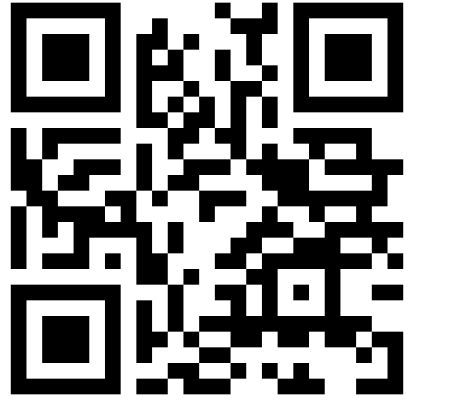
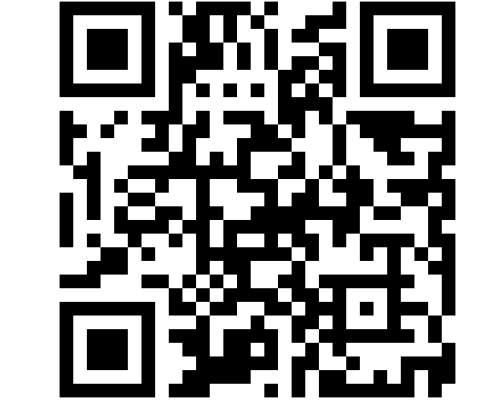


Incremental Causal Connection for Self-Adaptive Systems Based on Relational Reference Attribute Grammars

René Schöne* (✉ rene.schoene@tu-dresden.de) // Johannes Mey* (✉ johannes.mey@tu-dresden.de) // Sebastian Ebert** (✉ sebastian.ebert@tu-dresden.de) // Sebastian Götz* (✉ sebastian.goetz1@tu-dresden.de) // Uwe Aßmann** (✉ uwe.assmann@tu-dresden.de)

* Technische Universität Dresden // ** Centre for Tactile Internet with Human-in-the-Loop (CeTI)



DOI 10.5281/zenodo.6963426

connect.relatinal-rags.eu

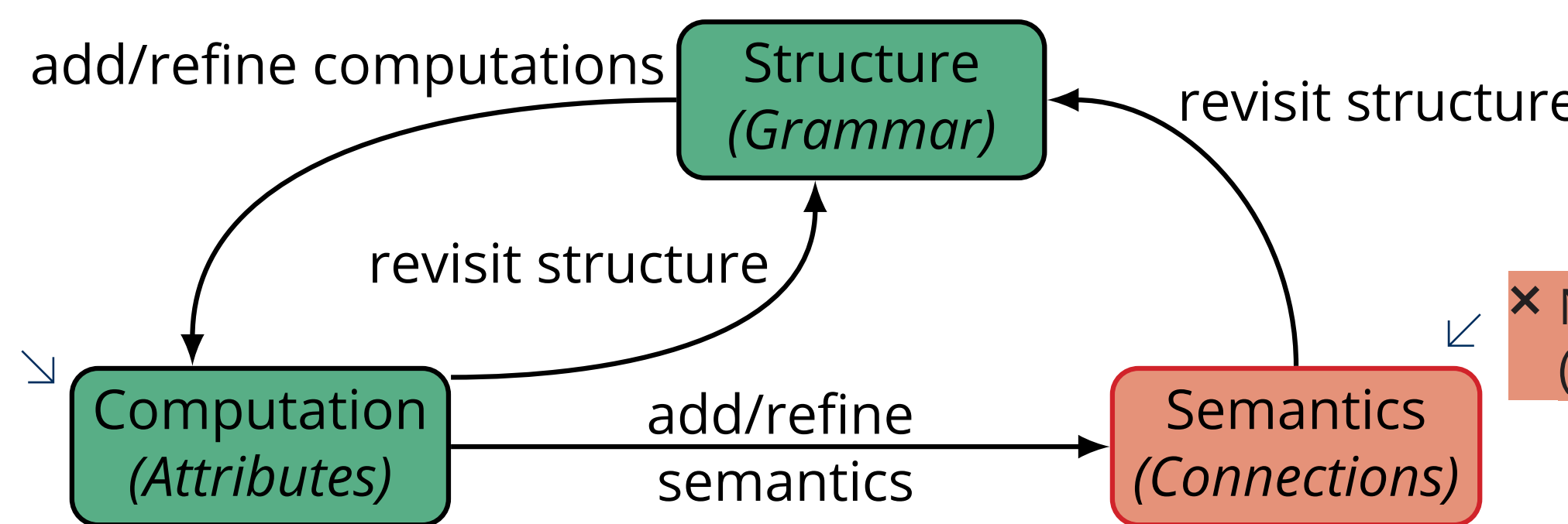
Context and Problem: Specifying Causal Connections for models@run.time

- Model-driven engineering reduces complexity during development of self-adaptive systems
- Models@run.time enables using models during runtime
- Connecting models to different external systems involves manual work

RQ1 How to send and receive updates of a runtime model or any part of it to/from an external system?

RQ2 How to manage these updates automatically and efficiently?

- ✓ Declarative specification
- ✓ Efficient execution using incremental evaluation



- ✓ Declarative and concise textual specification
- ✓ Similar to a metamodel using [Mey+20]

✗ No connections to other systems possible (only for simple tokens [Sch+20])

Solution: Self-Adaptive Systems with JastAdd [HM03] + Relational RAGs [Mey+20] + RagConnect

Connect Specification (Semantics)

```

receive WorldModelB.MyScene using ParseScene, ConvertScene ; ①
receive indexed WorldModelB.OtherScene ;
receive Robot.CurrentPosition ; // default mapping used
receive indexed with add WorldModelB.ExecutedOperation using
  ParseCommand, ConvertCommand ;
send WorldModelB.NextOperation using PrintOperation ; ②
send Robot.myPosition(String) ;

PrintOperation maps Operation op to byte[] {
  byte[] result = op.toProtobufByteArray(); // embedded java
  if (result == null) { reject(); } // code for the
  return result; // mapping content
};
  
```

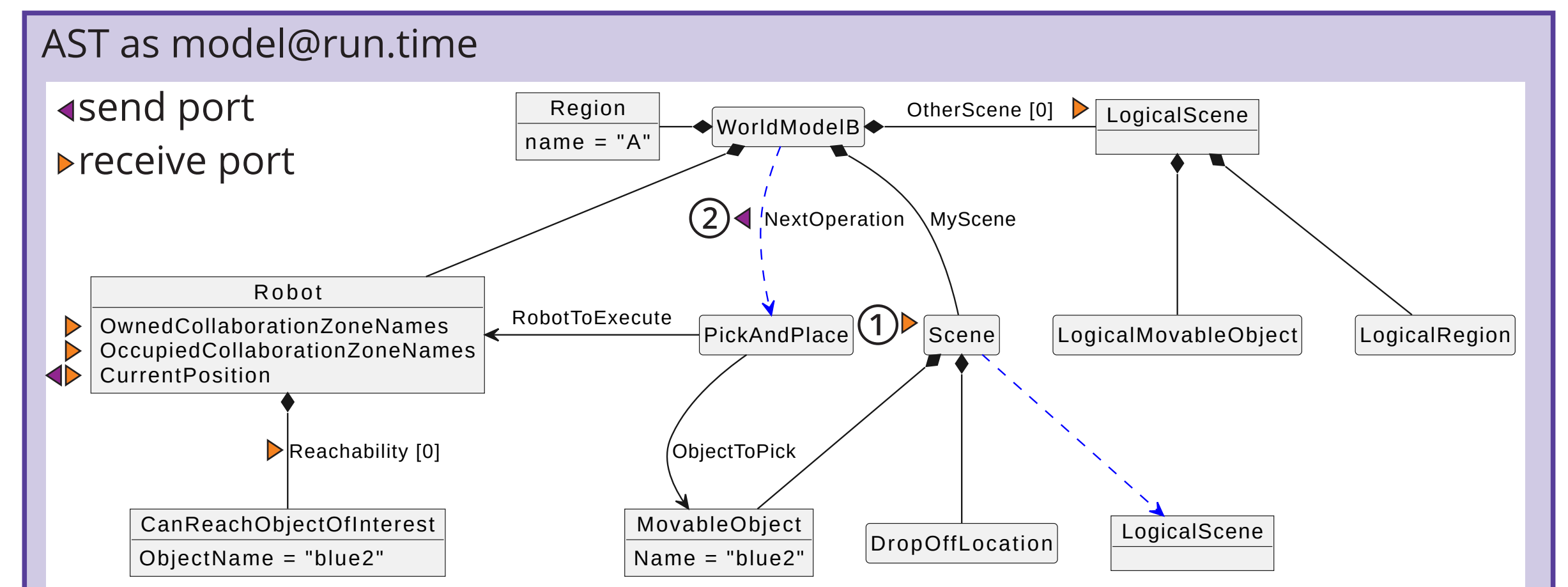
Grammar (Structure)

```

WorldModelB ::= Region* Robot* ExecutedOperation:Operation*
  ① [MyScene:Scene] OtherScene:LogicalScene*
  /NextOperation:Operation/ ; ②
Robot ::= <Name:String> <CurrentPosition> ;

PickAndPlace ;
rel PickAndPlace.RobotToExecute? -> Robot ;
rel PickAndPlace.ObjectToPick -> LogicalMovableObject ;
rel PickAndPlace.TargetLocation -> DropOffLocation ;

// some of the nonterminals shared by both sites
Scene ::= DropOffLocation* MovableObject* RobotObject* ;
LogicalScene ::= LogicalRegion* LogicalMovableObject* ;
Region ::= <Name:String> <LocationNames> ;
CollaborationZone : DropOffLocation ;
  
```



Driver Code

```

WorldModelB model = new WorldModelB();
model.ragconnectResetEvaluationCounter();
model.addOtherScene(new LogicalScene());

for (String topic : config.forB.topicsSceneUpdate) {
  model.connectMyScene(mqttUri(topic, config));
}

model.connectOtherScene(mqttUri("place-a/logical/update", config), 0); ①
model.connectNextOperation(mqttUri(config.forB.topicCommand, config), false); ②
model.connectExecutedOperation(mqttUri(config.forB.topicCommand, config));
  
```

RagConnect

```

<port> ::= <direction> <options> <target> using <mappings> ;
<direction> ::= send | receive
<options> ::= ε | indexed | with add | indexed with add
<target> ::= <nt-name>
  | <nt-name> . <child-name>
  | <nt-name> . <attr-name> ( <type-name> )
<mappings> ::= <mapping-name> , <mappings> | <mapping-name>
<mapping> ::=
  <mapping-name> maps <type-use> <ident> to
  <type-use> { <mapping-content> } ;
<type-use> ::= <type-name> | <array-type-name> []

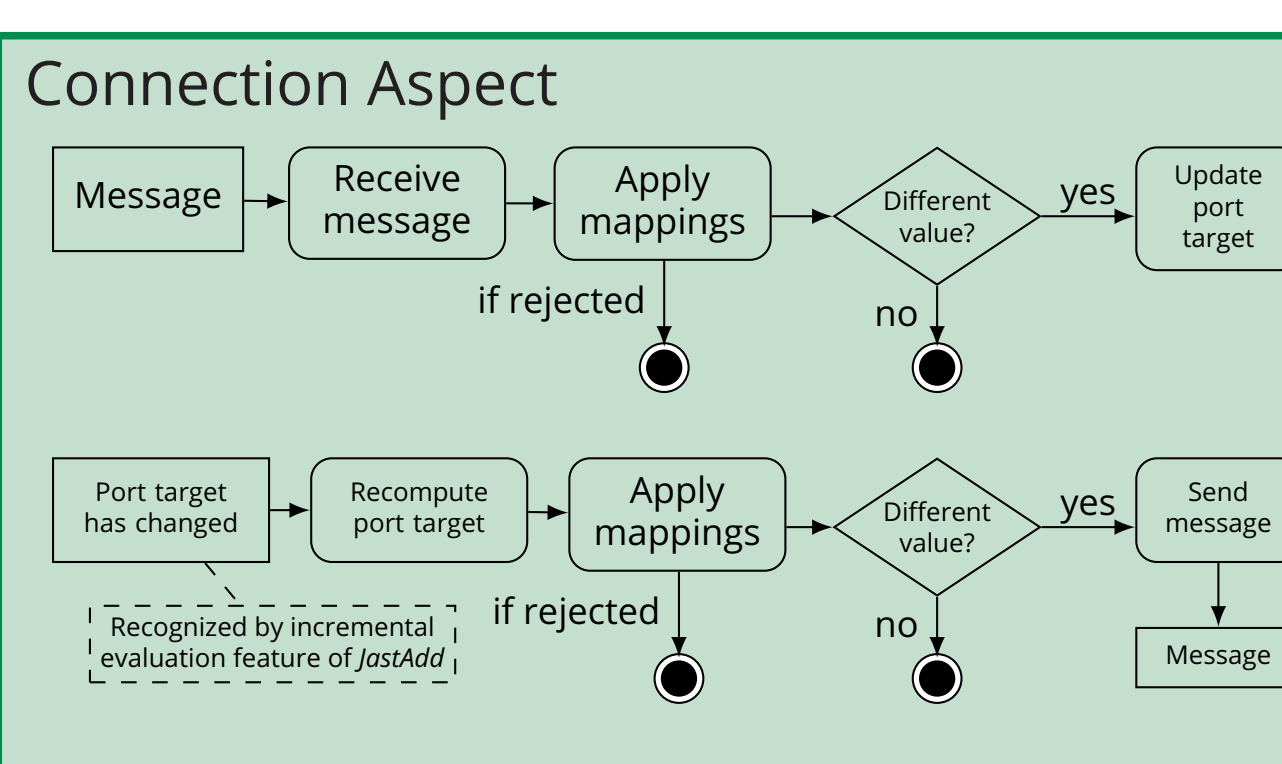
Support for Different Parts of the Grammar Using Send and Receive Ports. Legend: ● Full support, ○ Partly supported, / Not supported, / not applicable
  
```

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

Attributes (Computation)

```

syn Operation WorldModelB.getNextOperation() { ②
  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();
}
  
```

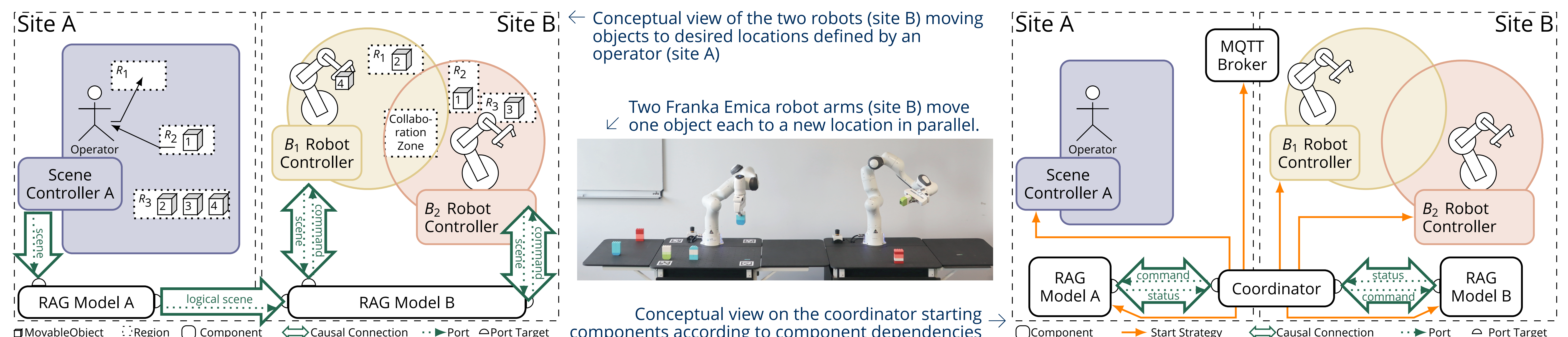


JastAdd

Generated Code					
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

1st phase: Specification 2nd phase: Generation 3rd phase: Usage

Evaluation: Collaborative, Teaching-Based Robotic Cells (Case study 1) and Coordinator (Case study 2)



Our approach uses models based on relational RAGs and RagConnect enables adding efficient connections to all possible parts of a given model specification.

References

- [HM03] Görel Hedin and Eva Magnusson. "JastAdd—an Aspect-Oriented Compiler Construction System". In: Science of Computer Programming 47,1 (Apr. 1, 2003).
- [Mey+20] Johannes Mey et al. "Relational Reference Attribute Grammars: Improving Continuous Model Validation". In: Journal of Computer Languages 57 (2020).
- [Sch+20] René Schöne et al. "Connecting Conceptual Models Using Relational Reference Attribute Grammars". In: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings. New York, NY, USA: ACM, Oct. 16, 2020, pp. 1–11.