Conan At Scale

A Build Tool-Chain across multiple teams

SIEMENS

Introduction

SIEMENS

Context

SIEMENS

Benoît Bleuzé

- Background:
  - Medical imaging/computational geometry background
  - architecture of C++ Graphical user applications as well as automated processes
  - mostly interested in tooling: remove manual tasks, toil, improve quality through repeatability, automation
- Software Architect, at Siemens Mobility since 2018, working on autonomous trains
  - Lead of the Software Engineering team: **SWEn** in the **Assisted and Driverless Train Operation** project, more precisely in the **Obstacle Detection** domain.

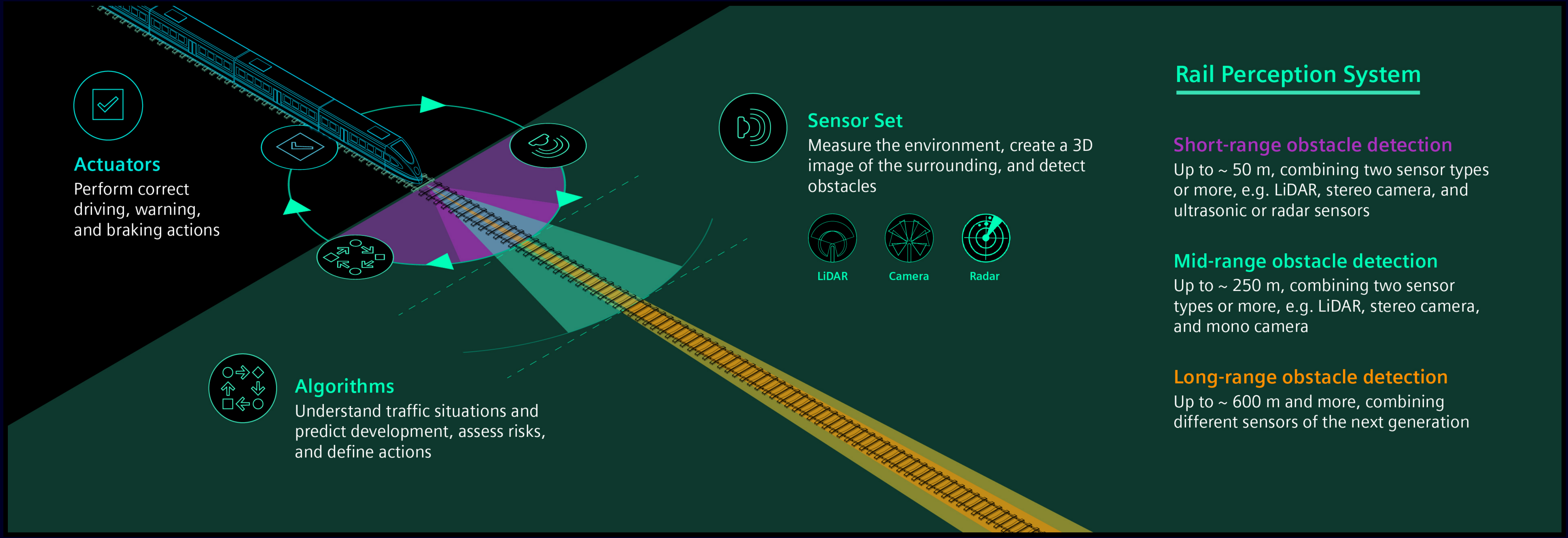
SIEMENS

SWEn

- 5-people team
- Improve development **quality**
- Create **tools** serving quality and speed
- Assemble and code OS and middleware components for the **Obstacle Detection** project

SIEMENS

Assisted And Driverless Train Operation



Rail Perception System

Short-range obstacle detection
Up to ~ 50 m, combining two sensor types or more, e.g. LiDAR, stereo camera, and ultrasonic or radar sensors

Mid-range obstacle detection
Up to ~ 250 m, combining two sensor types or more, e.g. LiDAR, stereo camera, and mono camera

Long-range obstacle detection
Up to ~ 600 m and more, combining different sensors of the next generation

Actuators
Perform correct driving, warning, and braking actions

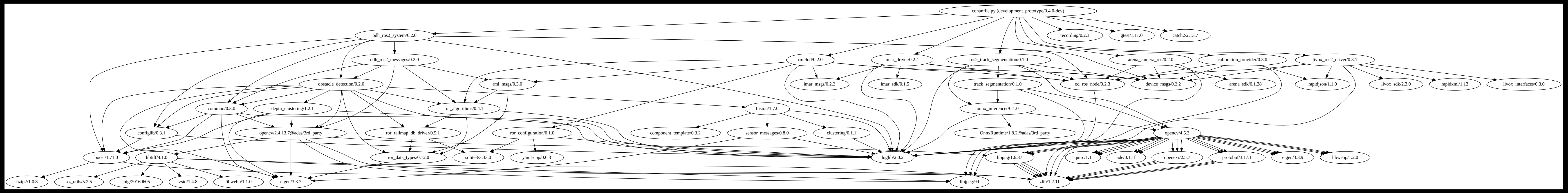
Sensor Set
Measure the environment, create a 3D image of the surrounding, and detect obstacles

Algorithms
Understand traffic situations and predict development, assess risks, and define actions

Assisted and Driverless Train Operation

SIEMENS

Dependency Graph



Communication Between Packages

- **Developer**: Sharing packages, versions of a package, integrate them together
- **Linker**: Loading symbols from a library, export symbols to other libraries

Enter Conan: C/C++ Package Manager

- **Resolves dependencies** across numerous and perhaps conflicting packages
- Handles **configuration**, **compilation**, **installation/deployment** of packages
- **stores** on a remote server **recipes** and **binary flavours** of packages

https://conan.io/

This talk is NOT:

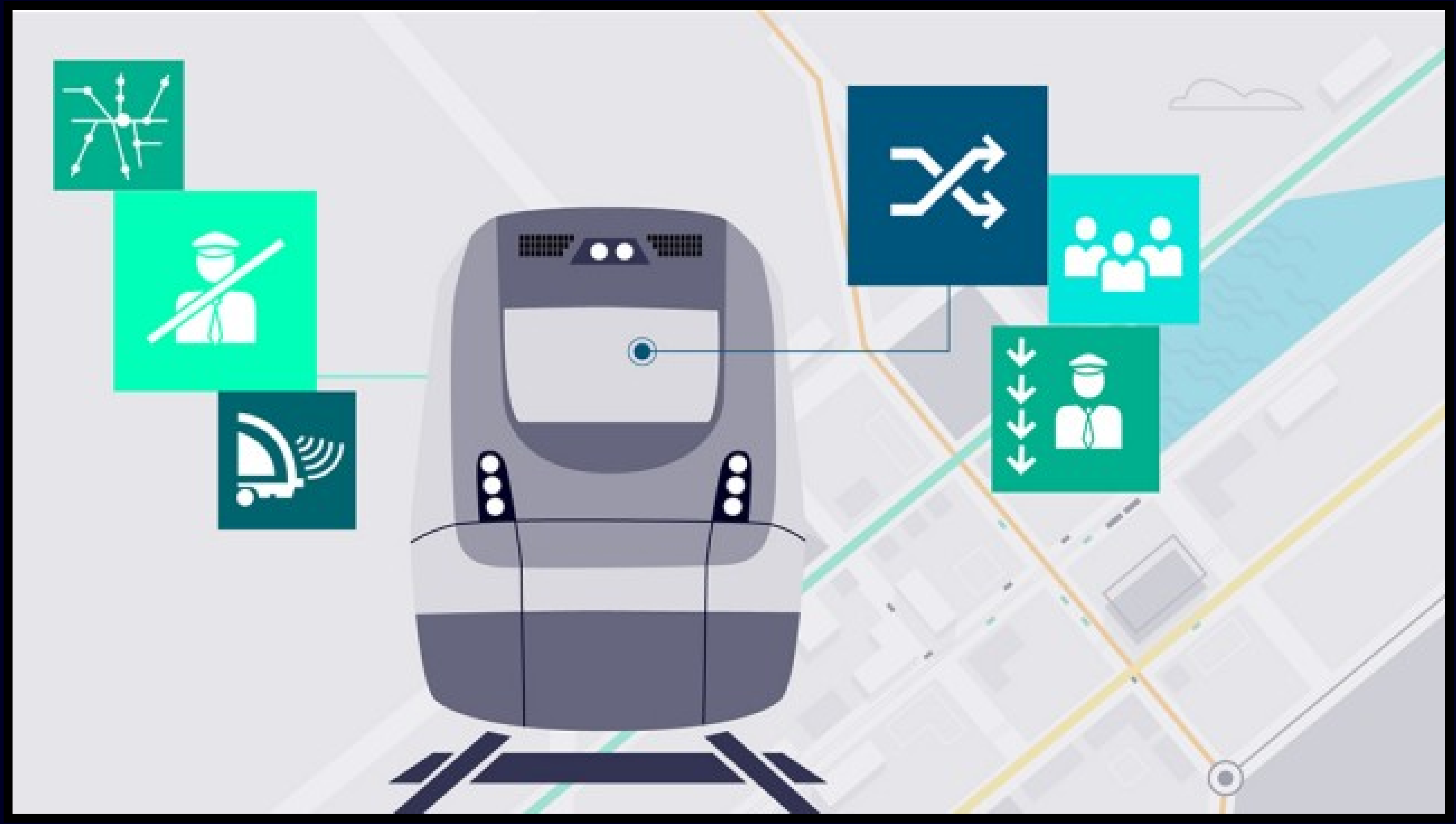
- a conan tutorial
- an how-to get a build system (generator) and conan to talk together

But this talk is:

- showing how to bring a multi-team, multi-package project to exchange code effectively
- giving tips on how Conan can help with this.

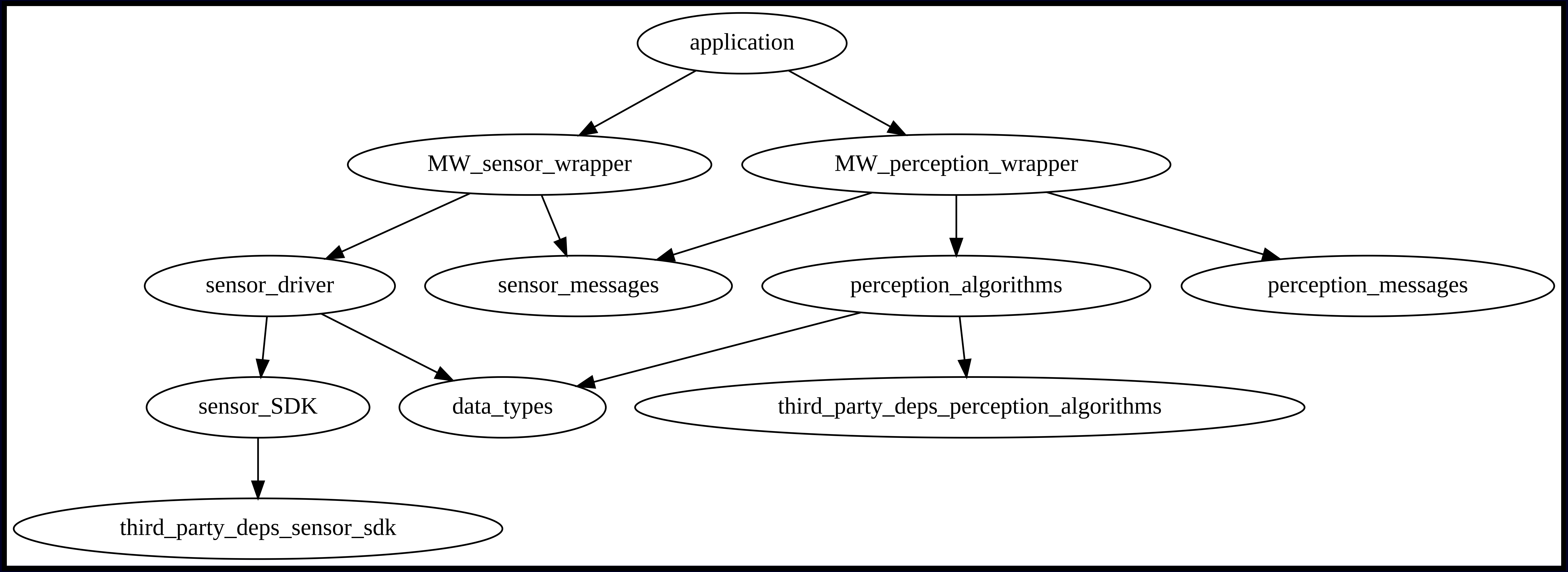
SIEMENS

Obstacle Detection Development Pipeline



SIEMENS

Component Hierarchy



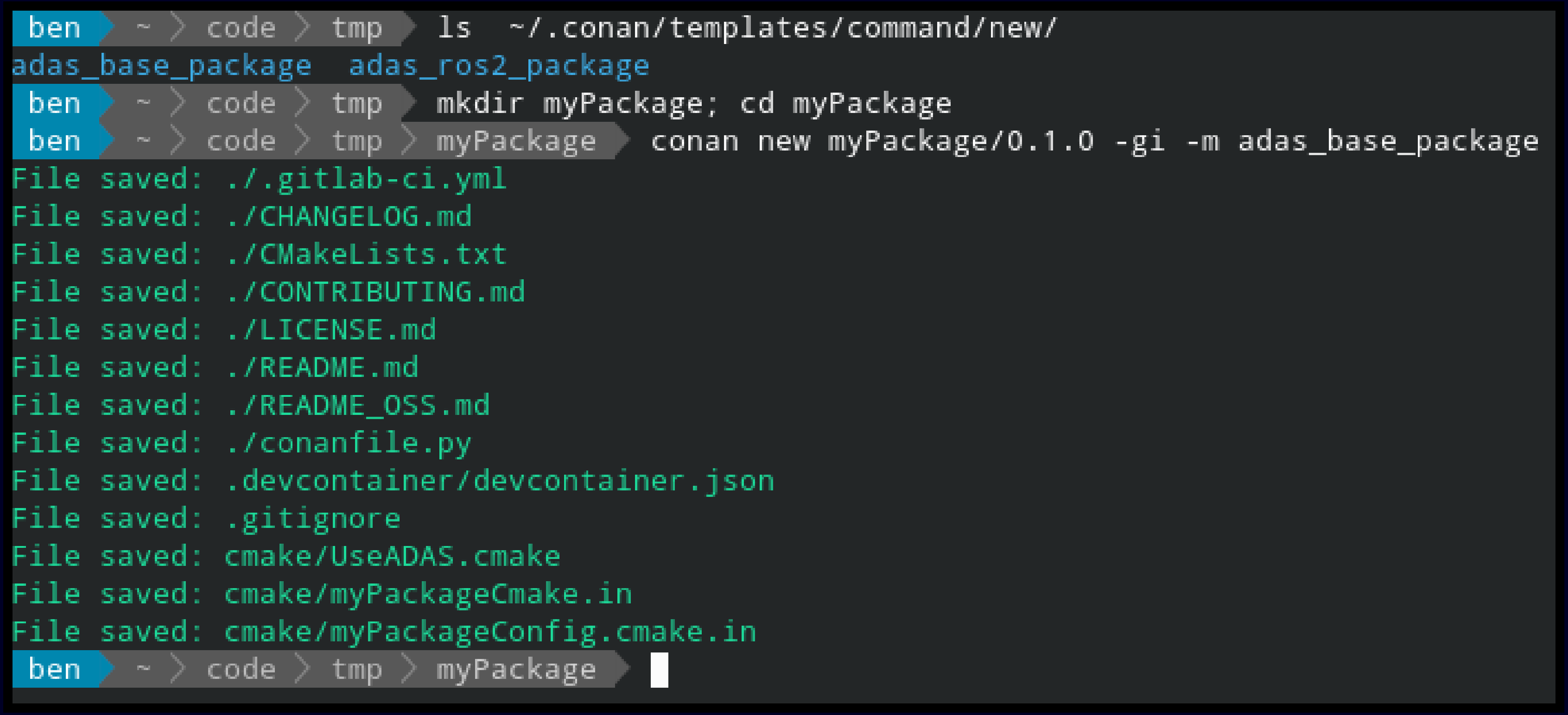
SIEMENS

Create A Project

***Tip for project creation** : Always use templates!*
All projects look familiar to any engineer from any team, and conventions are observed if mandatory files are needed.

SIEMENS

Conan Template



```
ben  ~  code  tmp  ls  ~/.conan/templates/command/new/
adas_base_package  adas_ros2_package
ben  ~  code  tmp  mkdir myPackage; cd myPackage
ben  ~  code  tmp  myPackage  conan new myPackage/0.1.0 -gi -m adas_base_package
File saved: ./.gitlab-ci.yml
File saved: ./CHANGELOG.md
File saved: ./CMakeLists.txt
File saved: ./CONTRIBUTING.md
File saved: ./LICENSE.md
File saved: ./README.md
File saved: ./README_OSS.md
File saved: ./conanfile.py
File saved: .devcontainer/devcontainer.json
File saved: .gitignore
File saved: cmake/UseADAS.cmake
File saved: cmake/myPackageCmake.in
File saved: cmake/myPackageConfig.cmake.in
ben  ~  code  tmp  myPackage 
```

Template file:

```
cmake_minimum_required(VERSION 3.18)
project({{ name }}
  LANGUAGES CXX
  VERSION {{ version }}
  DESCRIPTION "{{ name }}"
  HOMEPAGE_URL "https://code.siemens.com/ADAS4Rail"
)
```

Resulting file:

```
cmake_minimum_required(VERSION 3.18)
project(myPackage
  LANGUAGES CXX
  VERSION 0.1.0
  DESCRIPTION "myPackage"
  HOMEPAGE_URL "https://code.siemens.com/ADAS4Rail"
)
```

SIEMENS

Configuration

***Tip for project toolchain settings**: Control your compilation toolchain! Support and identify toolchain's clearly to control your binaries, and help developers with reporting issues.*

Conan Profiles

Harmonise development environment, runtime platform.

```
include(boost)

[build_requires]
[settings]
os=Linux
os_build=Linux
arch=x86_64
arch_build=x86_64
compiler=gcc
compiler.version=9
compiler.libcxx=libstdc++11
compiler.cppstd=17
build_type=Release
[options]
[env]
CC=gcc-9
CXX=g++-9
```

Example here: adas-gcc-9, but al so in our list:

- adas-gcc[9-11][-debug]
- adas-clang[10-14][-debug]

SIEMENS

```
ben ~ % conan profile list
adas-clang-10
adas-clang-10-debug
adas-clang-11
adas-clang-11-debug
adas-clang-12
adas-clang-12-debug
adas-clang-13
adas-clang-13-debug
adas-clang-14
adas-clang-14-debug
adas-gcc-10
adas-gcc-10-debug
adas-gcc-11
adas-gcc-11-debug
adas-gcc-7
adas-gcc-7-debug
adas-gcc-8
adas-gcc-8-debug
adas-gcc-9
adas-gcc-9-debug
boost
default
```

***Tip for managing developer environment**: upstream the configuration as much as possible, provide containers*

Give users easy access to conan settings, remotes, hooks and profiles:

```
# install version 0.7.2 from the repository
conan config install --type git -sf config git@code.siemens.com:ADAS4Rail/SWEn/conan/config.git --args="-b 0.7.2"
```

SIEMENS

Dependencies

SIEMENS

Declaring Dependencies

```python
from conans import ConanFile

class LoglibConan(ConanFile):
    # provide the base recipes
    python_requires = 'adas_recipe/0.8.4'
    python_requires_extend = 'adas_recipe.ADASConanFile'

    requires = 'spdlog/1.5.0'
```

Declaring Dependencies

```python
from conans import ConanFile

class LoglibConan(ConanFile):
    # provide the base recipes
    python_requires = 'adas_recipe/0.8.4'
    python_requires_extend = 'adas_recipe.ADASConanFile'

    requires = 'spdlog/1.5.0'
```

SIEMENS

Declaring Dependencies

```python
from conans import ConanFile

class LoglibConan(ConanFile):
    # provide the base recipes
    python_requires = 'adas_recipe/0.8.4'
    python_requires_extend = 'adas_recipe.ADASConanFile'

    requires = 'spdlog/1.5.0'
```

SIEMENS

Versioning

Conan package entities:

- recipe revision `RREV`: used from the recipe's content
  - use `scm` as a `revision_mode`*
- package ID, a combination of:
  - platform information, architecture, compiler, build type: e.g `Linux/GCC5/Debug`
  - configuration options (e.g. optional build features)
- binary package revision `PREV`: hash of the installed files

* only if package are created from SCM commits.

SIEMENS

Version Use Cases

- Fixed releases, pinned versions of binary packages
- Continuous integration of release trains, using Semantic versioning: always use the latest compatible major versions
- Experimental code shared between projects, non released, pinned or not

Tip for versioning*: Provide a blend of released version, moving aliases, and temporary non released versions, delete them after some time.*

SIEMENS

Package version	Branch	Postfix	Example	Alias
Release	`release`	-	`0.1.0`	-
Release candidate	`rc/`, `release/`	`rc.<CI_PIPELINE_IID>+<CI_COMMIT_SHORT_SHA>`	`0.2.0-rc.12+abcd123`	`0.2.0-rc`
Stable development versions	`<CI_DEFAULT_BRANCH>`	`<CI_DEFAULT_BRANCH>.<CI_PIPELINE_IID>+<CI_COMMIT_SHORT_SHA>`	`0.2.0-main.10+abcd123`	`0.2.0-main`
Feature development versions	Feature	`feat.<CI_PIPELINE_IID>+<CI_COMMIT_SHORT_SHA>`	`0.2.0-feat.1+abcd123`	`0.2.0-feat-SWEN-543-better-world`
Hotfix versions	Hotfix	`hotfix.<CI_PIPELINE_IID>+<CI_COMMIT_SHORT_SHA>`	`0.2.0-hotfix.15+abcd123`	`0.2.0-SWEN-234-fixCompilation`

SIEMENS

Build

SIEMENS

Control Binaries

- Make sure a binary is what you expect:
  - Have strict control over the ABI by using strict **semantic versioning**
  - Make sure the conan **package_id** reflect building options and building environment.

SIEMENS

Custom package definition of what settings count for a **package ID**:

```python
class ADASConanFile(ConanFile):
    """Basic conan recipe containing default settings and functions."""
    license = 'Siemens Inner Source 1.3'

    settings = 'os', 'compiler', 'build_type', 'arch'

    def package_id(self):
        """Remove options to not influence the generated package_id."""
        logger.debug('Generate the package id.')
        distribution = distro.LinuxDistribution()
        self.info.settings.os.distribution = f'{distribution.id()}{distribution.version()}'
        del self.info.options.acf_enable_pclp
        del self.info.options.acf_enable_testing
        del self.info.options.acf_enable_doc
        del self.info.options.acf_coverage_threshold
```

SIEMENS

Custom package definition of what settings count for a **package ID**:

```python
class ADASConanFile(ConanFile):
    """Basic conan recipe containing default settings and functions."""
    license = 'Siemens Inner Source 1.3'

    settings = 'os', 'compiler', 'build_type', 'arch'

    def package_id(self):
        """Remove options to not influence the generated package_id."""
        logger.debug('Generate the package id.')
        distribution = distro.LinuxDistribution()
        self.info.settings.os.distribution = f'{distribution.id()}{distribution.version()}'
        del self.info.options.acf_enable_pclp
        del self.info.options.acf_enable_testing
        del self.info.options.acf_enable_doc
        del self.info.options.acf_coverage_threshold
```

SIEMENS

Custom package definition of what settings count for a **package ID**:

```python
class ADASConanFile(ConanFile):
    """Basic conan recipe containing default settings and functions."""
    license = 'Siemens Inner Source 1.3'

    settings = 'os', 'compiler', 'build_type', 'arch'

    def package_id(self):
        """Remove options to not influence the generated package_id."""
        logger.debug('Generate the package id.')
        distribution = distro.LinuxDistribution()
        self.info.settings.os.distribution = f'{distribution.id()}{distribution.version()}'
        del self.info.options.acf_enable_pclp
        del self.info.options.acf_enable_testing
        del self.info.options.acf_enable_doc
        del self.info.options.acf_coverage_threshold
```

SIEMENS

Custom package definition of what settings count for a **package ID**:

```python
class ADASConanFile(ConanFile):
    """Basic conan recipe containing default settings and functions."""
    license = 'Siemens Inner Source 1.3'

    settings = 'os', 'compiler', 'build_type', 'arch'

    def package_id(self):
        """Remove options to not influence the generated package_id."""
        logger.debug('Generate the package id.')
        distribution = distro.LinuxDistribution()
        self.info.settings.os.distribution = f'{distribution.id()}{distribution.version()}'
        del self.info.options.acf_enable_pclp
        del self.info.options.acf_enable_testing
        del self.info.options.acf_enable_doc
        del self.info.options.acf_coverage_threshold
```

SIEMENS

Hooks

***Tip for quality enforcement**: As much as possible code automated tools to enforce quality guidelines at the project level.*

Conan hooks are:

- Python functions called at determined entry points
  - pre/post source, build, package and other steps
- can extend conan functionalities
- very handy to insert quality checks for packages
- stored in conan configuration, shared between users, package-independent

SIEMENS

Hooks Examples

- **license checker**: `pre_export` does the package contain a valid license?
- **naming convention checker**: `pre_export` check for forbidden characters, typographical rules (dashes may be forbidden for instance)
  - note : take advantage of CI env variables if run from project repo: project name can be checked against package name
- **version checker**: `pre_export`
  - check for semantic versioning compliance semver python lib
  - check dependencies only use released versions if current build is on stable branch or release (only on CI run of the hook)
  - check Changelog entries and version match
  - check version on tag: if package creation runs on CI due to a tag event: they must be the same
  - check version update: the current version should be higher than the last published tag in parent commits
- the sky is the limit…

SIEMENS

Wrapping Up

SIEMENS

Left-Overs

- Not used in the team
  - Conan workspaces: experimental feature, might help when working locally on multiple packages
  - Conan 2.0: working on it...
- Used in the team, but out of scope today
  - ROS2 integration with colcon: rich topic
  - conan generators: generate custom manifests, custom package formats (apt, etc...)
  - Shared CMake "*libraries*" interfacing with conan
  - virtual environment
  - upstream CI scripts to generate, test upload recipes and all needed binary flavours.

Tips

- **Project creation**: Always use templates!
- **Project toolchain setting**: Control your compilation toolchain!
- **Managing developer environment**: upstream as much as possible toolchain configuration, provide container.
- **Versioning**: Provide a blend of released version, moving aliases and temporary non released version.
- **Quality enforcement**: As much as possible code automated tools to enforce quality guidelines at the project level.

SIEMENS

Lessons Learned

- Treat infrastructure as code, CI as code, build system generators and package managers as code:
  - WRITE TESTS for them
  - Even then, if working with conan, brace yourself for bug reports
- System wide installation of package binaries is a thing of the past, all languages embraced project based dependencies, C++ can too
- Balancing what configuration goes into CMake, and what goes into Conan is still a mystery to us, conan 2.0 only brings more questions

SIEMENS

Thanks

- First and above all the SWEn team members, **past and present**, who did most of the work
- The code.siemens.com team for their use of reveal-md, and their stylesheet, that I borrowed for the occasion

SIEMENS

Contact

Published by Siemens Mobility GmbH

Benoît Bleuzé, Software Architect
SMO SDT TEC SPA CVG

Rudower Chaussee 29
12489 Berlin, Deutschland

Email: benoit.bleuze@siemens.com

SIEMENS