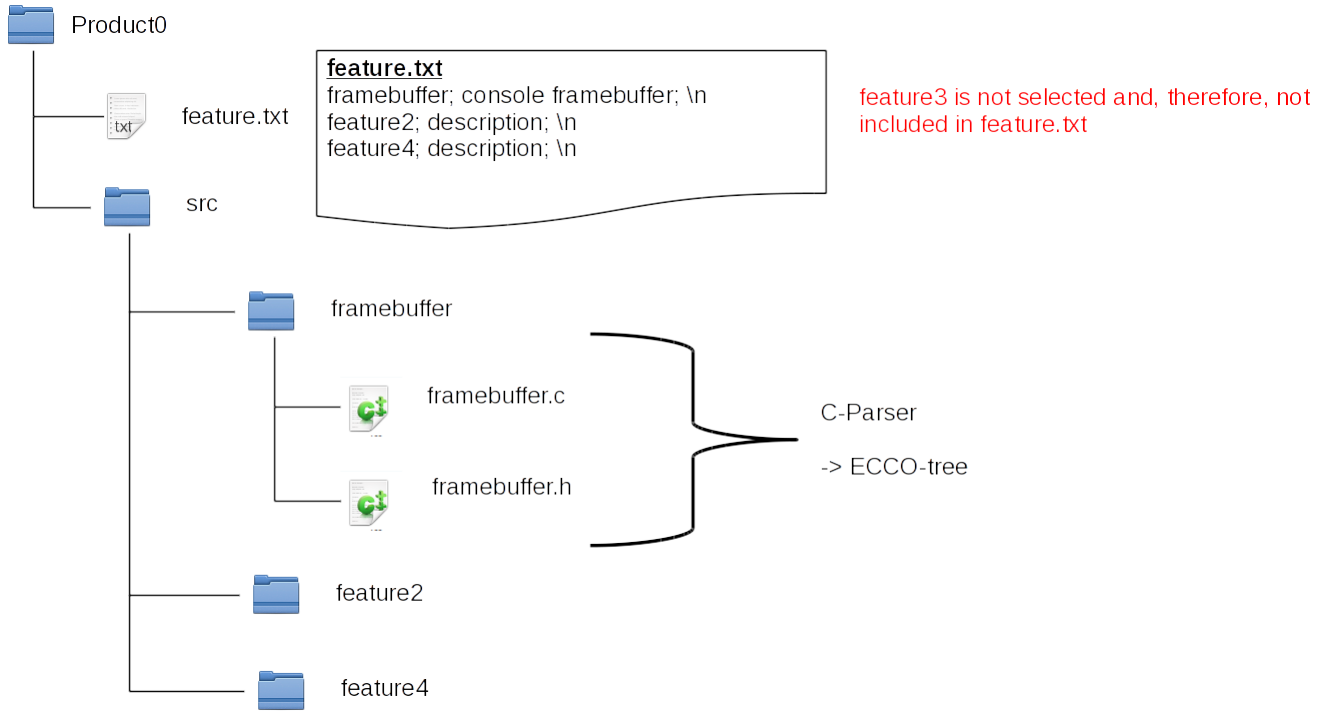

Fig. 6. Overview of required code structure

V. KeFaX requirements / KeFaX tasks

As a result of the required input structure of *ECCO*, the following coarse tasks can be identified for this project:

1) Parsing .config file
2) Obtaining compilation options (which source files are required to be parsed, which include directories are used)
3) Pre-processing source code and removing un-taken pre-processor conditionals
4) Parsing source code (.c / .h files)
5) "Glue code" + (Complete data model)
6) Providing the data structure somehow to *ECCO*

These steps can then be re-defined as:

1) Manually create *.config* by selecting features (e.g., in *menuconfig*)
2) Parse *.config*
3) Create *features.txt* (*.config* → *features.txt*)
4) Obtain compilation options from kbuild makefiles
5) Prune source code files (code files for features that are not selected should be deleted) (*.config* + makefiles → code structure)
6) Run pre-processor on files and do fine-grain (*# ifdef* code which is not selected should be thrown away) (*.config* → source code files)

References

[1] S. Fischer, L. Linsbauer, R. E. Lopez-Herrejon, and A. Egyed, "Enhancing clone-and-own with systematic reuse for developing software variants," in *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*. IEEE, 2014, pp. 391–400.

[2] "Institute for software systems engineering," http://www.jku.at/isse/content, last visited: 2016-05-09. [Online]. Available: http://www.jku.at/isse/content

[3] "Ecco tool website," http://www.isse.jku.at/tools/ecco, last visited: 2016-05-09.

[4] "Eclipse modeling download," https://eclipse.org/downloads/, last visited: 2016-05-09. [Online]. Available: https://eclipse.org/downloads/

[5] "Eclipse modeling download luna," https://eclipse.org/downloads/packages/release/luna/sr2, last visited: 2016-05-09. [Online]. Available: https://eclipse.org/downloads/packages/release/luna/sr2

[6] "Modeling package updates for eclipse mars," http://www.eclipse.org/modeling/amalgam/downloads/package/modeling/mars/, last visited: 2016-05-09. [Online]. Available: http://www.eclipse.org/modeling/amalgam/downloads/package/modeling/mars/

[7] "Modeling package updates for eclipse luna," http://www.eclipse.org/modeling/amalgam/downloads/package/modeling/luna/, last visited: 2016-05-09. [Online]. Available: http://www.eclipse.org/modeling/amalgam/downloads/package/modeling/luna/

[8] "Neoemf update site," https://timeraider4u.github.io/NeoEMF/, last visited: 2016-05-09. [Online]. Available: https://timeraider4u.github.io/NeoEMF/

[9] "org.xtext.antlr.generator update site," https://timeraider4u.github.io/org.xtext.antlr.generator/, last visited: 2016-05-09. [Online]. Available: https://timeraider4u.github.io/org.xtext.antlr.generator/

[10] "Modified xtext update site," https://timeraider4u.github.io/xtext/, last visited: 2016-05-09. [Online]. Available: https://timeraider4u.github.io/xtext/

[11] "Kefax update site," https://timeraider4u.github.io/kefax/, last visited: 2016-05-09. [Online]. Available: https://timeraider4u.github.io/kefax/

[12] "Eclipse public license text," http://www.eclipse.org/legal/epl-v10.html, last visited: 2016-05-09. [Online]. Available: http://www.eclipse.org/legal/epl-v10.html

[13] "Kefax source code," https://github.com/timeraider4u/kefax, last visited: 2016-05-09. [Online]. Available: https://github.com/timeraider4u/kefax

[14] "Linux kernel," https://www.kernel.org/, last visited: 2016-05-09. [Online]. Available: https://www.kernel.org/

[15] "Linux kernel (github mirror)," https://github.com/torvalds/linux, last visited: 2016-05-09. [Online]. Available: https://github.com/torvalds/linux

[16] G. Kroah-Hartman, "The kernel configuration and build process," http://www.linuxjournal.com/article/6568, 2003, last visited: 2016-05-09. [Online]. Available: http://www.linuxjournal.com/article/6568

[17] "Kconfig," https://www.kernel.org/doc/Documentation/kbuild/kconfig-language.txt, last visited: 2016-05-09. [Online]. Available: https://www.kernel.org/doc/Documentation/kbuild/kconfig-language.txt

[18] S. She, R. Lotufo, T. Berger, A. Wasowski, and K. Czarnecki, "The variability model of the linux kernel." *VaMoS*, vol. 10, pp. 45–51, 2010. [Online]. Available: http://gsd.uwaterloo.ca/sites/default/files/camera-vamos-20100107.pdf

[19] J. Sincero and W. Schröder-Preikschat, "The linux kernel configurator as a feature modeling tool." in *SPLC (2)*. Citeseer, 2008, pp. 257–260. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.157.2318&rep=rep1&type=pdf

[20] R. Tartler, J. Sincero, W. Schröder-Preikschat, and D. Lohmann, "Dead or alive: finding zombie features in the linux kernel," in *Proceedings of the First International Workshop on Feature-Oriented Software Development*. ACM, 2009, pp. 81–86. [Online]. Available: http://www.infosun.fim.uni-passau.de/spl/apel/FOSD2009/FOSD2009_Printed_Proceedings.pdf#page=89

[21] "Kbuild makefiles," https://www.kernel.org/doc/Documentation/kbuild/makefiles.txt, last visited: 2016-05-09. [Online]. Available: https://www.kernel.org/doc/Documentation/kbuild/makefiles.txt

[22] H. .p, "Makefile and kconfig," http://hemaprathaban.blogspot.co.at/2013/06/makefile-and-kconfig.html, 2013, last visited: 2016-05-09. [Online]. Available: http://hemaprathaban.blogspot.co.at/2013/06/makefile-and-kconfig.html