

A Graphical Domain-Specific Modeling Language for Capella System Capability Diagram

LOG8505E — Model Driven Software Engineering

December 4, 2024

Arthur Milani Giovanini

ID: 2407263

arthur.milani-giovanini@polymtl.ca

arthurusp@usp.br

Gustavo Palma dos Santos

ID: 2411673

gustavo.palma-dos-santos@polymtl.ca

gustavo.palma.gp@usp.br

Abstract

This project aims to collaborate with NASA JPL sierra-method's development by creating a Graphical Domain-Specific Modeling Language (DSML) based on its Ontological Modeling Language (OML) vocabulary[1]. Both projects focus not only on simplifying the structure of Capella's diagrams but also on allowing reasoning large model instances through querying. The authors of this project worked on the System Capability Diagram that is specifically designed to define how the modeled system should execute the operations outlined in the Operational Capability Diagram by detailing the technical requirements and design needs necessary to achieve the desired system behavior. From the provided OML vocabulary[1], the abstract syntax as well as the Object Constraint Language (OCL) constraints were derived and parsed into a ECore metamodel instance. As for the results of this project, an Eclipse Sirius plugin was developed for the System Capability Diagram DSML, using Obeo Designer software, which allows users to build instances of the System Capability Diagram using graphical tools, such as nodes, edge creations, and additional validations in order to ensure the OCL constraints.

1 Language Definition

The Capella System Capability Diagram allows users to design complex systems by providing elements and relations that describe the internal features at the system's implementation level. The three elements in the System Capability Diagram are:

- **Actor:** external element that interacts with the system (e.g., users, operators, administrators).

- **Capability:** high-level functionality that the system must perform to meet its objectives.
- **Entity:** structural element within the system that is involved in delivering capabilities.

Additionally, to describe the relations between these elements, the following relations are supported:

- **Involves:** represents a participation relationship in which an actor or an entity is involved in a specific capability.
- **Specialization:** represents a generalization-specialization relationship, where an element inherits the characteristics of another element of the same type.
- **Extension:** represents an additive relationship in which a basic capability is extended with additional ones.
- **Inclusion:** represents a mandatory inclusion relationship in which one capability explicitly incorporates another.

2 How to Run the DSML plugin

In order to use the developed DSML plugin for the System Capability Diagram and leverage the modeling tools, the following installation steps must be accomplished:

1. Download Sirius Obeo Designer¹
2. Download the project's plugin .zip package²
3. Install the package as a plugin:

¹<https://www.obeodesigner.com/en/download>

²<https://github.com/Palmaa/LOG8505E-project/releases/tag/v1.0.0>

- Launch Eclipse Sirius Obeo Designer
- In the menu: Help → Install New Software... → Add → Archive
- Choose the downloaded .zip file
- Complete installation

After the installation is completed, the user can now create a model instance and a graphical representation from the installed plugin following the steps:

1. In the menu, create a modeling project:
File → New → Modeling Project
2. Instantiate a model:
File → New → Other → Capability Model → Model Object: Diagram → Finish
3. Create representation view:
representations.aird → New → System Capability → Next → Finish

With the representation view created, the user can use the tools from the palette to model an instance model. The model in Figure 1, inspired by the Capella's Course[2] on Youtube, was created using the Graphical DSML tools to design a Trail Power system.

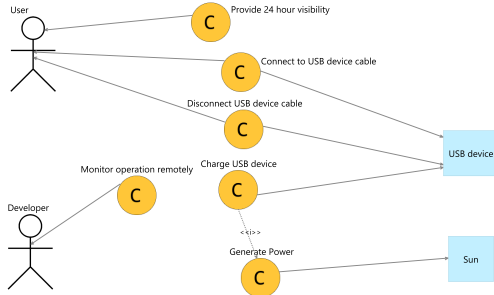


Figure 1: Trail Power Model Instance

3 Implementation Rationale and Details

Within the System Capability Diagram Metamodel, two interesting insights worth some additional comments, being both of them related to the abstract classes. The former concerns the abstract *SpecializableElement* class that takes place in the diagram to be the parent root element of the diagram and also to avoid unnecessary reflexive specializes relations' duplicates that would be needed for the Capability, Entity and Actor classes, improving the readability and clarity of the metamodel.

The latter regarding the *InvolvableElement* abstract class highlights that it plays an important role in

the relation *involves* between the Capability and Actor/Entity classes. Here, the same problem of relation duplication would happen without this parent abstract class.

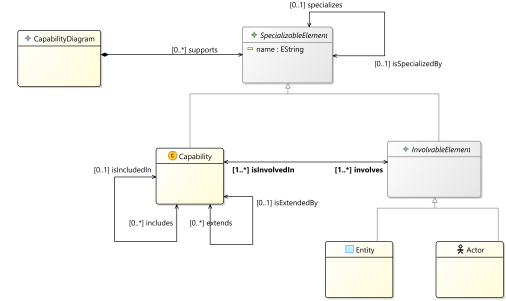


Figure 2: System Capability Diagram Metamodel

Along with the metamodel design, in order to implement all features of DSML the authors worked on the following files/folder:

- system.capability/model/capability.ecore: meta-model definition
- system.capability.edit/icons: actor and capability icons
- system.capability.design/description/capability.o-design: viewpoint representation definitions (abstract/concrete syntax mapping, tool palette and validations)
- system.capability.feature/update: generate the plugin installer

Also, the authors worked on the files below to create model instance samples:

- system.capability.examples/trail.power.capability
- system.capability.examples/library.system.capability
- system.capability.examples/queries.system.capability

4 Querying a Model Instance

This section presents several queries performed on the model instance Figure 3. Each query is listed below along with its result. Note that for all queries the context is the Diagram object.

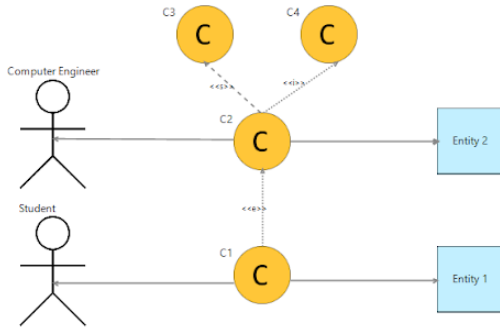


Figure 3: Instance model queried

1. **Number of capabilities:** get the total number of capabilities.

```
aql:self.supports->select(e |
e.ocIsTypeOf(capability::Capability))->size()
```

Expression
aql:self.supports->select(e e.ocIsTypeOf(capability::Capability))->size()
Evaluation Result
4

Figure 4: Result for Query 1

2. **Number of actors:** retrieve the total number of actors.

```
aql:self.supports->select(e |
e.ocIsTypeOf(capability::Actor))->size()
```

Expression
aql:self.supports->select(e e.ocIsTypeOf(capability::Actor))->size()
Evaluation Result
2

Figure 5: Result for Query 2

3. **Number of entities:** identify the total number of entities.

```
aql:self.supports->select(e |
e.ocIsTypeOf(capability::Entity))->size()
```

Expression
aql:self.supports->select(e e.ocIsTypeOf(capability::Entity))->size()
Evaluation Result
2

Figure 6: Result for Query 3

4. **Capabilities that do not involve entities:** select capabilities not involving entities.

```
aql:self.supports->select(e |
e.ocIsTypeOf(capability::Capability))
->select(c | c.involves->select(e |
e.ocIsTypeOf(capability::Entity))->isEmpty())
```

Expression
aql:self.supports->select(e e.ocIsTypeOf(capability::Capability))->select(c c.involves->select(e e.ocIsTypeOf(capability::Entity))->isEmpty())
Evaluation Result
Capability C3 Capability C4

Figure 7: Result for Query 4

5. **Capabilities related to another capability:** retrieve capabilities related to others.

```
aql:self.supports->select(e |
e.ocIsTypeOf(capability::Capability))
->reject(c | c.includes->isEmpty() and
c.extends->isEmpty() and c.isExtendedBy->isEmpty() and
c.isIncludedIn->isEmpty() and c.specializes->isEmpty()
and c.isSpecializedBy->isEmpty())
```

Expression
aql:self.supports->select(e e.ocIsTypeOf(capability::Capability))->reject(c c.includes->isEmpty() and c.extends->isEmpty() and c.isExtendedBy->isEmpty() and c.isIncludedIn->isEmpty() and c.specializes->isEmpty() and c.isSpecializedBy->isEmpty())
Evaluation Result
Capability C1 Capability C2 Capability C3 Capability C4

Figure 8: Result for Query 5

6. **Elements with big names:** identify elements with names longer than 10 characters.

```
aql:self.supports->reject(c | c.name.size() <= 10)
```

Expression
aql:self.supports->reject(c c.name.size() <= 10)
Evaluation Result
Actor Computer Engineer

Figure 9: Result for Query 6

7. **Actors related to fewer than 5 capabilities:** retrieve actors involved in fewer than five capabilities.

```
aql:self.supports->select(e |
e.ocIsTypeOf(capability::Actor)) ->select(a |
a.isInvolvedIn->size() < 5)
```



Expression
aql:self.supports->select(e e.oclsTypeOf(capability::Actor))->select(a a.isInvolvedIn->size() < 5)
Evaluation Result
 Actor Student  Actor Computer Engineer

Figure 10: Result for Query 7

5 Workload Distribution

Both authors provided significant and meaningful contributions during the development of the project and presentation/report elaboration. The author's main contributions are described as follows:

- Gustavo Palma dos Santos:
 - Development of the DSML (palette tool and validations)
 - Sample model instances
- Arthur Milani Giovanini
 - OCL constraints
 - Queries elaboration

References

- [1] M. Elaasar. An implementation of the sierra method using openCAESAR, <https://github.com/modelware/sierra-method>. *Accessed:* November 28, 2024.
- [2] T. Komar. An implementation of the sierra method using openCAESAR, <https://youtube.com/playlist?list=PLfrEYVpSGV-LywKkR4UWQ-R3ISIOly9xke>. *Accessed:* November 25, 2024.