Enhancing SSP Creation using sspgen

American Modelica Conference 2022, Dallas, TX

Lars Ivar Hatledal
Associate Professor, NTNU Aalesund, Norway
26.10.2022
Background

• **Co-simulation** is used by NTNU campus Aalesund
  – Mostly to simulate maritime systems

• FMI → Individual models
• SSP → Complete systems
System Structure & Parameterization (SSP)

- The SSP is a tool independent standard to define complete systems consisting of one or more models.
  - including its parameterization that can be transferred between simulation tools.
- A model could be an FMU adhering to the FMI standard.
- An SSP is a zip archive that includes an XML document describing the system, connections and any initial values of the models together with any required data.
Enabling tools for co-simulation

- Lower-level (Individual models)
  - FMI4j/FMU4j (FMI import/export in Java)
  - fmi4cpp/fmu4cpp (FMI import/export in C++)
  - PythonFMU (FMI export in Python)
  - FMU-proxy/proxy-fmu (Distributed FMU access)

sspgen -> Defining simulations

- Higher-level (Orchestration)
  - libcosim (Open-simulation-platform, C++/CLI)
  - Vico (NTNU, Java/CLI)
  - Ecos (NTNU, C++/CLI)
sspgen

- **sspgen** is a Kotlin DSL for easing the creation of SSP 1.0 compatible systems.
  - Creates the SystemStructure.xml.
  - Handles packaging of the SystemStructure.xml and any additional resources (local files, URLs) into a SSP archive (.zip).
  - Checks the system for correctness.

- The DSL is available though Maven and can be referenced in scripts.

- Allows SSP definitions to be distributed as a script.
  - Easy to modify and share.
  - Expressions as initial values!
  - Can reference files from both the file systems and URLs.

[https://github.com/Ecos-platform/sspgen](https://github.com/Ecos-platform/sspgen)
Domain-specific languages (DSLs)

• A DSL is a computer language specialized to a particular application domain.

• Two main classes of DSLs:
  – External
    • LaTeX, CMake++
  – Embedded
    • Gradle++
Kotlin is a modern language known for its readable, clean, and concise syntax.
- Default language for Android development.
- Drop-in replacement for Java.

With its advanced functional programming capabilities, we can create type-safe, statically typed builders that act as DSLs
- which are suitable for expressing complex hierarchical data structures in a semi-declarative way.
Kotlin scripting

• Kotlin code can execute as standalone scripts.
  – Runtime dependency resolution.

```kotlin
@file:Repository("https://maven.pkg.jetbrains.space/public/p/kotlinx-html/")
@file:DependsOn("org.jetbrains.kotlinx:kotlinx-html-jvm:0.7.3")

import kotlinx.html.*
import kotlinx.html.stream.*
import kotlinx.html.attributes.*

val addresssee = "World"

print(
    createHTML().html {
        body {
            h1 { +"Hello, $addresssee!" } }
        }
    )
```

Anatomy of a sspgen script

```plaintext
import no.ntnu.lhb.sspgen.dsl.*

ssp("TestSSPGen") {
    resources {
        file("path/to/FMU1.fmu")
        file("path/to/FMU2.fmu")
        url("example.com/someFile.txt")
    }

    ssd("A simple CLI test") {
        author = "John Doe"
        description = "A simple description"

        system("Test") {
            description = "An even simpler description"

            elements {
                component("FMU1", "resources/FMU1.fmu") {
                    connectors {
                        real("output", output) {
                            unit("m/s")
                        }
                        real("input", input)
                        integer("counter", output)
                    }
                    parameterBindings {
                        parameterSet("InitialValues") {
                            real("input", 2.0)
                            integer("counter", 0)
                        }
                    }
                }
            }

            connections {
                "FMU2.output" to "FMU1.input"
                ("FMU1.output" to "FMU2.input").linearTransformation(factor = 1.5)
            }

            defaultExperiment(startTime = 1.0)
        }
    }
}
```

NTNU | Norwegian University of Science and Technology
OSP-IS

- The OSP interface specification (OSP-IS) is an addition to the FMI standard which provide:
  - A method for adding semantic meaning to model interface variables.
  - A simpler model connection process.
  - Validation of semantically correct simulations.

https://opensimulationplatform.com/specification/
sspgen + OSP-IS = True

• *sspgen* can transpilie compound OSP-IS connections to single scalar connections supported by SSP.
• Additionally, *sspgen* can verify the connections according to the standard (both SSP and OSP-IS).
Other sspgen features

• proxy-fmu support
  – Distributed FMUs
  – https://github.com/open-simulation-platform/proxy-fmu

• PythonFMU integration
  – Build FMUs on demand from Python code
  – https://github.com/NTNU-IHB/PythonFMU

• FMI-VDM-Model integration
  – Optional static analysis of included FMUs
Conclusion

• Utilizing the SSP, simulations can be defined in a standardized way and ssppgen:
  • Makes them easier to create, modify and share.
  • Enables non-trivial parameters to be defined.
  • Enables the OSP-IS to be used in this context, making it usable by a larger audience.
Future work

- Add more tests and general polishing of the code.
- Improve current SSP 1.0 support.
- Support FMI 3.0 and future SSP versions.
Q & A