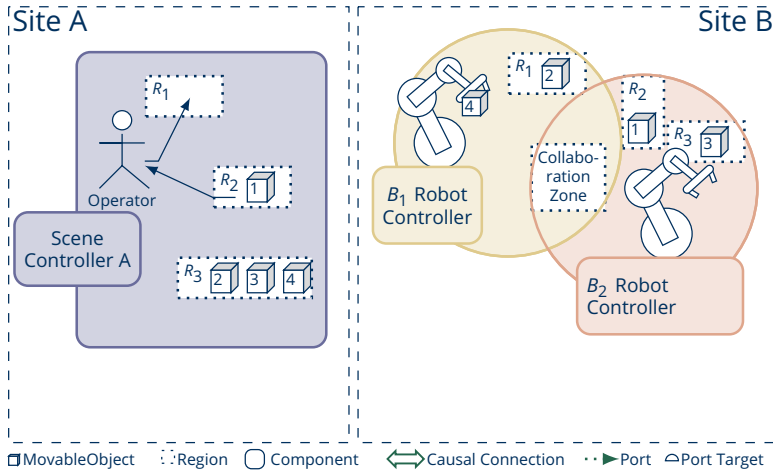


René Schöne, Johannes Mey, Sebastian Ebert, Sebastian Götz, Uwe Aßmann

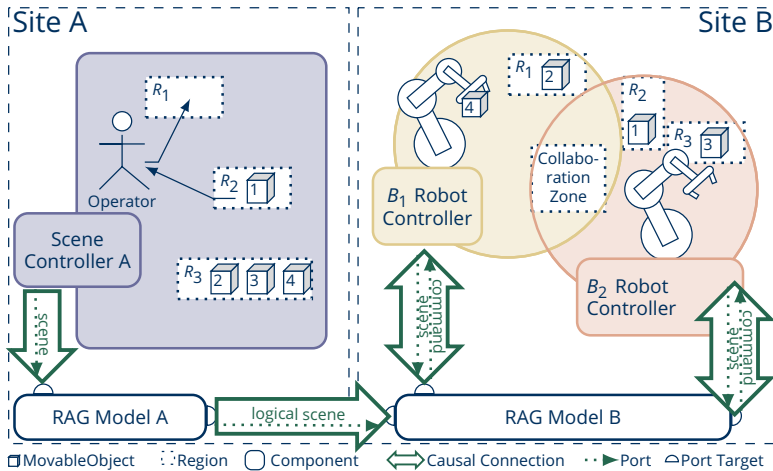
Incremental Causal Connection for Self-Adaptive Systems Based on Relational Reference Attribute Grammars

Montréal, October 26th 2022

A Case Study



A Case Study



Combining Two Concepts

Self-Adaptive Systems (SAS)

- Modern software: Start small. Gets complex, distributed, heterogeneous
- Goals: Separation of concerns, fast prototyping, theoretical grounding, efficiency
- One approach: Model-driven engineering / Models@run.time

Reference Attribute Grammars (RAG) [Hed00]

- Based on Attribute Grammars [Knu68]
- Origin: compiler construction (program analysis)
- Advantages: concise specification, efficient evaluation, natural separation of concerns

Combining Two Concepts

Self-Adaptive Systems (SAS)

- Modern software: Start small. Gets complex, distributed, heterogeneous
- Goals: Separation of concerns, fast prototyping, theoretical grounding, efficiency
- One approach: Model-driven engineering / Models@run.time

Reference Attribute Grammars (RAG) [Hed00]

- Based on Attribute Grammars [Knu68]
- Origin: compiler construction (program analysis)
- Advantages: concise specification, efficient evaluation, natural separation of concerns

Combining Two Concepts

Self-Adaptive Systems (SAS)

- Modern software: Start small. Gets complex, distributed, heterogeneous
- Goals: Separation of concerns, fast prototyping, theoretical grounding, efficiency
- One approach: Model-driven engineering / Models@run.time

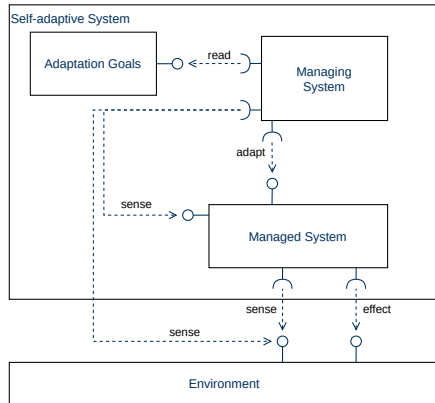
Reference Attribute Grammars (RAG) [Hed00]

- Based on Attribute Grammars [Knu68]
- Origin: compiler construction (program analysis)
- Advantages: concise specification, efficient evaluation, natural separation of concerns



What if: we use RAGs as technological space for the model?

Challenges Ahead



[Wey19]

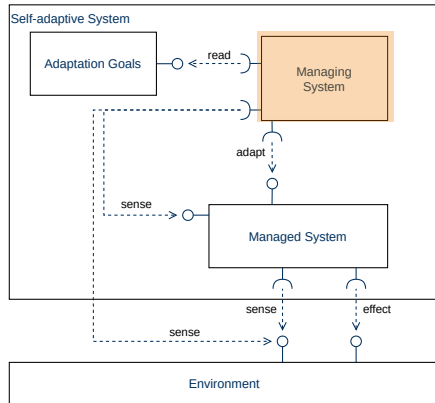
Specifications of Managing System

- modular
- (re-)usable
- understandable
- correct
- efficient

Connections from Managing System

- modular
- declarative
- reusable

Challenges Ahead



[Wey19]

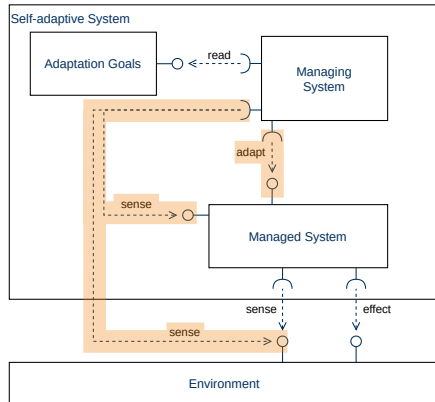
Specifications of Managing System

- modular
- (re-)usable
- understandable
- correct
- efficient

Connections from Managing System

- modular
- declarative
- reusable

Challenges Ahead



[Wey19]

Specifications of Managing System

- modular
- (re-)usable
- understandable
- correct
- efficient

Connections from Managing System

- modular
- declarative
- reusable

Model Specification using Attribute Grammars (AGs)

Attribute Grammars specify **semantics/behaviour** for a **tree** of a context-free grammar.

We use the **JastAdd** object-oriented Reference AG system [HM03].

Tree Structure

- Concise specification along tree
inherited (\downarrow), synthesized (\uparrow) attributes
- Collections (JastAdd)

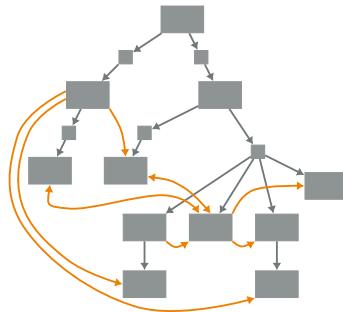
Declarative Specification

- Efficient evaluation
 - Incremental evaluation
 - Concurrent evaluation
- Circular attributes, AOP features (JastAdd)

Grammars as Models

Main Difference: Relations

- In **models**:
 - Containment relations form *overlay tree*
 - Non-containment relations
 - Bidirectional relations
 - In **grammars**:
 - Containment references:
 - Non-containment references
 - Bidirectional references
- AST } Relational RAGs [Mey+20]



Grammars as Models

```
WorldModel ::= [MyScene:Scene]
             ExecutedOperation:Operation*
             /NextOperation:Operation/;

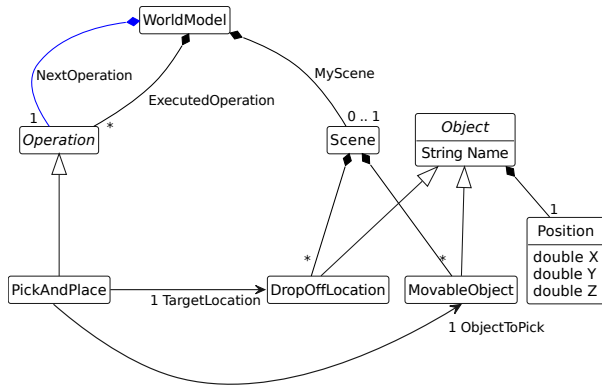
Position ::= <X:double> <Y:double> <Z:double>;

Scene ::= DropOffLocation* MovableObject*;

abstract Object ::= <Name:String> Position;

DropOffLocation : Object ::= ;
MovableObject : Object ::= ;

abstract Operation ;
PickAndPlace : Operation ::= ;
```



Grammars as Models

```
WorldModel ::= [MyScene:Scene]
             ExecutedOperation:Operation*
             /NextOperation:Operation/;

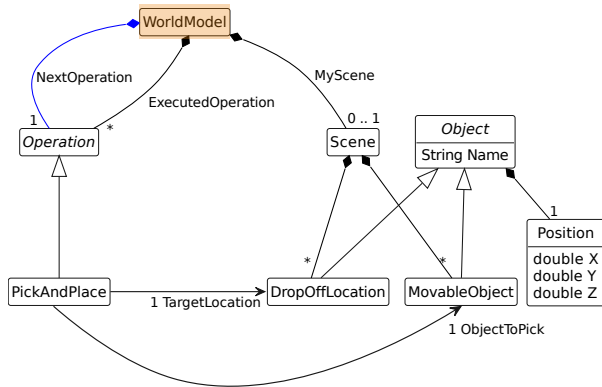
Position ::= <X:double> <Y:double> <Z:double>;

Scene ::= DropOffLocation* MovableObject*;

abstract Object ::= <Name:String> Position;

DropOffLocation : Object ::= ;
MovableObject : Object ::= ;

abstract Operation ;
PickAndPlace : Operation ::= ;
```



Grammars as Models

```
WorldModel ::= [MyScene:Scene]
              ExecutedOperation:Operation*
              /NextOperation:Operation/;

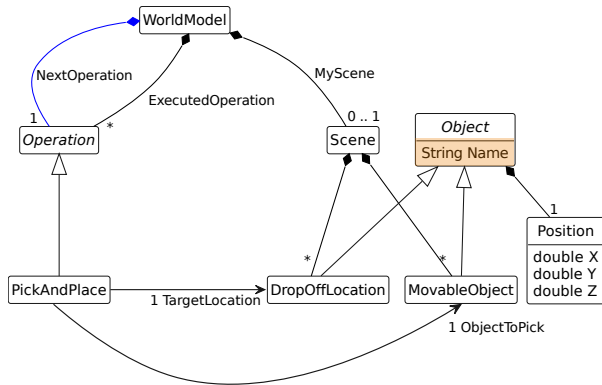
Position ::= <X:double> <Y:double> <Z:double>;

Scene ::= DropOffLocation* MovableObject*;

abstract Object ::= <Name:String> Position;

DropOffLocation : Object ::= ;
MovableObject : Object ::= ;

abstract Operation ;
PickAndPlace : Operation ::= ;
```



Grammars as Models

```
WorldModel ::= [MyScene:Scene]
              ExecutedOperation:Operation*
              /NextOperation:Operation/;

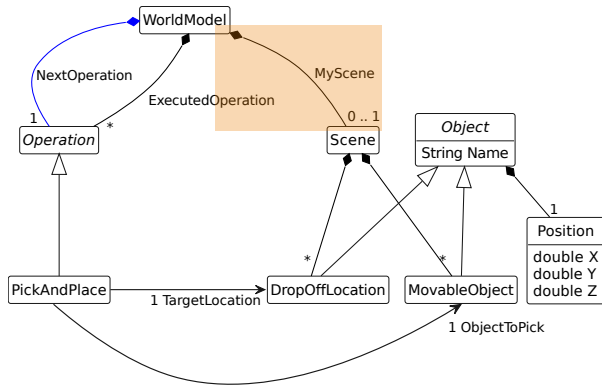
Position ::= <X:double> <Y:double> <Z:double>;

Scene ::= DropOffLocation* MovableObject*;

abstract Object ::= <Name:String> Position;

DropOffLocation : Object ::= ;
MovableObject : Object ::= ;

abstract Operation ;
PickAndPlace : Operation ::= ;
```



Grammars as Models

```
WorldModel ::= [MyScene:Scene]
              ExecutedOperation:Operation*
              /NextOperation:Operation/;

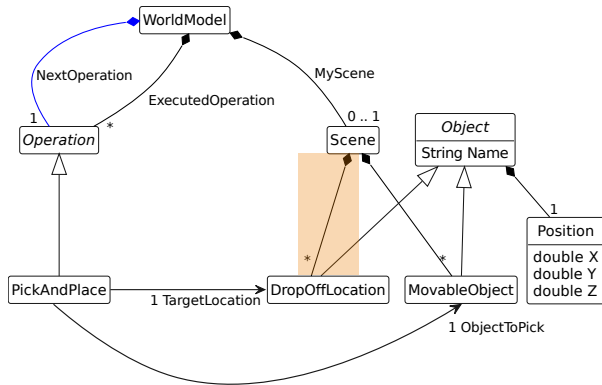
Position ::= <X:double> <Y:double> <Z:double>;

Scene ::= DropOffLocation* MovableObject*;

abstract Object ::= <Name:String> Position;

DropOffLocation : Object ::= ;
MovableObject : Object ::= ;

abstract Operation ;
PickAndPlace : Operation ::= ;
```



Grammars as Models

```
WorldModel ::= [MyScene:Scene]
              ExecutedOperation:Operation*
              /NextOperation:Operation/;

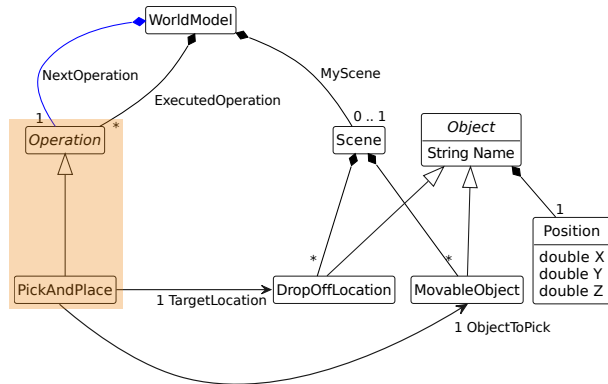
Position ::= <X:double> <Y:double> <Z:double>;

Scene ::= DropOffLocation* MovableObject*;

abstract Object ::= <Name:String> Position;

DropOffLocation : Object ::= ;
MovableObject : Object ::= ;

abstract Operation ;
PickAndPlace : Operation ::= ;
```



Grammars as Models

```
WorldModel ::= [MyScene:Scene]
              ExecutedOperation:Operation*
              /NextOperation:Operation/;

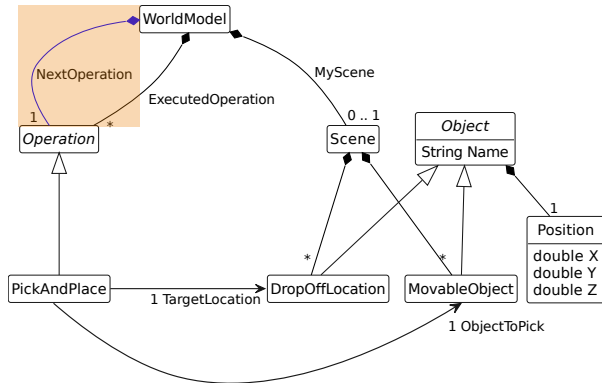
Position ::= <X:double> <Y:double> <Z:double>;

Scene ::= DropOffLocation* MovableObject*;

abstract Object ::= <Name:String> Position;

DropOffLocation : Object ::= ;
MovableObject : Object ::= ;

abstract Operation ;
PickAndPlace : Operation ::= ;
```



Grammars as Models

```
WorldModel ::= [MyScene:Scene]
              ExecutedOperation:Operation*
              /NextOperation:Operation/;

Position ::= <X:double> <Y:double> <Z:double>;

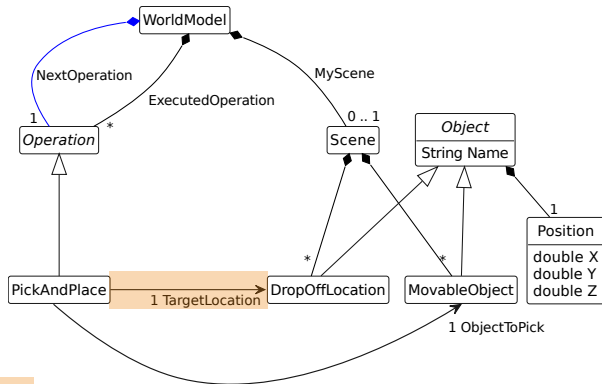
Scene ::= DropOffLocation* MovableObject*;

abstract Object ::= <Name:String> Position;

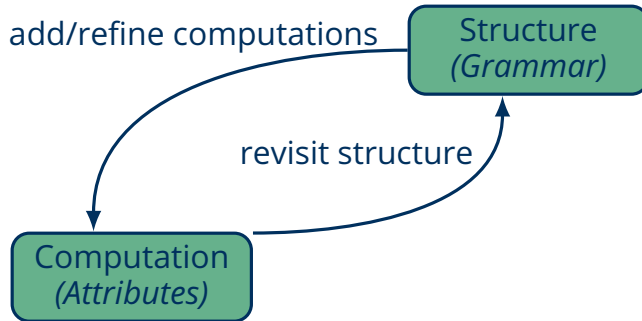
DropOffLocation : Object ::= ;
MovableObject : Object ::= ;

abstract Operation ;
PickAndPlace : Operation ::= ;

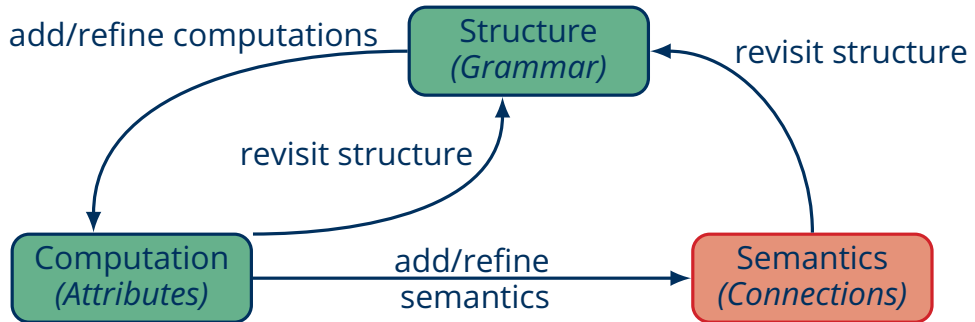
rel PickAndPlace.ObjectToPick -> MovableObject;
rel PickAndPlace.TargetLocation -> DropOffLocation;
```



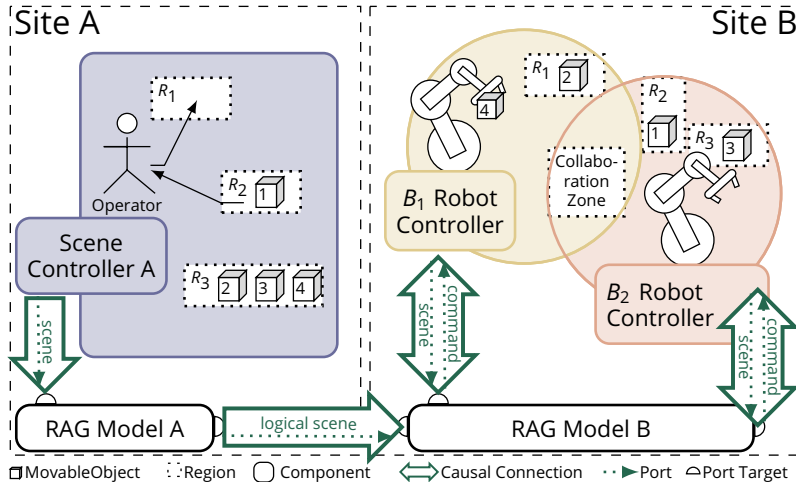
An Iterative Process to Build Self-Adaptive Systems



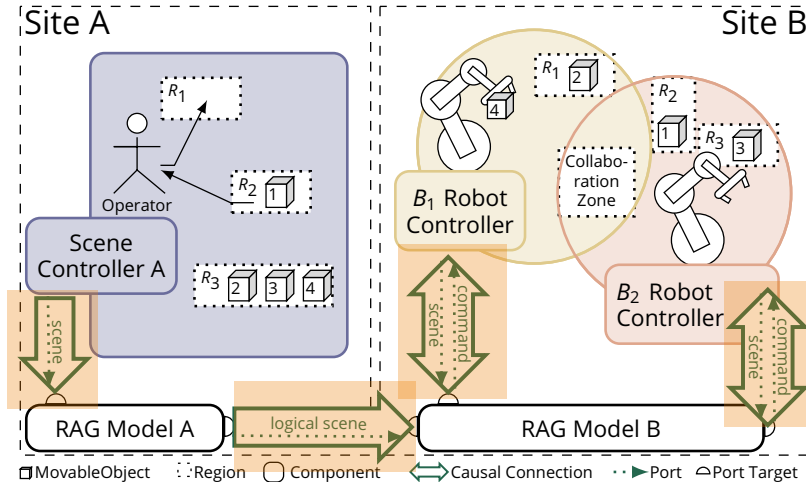
An Iterative Process to Build Self-Adaptive Systems



Coming Back to our Case Study

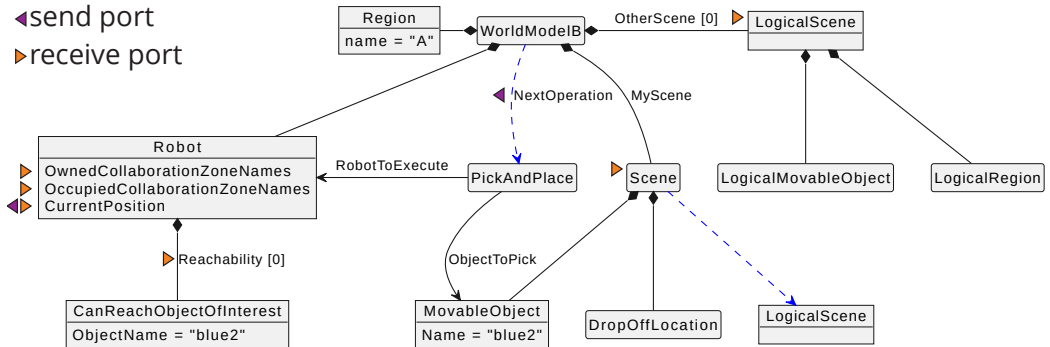


Coming Back to our Case Study



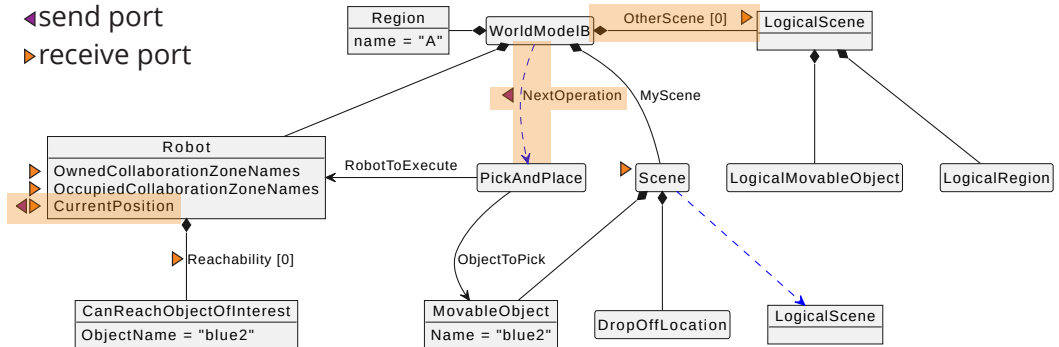
Ports Mark Required Connections for our Case Study

- ◀ send port
- ▶ receive port



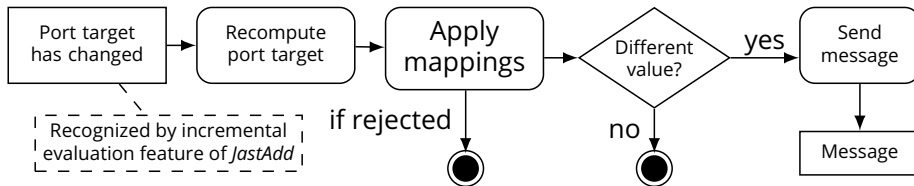
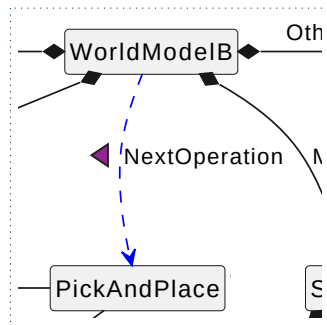
Ports Mark Required Connections for our Case Study

- ◀ send port
- ▶ receive port



A DSL to Specify Connections

```
send WorldModelB.NextOperation using PrintOperation ;  
PrintOperation maps Operation op to byte[] {:  
  byte[] result = op.toProtobufByteArray();  
  if (result == null) { reject(); }  
  return result;  
:}
```

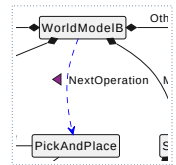
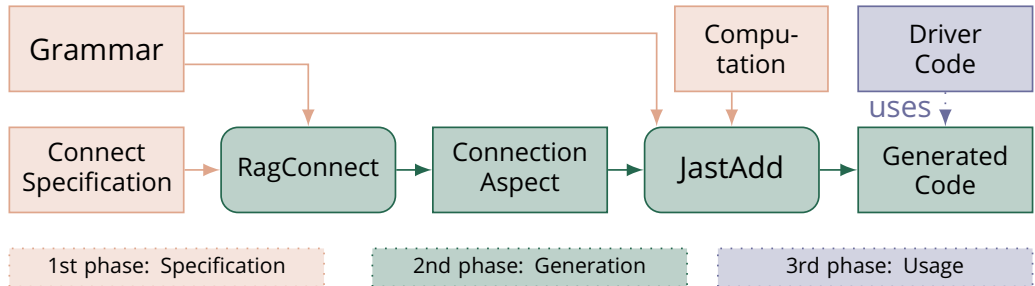


Supported Grammar Elements of the DSL

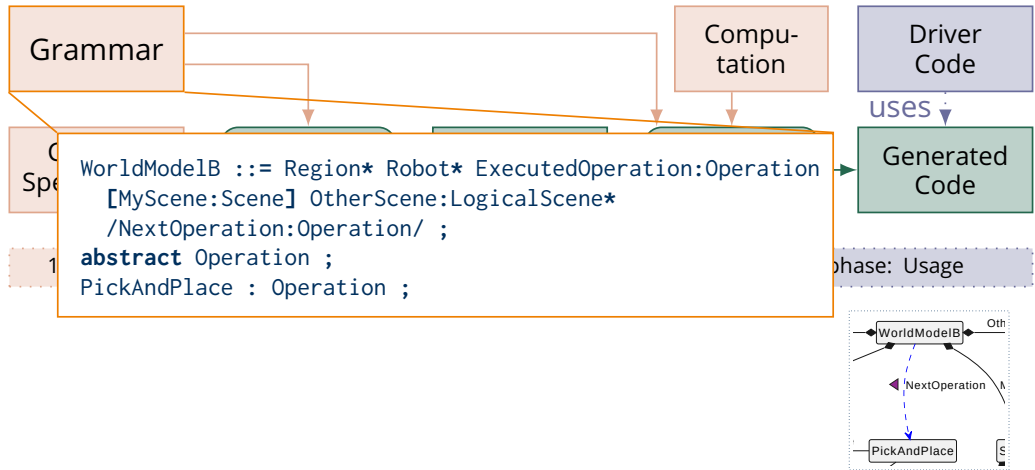
Support for Different Parts of the Grammar Using Send and Receive Ports. Legend: ● Full support, ◐ Partly supported, ○ Not supported, / not applicable

Element of a grammar	Send	Receive
Terminal	●	●
Non-Terminal (Context-free)	●	●
Non-Terminal (Single)	●	●
Non-Terminal (List)	●	●
Non-Terminal (Optional)	●	●
Relation	◐	○
Attribute	●	/

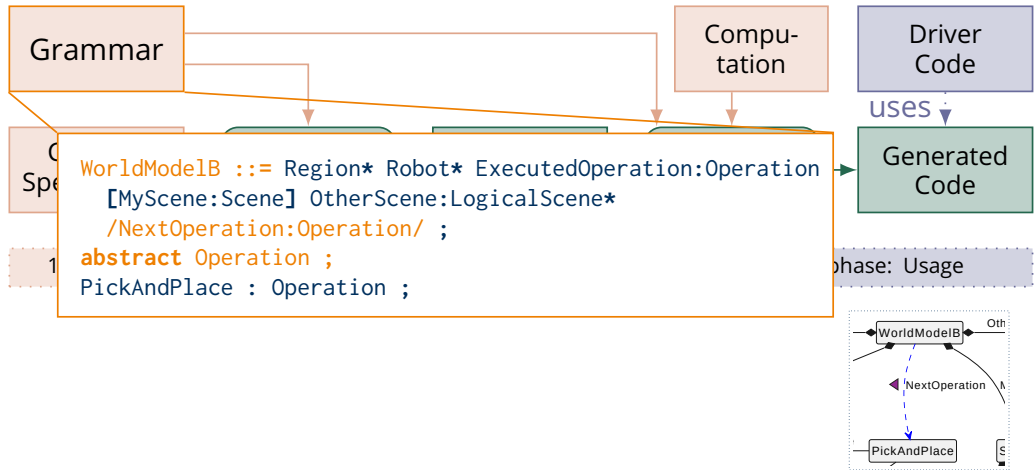
Detailed Generation Process



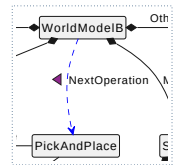
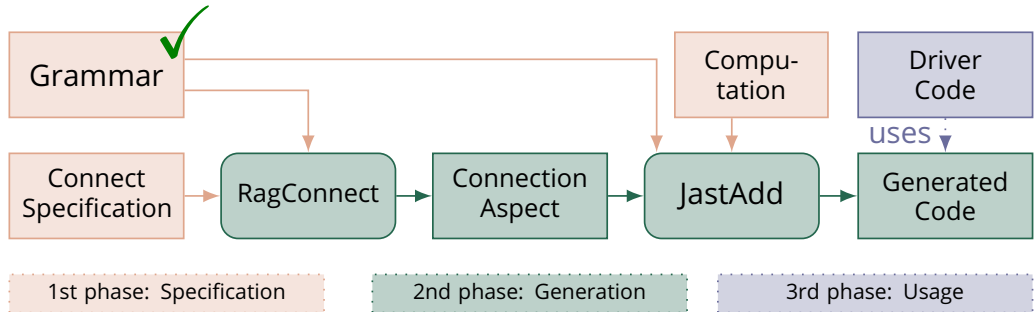
Detailed Generation Process



Detailed Generation Process



Detailed Generation Process



Detailed Generation Process

```
syn Operation WorldModelB.getNextOperation() {  
  if (diffToOperations().getNumChild() == 0)  
    return errorNoOperationComputed();  
  for (Operation op : diffToOperations()) {  
    Robot executingRobot = op.getRobotToExecute();  
    if (!op.isErrorOperation()) {  
      if (op.equals(lastOperationFor(executingRobot))) {  
        return errorDuplicateOperation();  
      }  
      return op;  
    }  
  }  
  return errorNoExecutableOperation();  
}
```

Compu-
tation

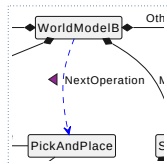
Driver
Code

uses

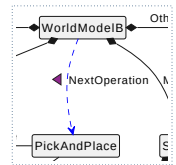
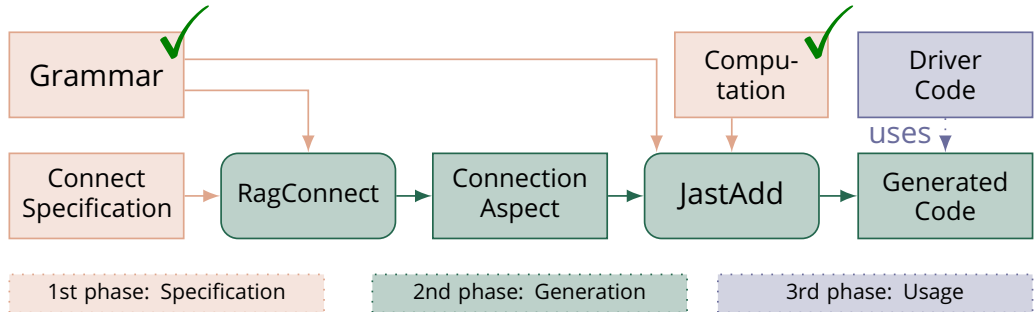
JastAdd

Generated
Code

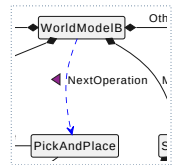
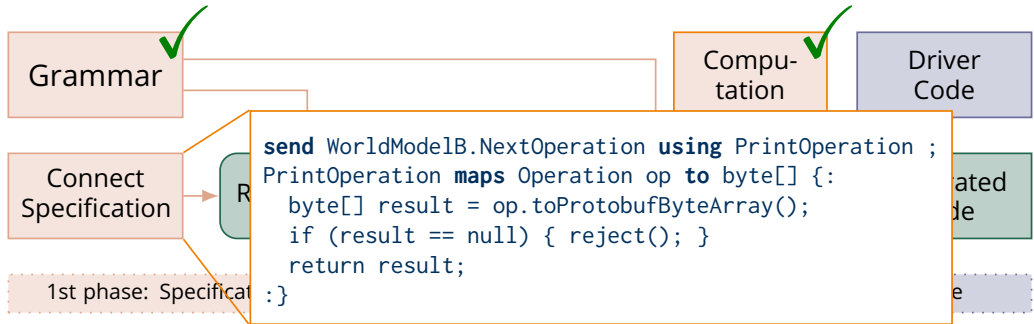
3rd phase: Usage



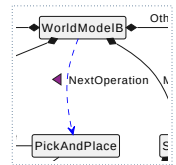
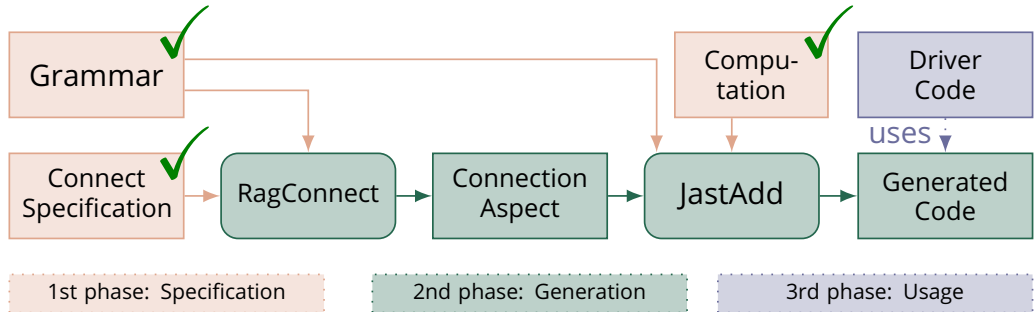
Detailed Generation Process



Detailed Generation Process



Detailed Generation Process



Detailed Generation Process

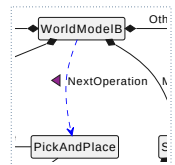
```
WorldModelB model = new WorldModelB();  
model.ragconnectResetEvaluationCounter();  
model.addOtherScene(new LogicalScene());  
  
for (String topic : config.forB.topicsSceneUpdate) {  
    model.connectMyScene(topic);  
}  
  
model.connectOtherScene("place-a/logical/update", 0);  
model.connectNextOperation("robot/operation", false);
```

Driver Code

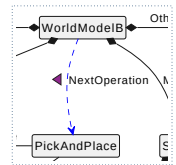
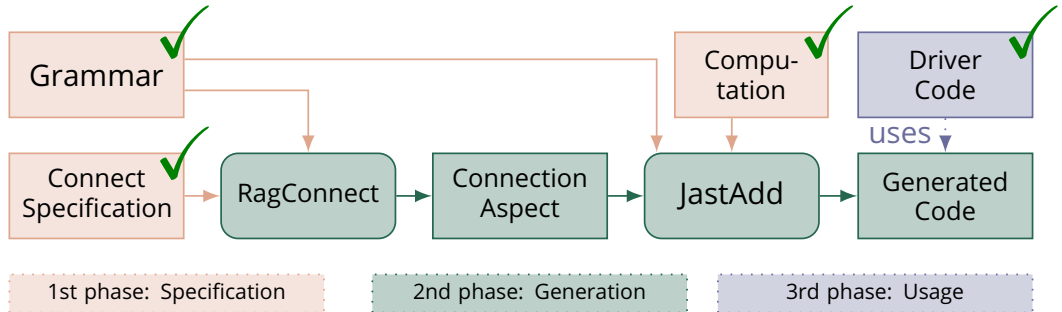
uses

Generated Code

3rd phase: Usage



Detailed Generation Process



Evaluation of Concisness

LOC of all Case Studies					
Case Study	Part	Grammar	Attributes	Connect	Java
Main	Common (manual)	15	255	52	318
	Site A (manual)	1	17	2	136
	Site A (generated)	31	2570		11 042
	Site B (manual)	29	668	136	350
	Site B (generated)	69	5443		23 075
Coordinator	Main (manual)	15	261	8	144
	Main (generated)	28	1510		9592

Evaluation of Concisness

LOC of all Case Studies					
Case Study	Part	Grammar	Attributes	Connect	Java
Main	Common (manual)	15	255	52	318
	Site A (manual)	1	17	2	136
	Site A (generated)	31	2570		11 042
	Site B (manual)	29	668	136	350
	Site B (generated)	69	5443		23 075
Coordinator	Main (manual)	15	261	8	144
	Main (generated)	28	1510		9592

Conclusion

This work presents

- a way to connect RAG-based models with external systems
- **easily**, with a DSL
- **efficiently**, exploiting RAG technology

We showed this

- in a robot coordination case study
- ... for solving a cooperative task
- ... *and* for coordinating the runtime components.

We provide the missing concepts to use RAGs for Models@run.time!

Conclusion

This work presents

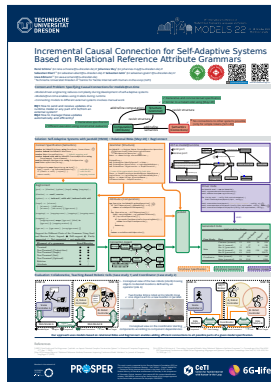
- a way to connect RAG-based models with external systems
- **easily**, with a DSL
- **efficiently**, exploiting RAG technology

We showed this

- in a robot coordination case study
- ... for solving a cooperative task
- ... *and* for coordinating the runtime components.

We provide the missing concepts to use RAGs for Models@run.time!

Visit us at our poster!



References I

- [Hed00] Görel Hedin. “Reference attributed grammars”. In: *Informatica (Slovenia)* 24.3 (2000), pp. 301–317. URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.108.8792&rep=rep1&type=pdf> (visited on 08/03/2015).
- [HM03] Görel Hedin and Eva Magnusson. “JastAdd: an aspect-oriented compiler construction system”. In: *Science of Computer Programming. Special Issue on Language Descriptions, Tools and Applications (L DTA'01)* 47.1 (Apr. 2003), pp. 37–58. ISSN: 0167-6423. DOI: 10.1016/S0167-6423(02)00109-0. URL: <http://www.sciencedirect.com/science/article/pii/S0167642302001090> (visited on 10/01/2019).
- [Knu68] Donald E. Knuth. “Semantics of context-free languages”. In: *Mathematical systems theory* 2.2 (1968), pp. 127–145. URL: <http://link.springer.com/article/10.1007/BF01692511> (visited on 08/03/2015).

References II

- [Mey+20] Johannes Mey et al. “Relational Reference Attribute Grammars: Improving Continuous Model Validation”. en. In: *Journal of Computer Languages* 57 (Jan. 2020), p. 21. ISSN: 2590-1184. DOI: 10.1016/j.co1a.2019.100940. URL: <http://www.sciencedirect.com/science/article/pii/S2590118419300656> (visited on 02/14/2020).
- [Wey19] Danny Weyns. “Software Engineering of Self-adaptive Systems”. en. In: *Handbook of Software Engineering*. Cham: Springer International Publishing, 2019, pp. 399–443. ISBN: 978-3-030-00262-6. URL: https://doi.org/10.1007/978-3-030-00262-6_11.

RagConnect DSL // Grammar

$\langle port \rangle ::= \langle direction \rangle \langle options \rangle \langle target \rangle \textbf{using} \langle mappings \rangle ;$

$\langle direction \rangle ::= \textbf{send} \mid \textbf{receive}$

$\langle options \rangle ::= \epsilon \mid \textbf{indexed} \mid \textbf{with add} \mid \textbf{indexed with add}$

$\langle target \rangle ::= \langle nt-name \rangle$
| $\langle nt-name \rangle . \langle child-name \rangle$
| $\langle nt-name \rangle . \langle attr-name \rangle (\langle type-name \rangle)$

$\langle mappings \rangle ::= \langle mapping-name \rangle , \langle mappings \rangle \mid \langle mapping-name \rangle$

$\langle mapping \rangle ::=$
 $\langle mapping-name \rangle \textbf{maps} \langle type-use \rangle \langle ident \rangle \textbf{to}$
 $\langle type-use \rangle \{ \langle mapping-content \rangle : \}$

$\langle type-use \rangle ::= \langle type-name \rangle \mid \langle array-type-name \rangle []$