Adaptive Structural Operational Semantics

Gwendal Jouneaux [1] Damian Frölich [3] Olivier Barais [1]
Benoit Combemale [1] Gurvan Le Guernic [2 4] Gunter Mussbacher [2 5]
L. Thomas van Binsbergen [3]

[1]Univ. Rennes – Rennes, France [2]Inria – Rennes, France

[3]University of Amsterdam – Amsterdam, The Netherlands

[4]DGA Maîtrise de lInformation – Rennes, France

[5]McGill University – Montreal, Canada

SLE'23 — October 23, 2023

Context

Software . . .

Context

Software . . .

- Evolve in complex/changing environment (e.g, IoT, embedded systems)

Context

Software ...

- Evolve in complex/changing environment (e.g, IoT, embedded systems)
- Need dynamic adaptation to best deliver the service (e.g., Waymo[1], Netflix[1])

[1] Cf. `https://waymo.com`, `https://www.netflix.com`

Context

Software . . .

- Evolve in complex/changing environment (e.g, IoT, embedded systems)
- Need dynamic adaptation to best deliver the service (e.g., Waymo[1], Netflix[1])

Consequence : There is a need to provide abstraction for self-adaption when it's a secondary but important concern

[1] Cf. `https://waymo.com`, `https://www.netflix.com`

Abstraction for self-adaptation

Problem studied in the Self-Adaptive Systems community (*e.g.* SEAMS[2])

[2] Software Engineering for Adaptive and Self-Managing Systems

Abstraction for self-adaptation

Problem studied in the Self-Adaptive Systems community (*e.g.* SEAMS[2])

- Architectural solutions (*e.g.* MAPE-K [3], 3 Layer Architecture [4], MORPH [5])

[3] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003, ISSN: 1558-0814

[4] J. Kramer and J. Magee, "Self-managed systems: An architectural challenge," in *Future of Software Engineering (FOSE'07)*, IEEE, 2007, pp. 259–268

[5] V. Braberman, N. D'Ippolito, J. Kramer, *et al.*, "Morph: A reference architecture for configuration and behaviour self-adaptation," in *Proceedings of the 1st International Workshop on Control Theory for Software Engineering*, 2015, pp. 9–16

Abstraction for self-adaptation

Problem studied in the Self-Adaptive Systems community (*e.g.* SEAMS[2])

- Architectural solutions (*e.g.* MAPE-K [3], 3 Layer Architecture [4], MORPH [5])
- Frameworks (*e.g.* Executable Runtime Megamodels [6], DCL [7], Ponder2 [8])

---

[6] T. Vogel and H. Giese, "A language for feedback loops in self-adaptive systems: Executable runtime megamodels," in *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, IEEE, 2012, pp. 129–138

[7] H. Nakagawa, A. Ohsuga, and S. Honiden, "Towards dynamic evolution of self-adaptive systems based on dynamic updating of control loops," in *2012 IEEE Sixth International Conference on Self-Adaptive and Self-Organizing Systems*, IEEE, 2012, pp. 59–68

[8] K. Twidle, N. Dulay, E. Lupu, *et al.*, "Ponder2: A policy system for autonomous pervasive environments," in *2009 Fifth International Conference on Autonomic and Autonomous Systems*, IEEE, 2009, pp. 330–335

Abstraction for self-adaptation

Problem studied in the Self-Adaptive Systems community (*e.g.* SEAMS[2])

- Architectural solutions (*e.g.* MAPE-K [3], 3 Layer Architecture [4], MORPH [5])
- Frameworks (*e.g.* Executable Runtime Megamodels [6], DCL [7], Ponder2 [8])

Yes, but I already/want to have a nice DSL tailored to my domain

DSLs and self-adaptation

DSLs are tools for experts with appropriate constructs for a given domain.

What if self-adaptation is **not** a primary concern ?

DSLs and self-adaptation

DSLs are tools for experts with appropriate constructs for a given domain.

What if self-adaptation is **not** a primary concern ?

- Re-implementation of frameworks tedious or impossible

DSLs and self-adaptation

DSLs are tools for experts with appropriate constructs for a given domain.

What if self-adaptation is **not** a primary concern ?

- Re-implementation of frameworks tedious or impossible
- Require expertise in self-adaptation from the language users

DSLs and self-adaptation

DSLs are tools for experts with appropriate constructs for a given domain.

What if self-adaptation is **not** a primary concern ?

- Re-implementation of frameworks tedious or impossible
- Require expertise in self-adaptation from the language users

Example of RobLANG :

- Procedural DSL to program robot missions
- Abstraction for robot sensors and actuators
- For instance, we might want to adapt the robot's speed to save energy

Manual Implementation of Adaptations (1)

```
1  void adapt (double energy, double time) {
2      double tradeoffquarter = energy * 0.984375 + time * -0.75
3      double tradeoffhalf = energy * 0.875 + time * -0.5
4      double tradeoffthreequarter = energy * 0.578125 + time * -0.25
5
6      if tradeoffquarter < tradeoffhalf {
7          if tradeoffhalf < tradeoffthreequarter {
8              setSpeed(0.0 < tradeoffthreequarter ? MAX_SPEED * 0.75 : MAX_SPEED)
9          } else {
10             setSpeed(0.0 < tradeoffhalf ? MAX_SPEED * 0.5 : MAX_SPEED)
11         }
12     } else {
13         if tradeoffquarter < tradeoffthreequarter {
14             setSpeed(0.0 < tradeoffthreequarter ? normalSpeed * 0.75 : MAX_SPEED)
15         } else {
16             setSpeed(0.0 < tradeoffquarter ? MAX_SPEED * 0.25 : MAX_SPEED)
17         }
18     }
19 }
```

Manual Implementation of Adaptations (2)

For each system, manually designed adaptation requires:

- Expertise to design the system
- Manual implementation of the trade-off reasoning
- To be correctly integrated to the existing code base.

Is it the end ?

Previous work : The SEALS Framework [SLE '21]

SEALS: A framework for building Self-Adaptive Virtual Machines [9]

- Framework to implement DSLs with self-adaptive operational semantics
- Provides modularity for adaptation definitions

https://inria.hal.science/hal-03355253/document

[9] G. Jouneaux, O. Barais, B. Combemale, *et al.*, "SEALS: A framework for building Self-Adaptive Virtual Machines," in *SLE 2021*, Chicago, United States, Oct. 2021. DOI: 10.1145/3486608.3486912

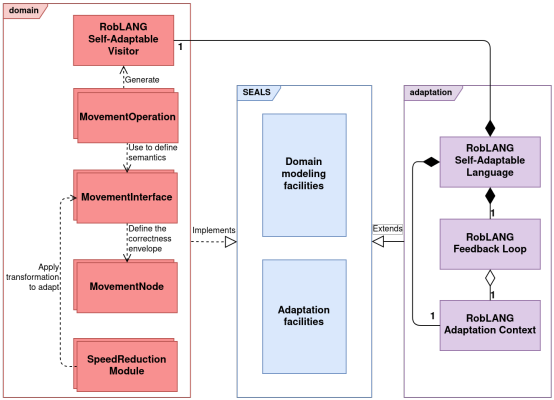
The SEALS Framework

Conceptually free the DSL user of :

- Implementing the feedback-loop
- Integrate the execution variants

Concretely helps the DSL designer in :

- Modeling domain concepts
- Adaptation process' specialization



Overview of RobLANG in SEALS

The SEALS Framework

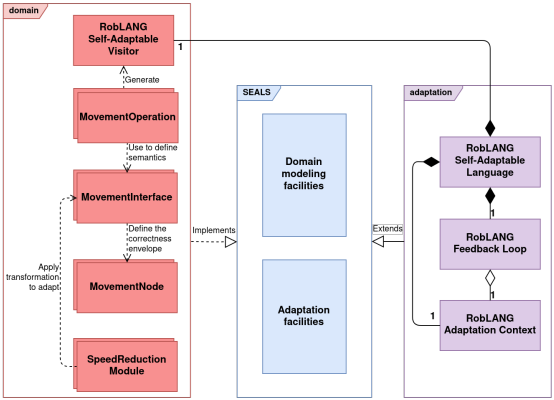
Conceptually free the DSL user of :

- Implementing the feedback-loop
- Integrate the execution variants

Concretely helps the DSL designer in :

- Modeling domain concepts
- Adaptation process' specialization

However, it remains hard to reason over the operational semantics



Overview of RobLANG in SEALS

ASOS : Adaptive Structural Operational Semantics

A metalanguage to specify and reason on self-adaptable operational semantics

Adaptive Structural Operational Semantics

Defined as an extension of MSOS [10] :

- Metalanguage to define Operational Semantics
- Modular definition of semantic rules
- Ability to verify properties such as determinism, completeness or termination

Extended by providing additional adaptation rules and how to introduce them

Generation of a SEALS implementation for execution

[10] P. D. Mosses, “Modular structural operational semantics,” *The Journal of Logic and Algebraic Programming*, vol. 60, pp. 195–228, 2004

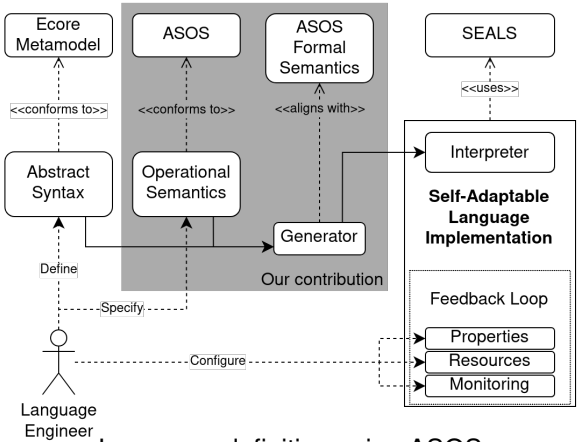
Approach overview

Language definition using ASOS :

- Abstract syntax as metamodel
- Semantics rules in ASOS
- Feedback-loop configuration

ASOS semantics components :

- A "default" semantics
- Adaptation semantics
- Pointcuts for adaptation semantics



Language definition using ASOS

Semantic rules in ASOS

```
1  model RobLANG.ecore with sd
2
3  rule program,
4       Program(command) -> Program(newcommand)
5  resolve
6       command -> newcommand
7
8  rule program_error,
9       Program(command) -> sd.RuntimeError()
10 resolve
11      command -> termination sd.RuntimeError()
12
13 rule assign_set,
14      Assignment(name, sd.Integer(n))
15      ->
16      sd.NilValue()
17 bind
18      self.value = sd.Integer(n)
```

```
1 rule if_then,
2      If(sd.Boolean(b), c1, c2) -> c1
3 where
4      b == true
```

- Model merging metamodel and dynamic information

Semantic rules in ASOS

```
1  model RobLANG.ecore with sd
2
3  rule program,
4       Program(command) -> Program(newcommand)
5  resolve
6       command -> newcommand
7
8  rule program_error,
9       Program(command) -> sd.RuntimeError()
10 resolve
11      command -> termination sd.RuntimeError()
12
13 rule assign_set,
14      Assignment(name, sd.Integer(n))
15      ->
16      sd.NilValue()
17 bind
18      self.value = sd.Integer(n)
```

```
1  rule if_then,
2       If(sd.Boolean(b), c1, c2) -> c1
3  where
4       b == true
```

- Model merging metamodel and dynamic information
- A set of semantic rules
  - Conclusion as transition between concepts
  - Transition premises
  - Binding computed values
  - Side condition

Semantic rules in ASOS

```
1  model RobLANG.ecore with sd
2
3  rule program,
4       Program(command) -> Program(newcommand)
5  resolve
6       command -> newcommand
7
8  rule program_error,
9       Program(command) -> sd.RuntimeError()
10 resolve
11      command -> termination sd.RuntimeError()
12
13 rule assign_set,
14      Assignment(name, sd.Integer(n))
15      ->
16      sd.NilValue()
17 bind
18      self.value = sd.Integer(n)
```

```
1 rule if_then,
2     If(sd.Boolean(b), c1, c2) -> c1
3 where
4     b == true
```

- Model merging metamodel and dynamic information
- A set of semantic rules
  - Conclusion as transition between concepts
  - Transition premises
  - Binding computed values
  - Side condition

Semantic rules in ASOS

```
1  model RobLANG.ecore with sd
2
3  rule program,
4       Program(command) -> Program(newcommand)
5  resolve
6       command -> newcommand
7
8  rule program_error,
9       Program(command) -> sd.RuntimeError()
10 resolve
11      command -> termination sd.RuntimeError()
12
13 rule assign_set,
14      Assignment(name, sd.Integer(n))
15      ->
16      sd.NilValue()
17 bind
18      self.value = sd.Integer(n)
```

```
1 rule if_then,
2     If(sd.Boolean(b), c1, c2) -> c1
3 where
4     b == true
```

- Model merging metamodel and dynamic information
- A set of semantic rules
  - Conclusion as transition between concepts
  - Transition premises
  - Binding computed values
  - Side condition

Semantic rules in ASOS

```
1  model RobLANG.ecore with sd
2
3  rule program,
4       Program(command) -> Program(newcommand)
5  resolve
6       command -> newcommand
7
8  rule program_error,
9       Program(command) -> sd.RuntimeError()
10 resolve
11      command -> termination sd.RuntimeError()
12
13 rule assign_set,
14      Assignment(name, sd.Integer(n))
15      ->
16      sd.NilValue()
17 bind
18      self.value = sd.Integer(n)
```

```
1 rule if_then,
2      If(sd.Boolean(b), c1, c2) -> c1
3 where
4      b == true
```

- Model merging metamodel and dynamic information
- A set of semantic rules
  - Conclusion as transition between concepts
  - Transition premises
  - Binding computed values
  - Side condition

Semantic rules in ASOS

```
1  model RobLANG.ecore with sd
2
3  rule program,
4       Program(command) -> Program(newcommand)
5  resolve
6       command -> newcommand
7
8  rule program_error,
9       Program(command) -> sd.RuntimeError()
10 resolve
11      command -> termination sd.RuntimeError()
12
13 rule assign_set,
14      Assignment(name, sd.Integer(n))
15      ->
16      sd.NilValue()
17 bind
18      self.value = sd.Integer(n)
```

```
1 rule if_then,
2     If(sd.Boolean(b), c1, c2) -> c1
3 where
4     b == true
```

- Model merging metamodel and dynamic information
- A set of semantic rules
  - Conclusion as transition between concepts
  - Transition premises
  - Binding computed values
  - Side condition

Semantic rules in ASOS

```
1  model RobLANG.ecore with sd
2
3  rule program,
4       Program(command) -> Program(newcommand)
5  resolve
6       command -> newcommand
7
8  rule program_error,
9       Program(command) -> sd.RuntimeError()
10 resolve
11      command -> termination sd.RuntimeError()
12
13 rule assign_set,
14      Assignment(name, sd.Integer(n))
15      ->
16      sd.NilValue()
17 bind
18      self.value = sd.Integer(n)
```

```
1 rule if_then,
2     If(sd.Boolean(b), c1, c2) -> c1
3 where
4     b == true
```

- Model merging metamodel and dynamic information
- A set of semantic rules
  - Conclusion as transition between concepts
  - Transition premises
  - Binding computed values
  - Side condition

Expressing adaptations

```
1  ApproximateDouble{
2
3      match Assignement(VarRef(def), expr)
4      where def.type == Float
5
6      Before binop_rhs rule binop_lhs_to_float,
7          Binop(sd.Double(n1), a2)
8          ->
9          Binop(sd.Float(n1), a2)
10
11     Before binop_compute rule
       binop_rhs_to_float,
12         Binop(Number(n1), sd.Double(n2))
13         ->
14         Binop(Number(n1), sd.Float(n2))
15 }
```

- Matching definition
  - Structural matching
  - Additional constraints
- Adaptation rules
  - Kind of adaptation rule
  - Affected rule in semantics
  - Adaptation semantic rule

Expressing adaptations

```
1  ApproximateDouble{
2
3      match Assignement(VarRef(def), expr)
4      where def.type == Float
5
6      Before binop_rhs rule binop_lhs_to_float,
7          Binop(sd.Double(n1), a2)
8          ->
9          Binop(sd.Float(n1), a2)
10
11     Before binop_compute rule
       binop_rhs_to_float,
12         Binop(Number(n1), sd.Double(n2))
13         ->
14         Binop(Number(n1), sd.Float(n2))
15 }
```

- Matching definition
  - Structural matching
  - Additional constraints
- Adaptation rules
  - Kind of adaptation rule
  - Affected rule in semantics
  - Adaptation semantic rule

Expressing adaptations

```
1  ApproximateDouble{
2
3      match Assignement(VarRef(def), expr)
4      where def.type == Float
5
6      Before binop_rhs rule binop_lhs_to_float,
7          Binop(sd.Double(n1), a2)
8          ->
9          Binop(sd.Float(n1), a2)
10
11     Before binop_compute rule
       binop_rhs_to_float,
12         Binop(Number(n1), sd.Double(n2))
13         ->
14         Binop(Number(n1), sd.Float(n2))
15 }
```

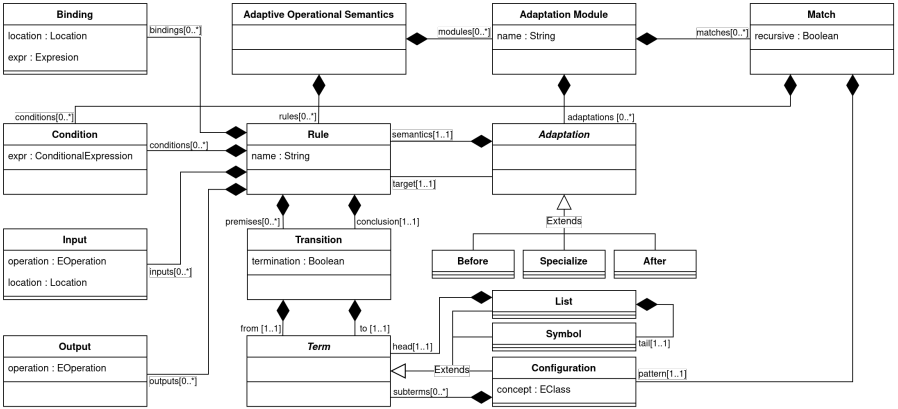
- Matching definition
  - Structural matching
  - Additional constraints
- Adaptation rules
  - Kind of adaptation rule
  - Affected rule in semantics
  - Adaptation semantic rule

Expressing adaptations

```
1  ApproximateDouble{
2
3      match Assignement(VarRef(def), expr)
4      where def.type == Float
5
6      Before binop_rhs rule binop_lhs_to_float,
7          Binop(sd.Double(n1), a2)
8          ->
9          Binop(sd.Float(n1), a2)
10
11     Before binop_compute rule
       binop_rhs_to_float,
12         Binop(Number(n1), sd.Double(n2))
13         ->
14         Binop(Number(n1), sd.Float(n2))
15 }
```

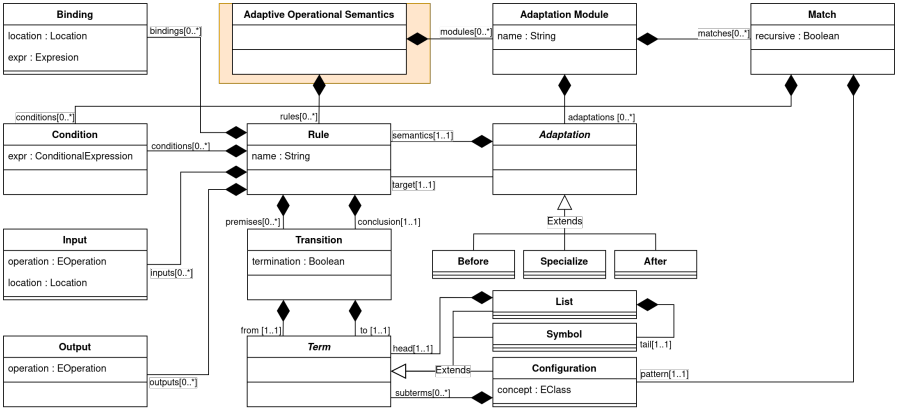
- Matching definition
  - Structural matching
  - Additional constraints
- Adaptation rules
  - Kind of adaptation rule
  - Affected rule in semantics
  - Adaptation semantic rule

The Metamodel of ASOS



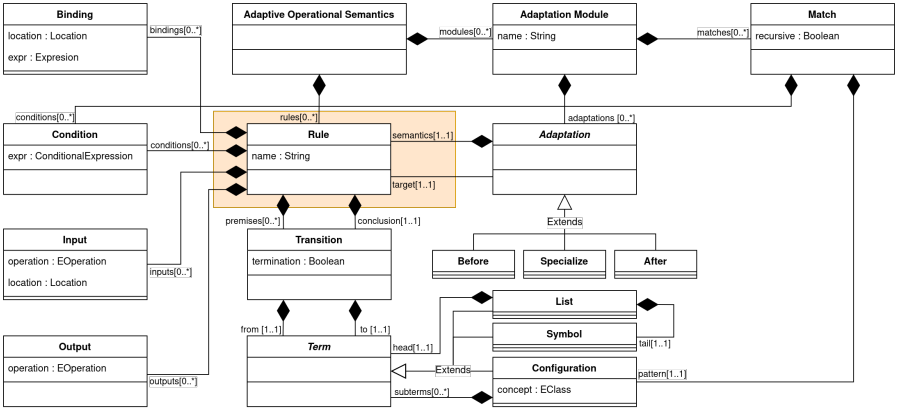
ASOS Metamodel (Expressions and Location omitted to focus on the main Concepts)

The Metamodel of ASOS



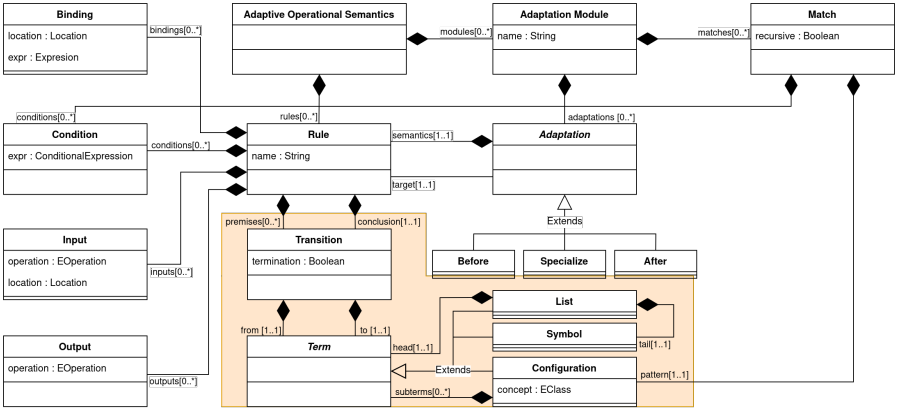
ASOS Metamodel (Expressions and Location omitted to focus on the main Concepts)

The Metamodel of ASOS



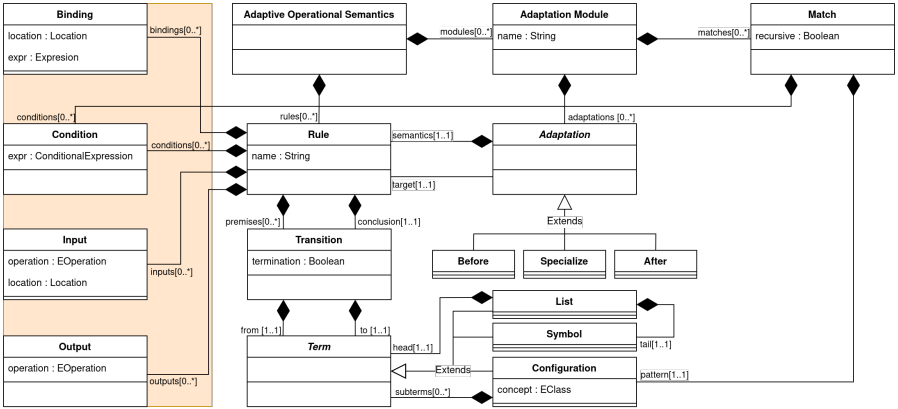
ASOS Metamodel (Expressions and Location omitted to focus on the main Concepts)

The Metamodel of ASOS



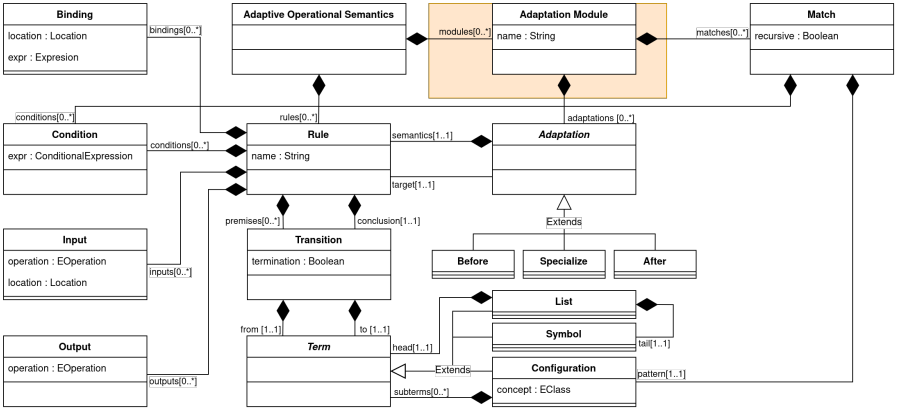
ASOS Metamodel (Expressions and Location omitted to focus on the main Concepts)

The Metamodel of ASOS



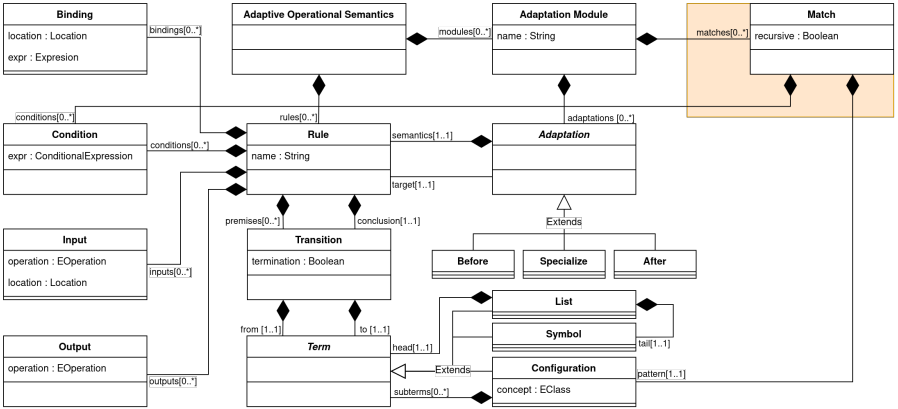
ASOS Metamodel (Expressions and Location omitted to focus on the main Concepts)

The Metamodel of ASOS



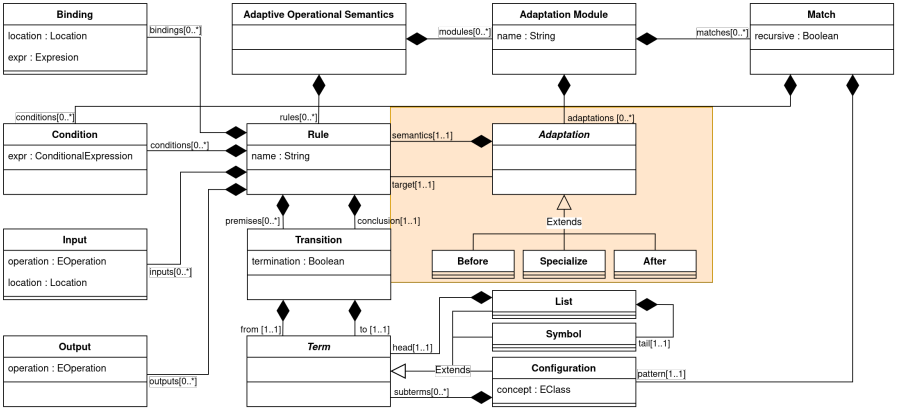
ASOS Metamodel (Expressions and Location omitted to focus on the main Concepts)

The Metamodel of ASOS



ASOS Metamodel (Expressions and Location omitted to focus on the main Concepts)

The Metamodel of ASOS



ASOS Metamodel (Expressions and Location omitted to focus on the main Concepts)

ASOS Formal Semantics

$\pi(\gamma)(\kappa(\text{source(X)}))$ = D'
Get the set of activated adaptations

$\zeta(\text{D} \cup \text{D'}) = \delta'$
Select the adaptations to perform

$\text{D} \diamond \text{D'} \vdash (\gamma, \delta') \rightsquigarrow \gamma'$
There is a valid adaptation transition to γ'

$\text{D} \diamond \text{D'} \vdash (\gamma, \delta') \not\rightsquigarrow$
There is no valid adaptation transition

$\text{D} \diamond \text{D'} \vdash \gamma \longrightarrow \gamma'$
Apply the default semantics

$$\text{adaptation}\ \frac{\begin{array}{c}\pi(\gamma)(\kappa(source(X))) = D' \\ \zeta(D \cup D') = \delta' \\ D \diamond D' \vdash (\gamma, \delta') \rightsquigarrow \gamma'\end{array}}{D \vdash \gamma \twoheadrightarrow \gamma'}$$

$$\text{default}\ \frac{\begin{array}{c}\pi(\gamma)(\kappa(source(X))) = D' \\ \zeta(D \cup D') = \delta' \\ D \diamond D' \vdash (\gamma, \delta') \not\rightsquigarrow \\ D \diamond D' \vdash \gamma \longrightarrow \gamma'\end{array}}{D \vdash \gamma \twoheadrightarrow \gamma'}$$

ASOS Formal Semantics

$\pi(\gamma)(\kappa(\text{source(X)}))$ = D'
Get the set of activated adaptations

$\zeta(\text{D} \cup \text{D'}) = \delta'$
Select the adaptations to perform

$\text{D} \diamond \text{D'} \vdash (\gamma, \delta') \rightsquigarrow \gamma'$
There is a valid adaptation transition to γ'

$\text{D} \diamond \text{D'} \vdash (\gamma, \delta') \not\rightsquigarrow$
There is no valid adaptation transition

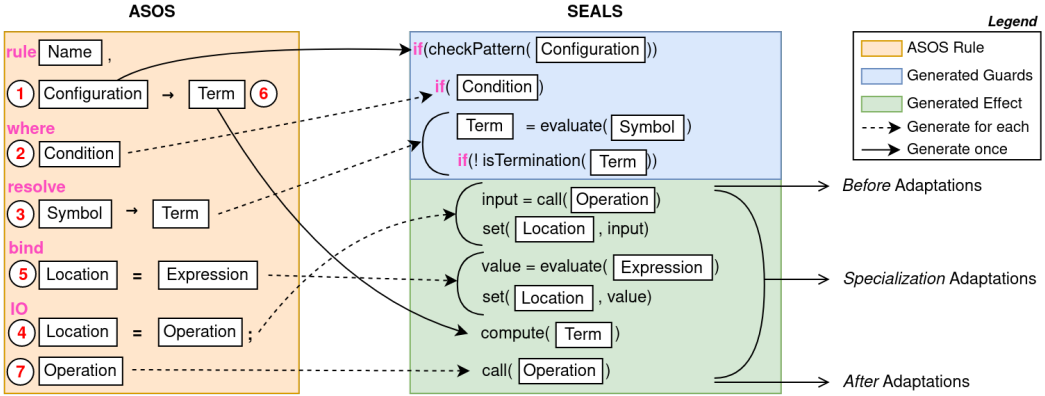
$\text{D} \diamond \text{D'} \vdash \gamma \longrightarrow \gamma'$
Apply the default semantics

$$\text{adaptation}\ \frac{\begin{array}{c}\pi(\gamma)(\kappa(source(X))) = D' \\ \zeta(D \cup D') = \delta' \\ D \diamond D' \vdash (\gamma, \delta') \rightsquigarrow \gamma'\end{array}}{D \vdash \gamma \twoheadrightarrow \gamma'}$$

$$\text{default}\ \frac{\begin{array}{c}\pi(\gamma)(\kappa(source(X))) = D' \\ \zeta(D \cup D') = \delta' \\ D \diamond D' \vdash (\gamma, \delta') \not\rightsquigarrow \\ D \diamond D' \vdash \gamma \longrightarrow \gamma'\end{array}}{D \vdash \gamma \twoheadrightarrow \gamma'}$$

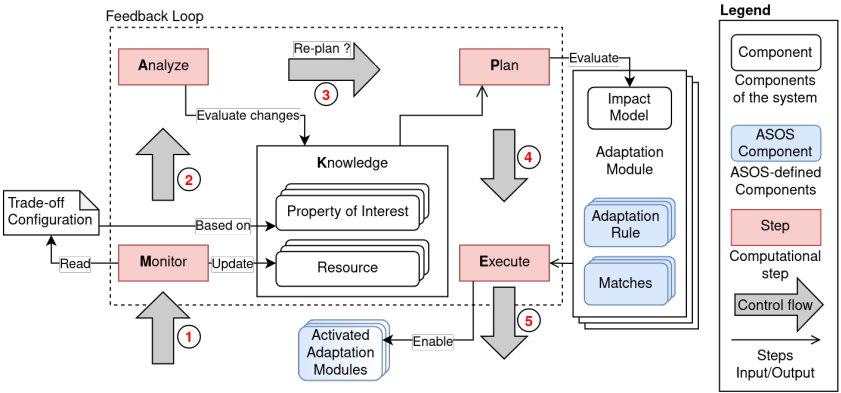
For more detail : Take a look at the paper !

ASOS Translational Semantics



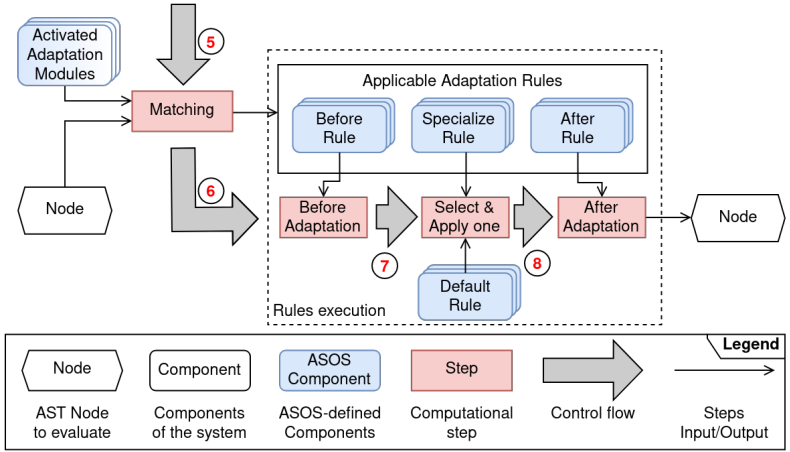
General pattern of translation from ASOS rules to SEALS Operation in Java

Execution of an ASOS Rule (1)



Execution of the feedback loop and activation of adaptations

Execution of an ASOS Rule (2)



Execution of the matching system and call of rules

Evaluation

We provide proof for :

- Determinism
- Completeness

On non-termination :

- Patterns can be detected
- Currently over estimate

Applicability on the Roblang DSL :

- Adaptations proposed affect correctly the behavior of the robot
- Adaptations are correctly selected depending on the context

Performance

We compare ASOS to self-adaptation defined in the program

Experimental setup :

- Experimented on 31Gb of RAM, i7-10850H and OpenJDK 11.0.18
- 30 executions in a row
- Repeated 3 times with reboot mitigating effects of the initial state[12].

Results :

- Speedups ranging from x0.80 to 0.95 (Geometrical Mean of x0.88).
- Probably due to JVM optimizations of the feedback loop in Java

[12] T. Kalibera, L. Bulej, and P. Tuma, "Benchmark precision and random initial state," in *Proceedings of the 2005 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2005)*, 2005, pp. 484–490

Conclusion

The ASOS metalanguage provides :

- Abstractions for defining operational semantics adaptations
- Mechanism for composing adaptations with the default semantics
- Support for verifying properties (*e.g.* determinism) of operational semantics

Technical aspects :

- IDE tool support for ASOS
- Generator targeting the SEALS Framework
- Compatible with Xtext

Perspectives

- Evaluate the complexity of using the ASOS metalanguage
- Reify SEALS correctness envelope at the rule level
- Extend ASOS to formally configure the feedback loop
- Use ASOS declarative nature to support SALs composition

Thanks for your attention!

https://www.gwendal-jouneaux.fr/assets/pdf/ASOS_SLE2023.pdf