



A Case Study of the Development of an Agent-based Simulation in the Traffic Signal Control Domain using an MDD Approach

Fernando Santos^{1,2}, Ingrid Nunes^{1,3}, Ana L.C. Bazzan¹

¹ Instituto de Informática, UFRGS, Brazil

² Depto. de Engenharia de Software, UDESC, Brazil

³ TU Dortmund, Germany

{fsantos, ingridnunes, bazzan}@inf.ufrgs.br

May 8, 2017



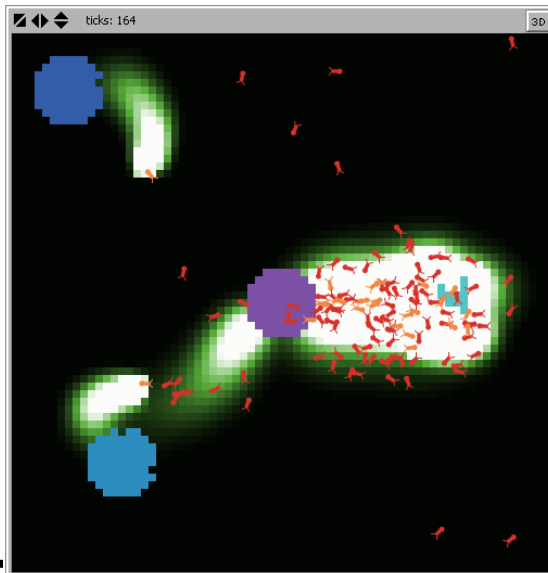
Prosoft



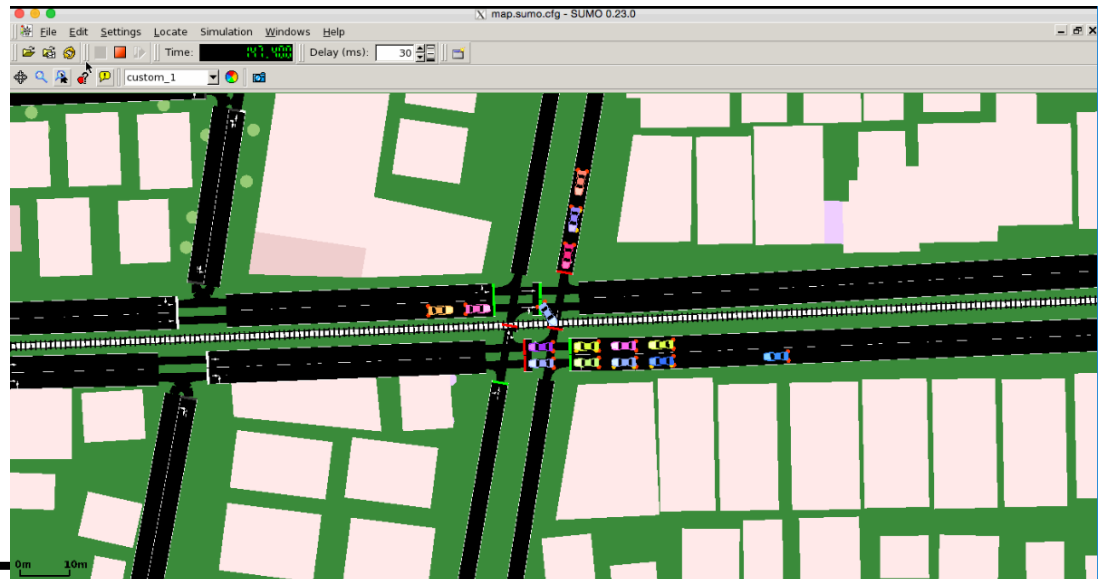
ABMS



- Agent Based Modeling and Simulation (ABMS)
 - **Paradigm** that uses **agent-based simulations** to [re]produce, analyze, or predict a phenomenon under study
 - Particularly suitable for **complex systems** and **emergent phenomena**

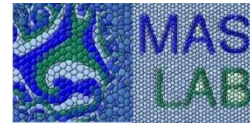


Ant foraging



Traffic

Agent-based Simulation Development



- Agent-based Simulation (ABS) Platforms



M A S O N



GAMA PLATFORM



SeSAM

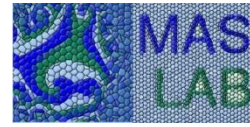
Shell for Simulated Multi-Agent Systems

v1.3 Jan2017



- Demand for technical expertise
- No **building blocks** for recurrent concepts and behaviors

Model-driven Development (MDD)



- Models as *first-class citizens*
- Trades *generality* for *expressiveness*
 - Domain concepts are available for modeling
 - Reduces the abstraction gap and increases productivity
- MDD in ABMS: **Limitations**
 - Limited to particular MDD aspects
 - Metamodels: AMASON, MAIA, metamodel of Ribino *et al.* (2014)
 - Methods for identifying certain domain concepts: easyAMBS, IODA
 - Much is left to be **developed manually**
 - Often, only code skeletons are produced automatically
 - **Lack of evidence of the real benefits** that an MDD approach can promote for developing agent-based simulations.

Research Question



*What is the **benefit** an MDD approach can provide for developing agent-based simulations?*

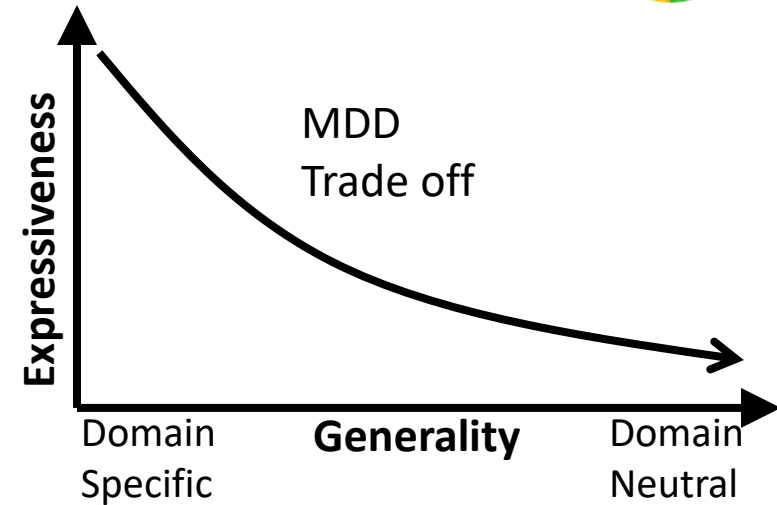
- We explore this question through a case study
 - Provision of an MDD approach for ABMS
 - Empirical assessment of its gains
 - Development effort

MDD for ABMS: preliminaries



- The more specific the application domain, the higher the chance of success

- J. Hutchinson, M. Rouncefield, and J. Whittle. Model-driven engineering practices in industry. In *International Conference on Software Engineering*, 633-642, 2011.



- Selected domain
 - **Agent-based simulations** of Adaptive Traffic Signal Control (ATSC)

Adaptive Traffic Signal Control



ATSC

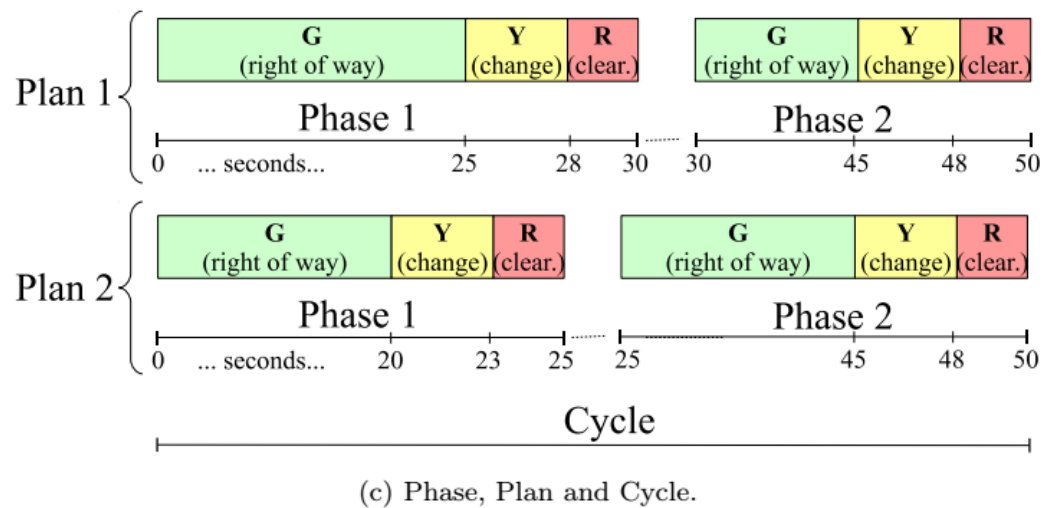
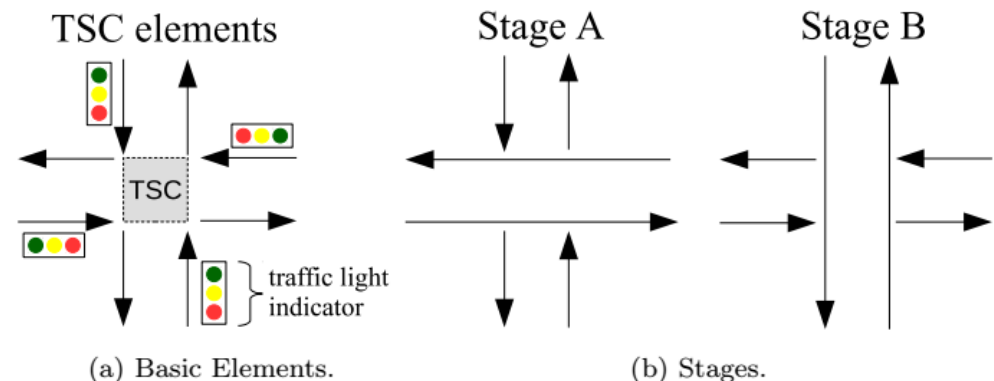
- Traffic Signal Control (TSC)
agents in charge of
managing traffic light
indicators so as to:
 - Maximize traffic flow
 - Minimize travel time
 - Other metrics
- Distributed and
autonomous agents
- Availability of decision-
making techniques



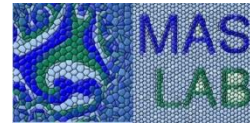
ATSC elementary concepts



- TSC agents
- Incoming/outgoing lanes
- Traffic light indicators
- Stages
- Phases
- Plans
- Cycle



MDD for ABMS: elements



1. Metamodel

- Domain concepts → meta-entities and their relationship
- Built through a domain analysis activity

2. Domain-specific language (DSL)

- Building blocks for recurrent concepts → expressiveness
- Allow modeling in an expressive way (abstraction gap)

3. Transformations & Code Generation

- Rules for producing code automatically → productivity

Domain Analysis Method



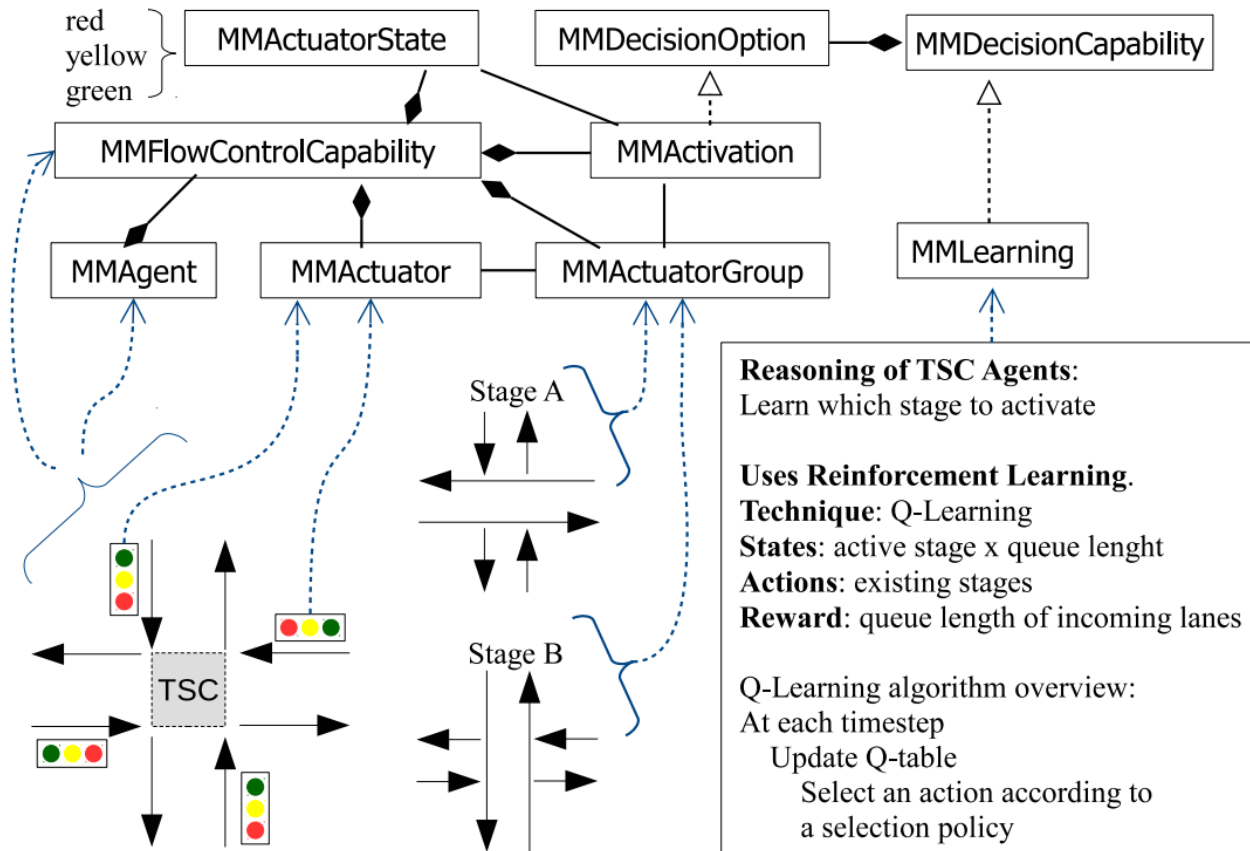
- Bottom-up, based on existing simulations
 1. Build a preliminary list of agent-related concepts following the steps of existing methodologies for ABMS
 2. Refine the identified concepts using the ODD protocol
 - Adaptation, learning, collectives, ...
 3. Find the essence behind each identified concept
 - Recurrent characteristics and behaviors
 4. Build the domain model

Domain Analysis

Step 3



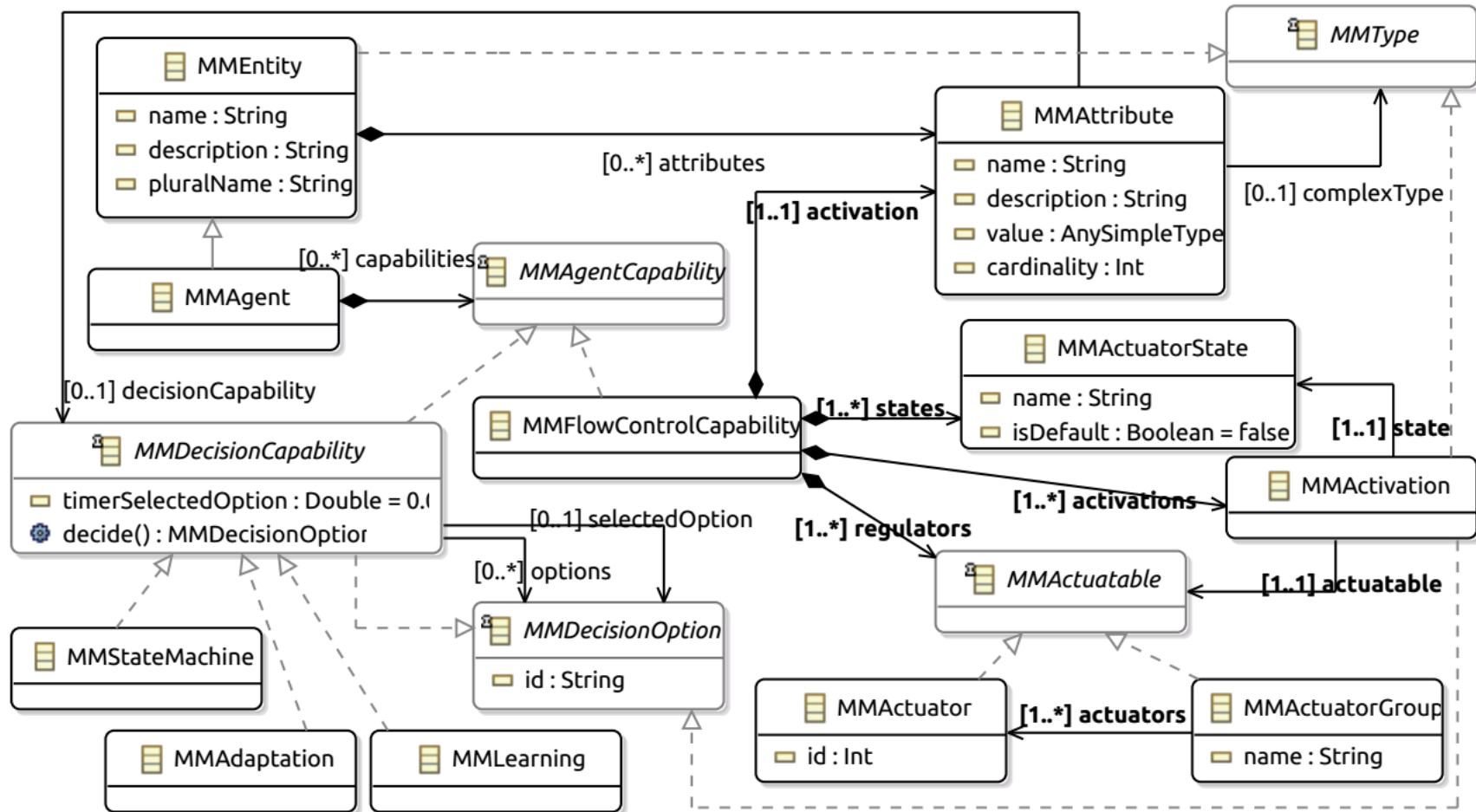
- How domain concepts were abstracted:



Metamodel



Step 4



- Also, acts as the DSL abstract syntax

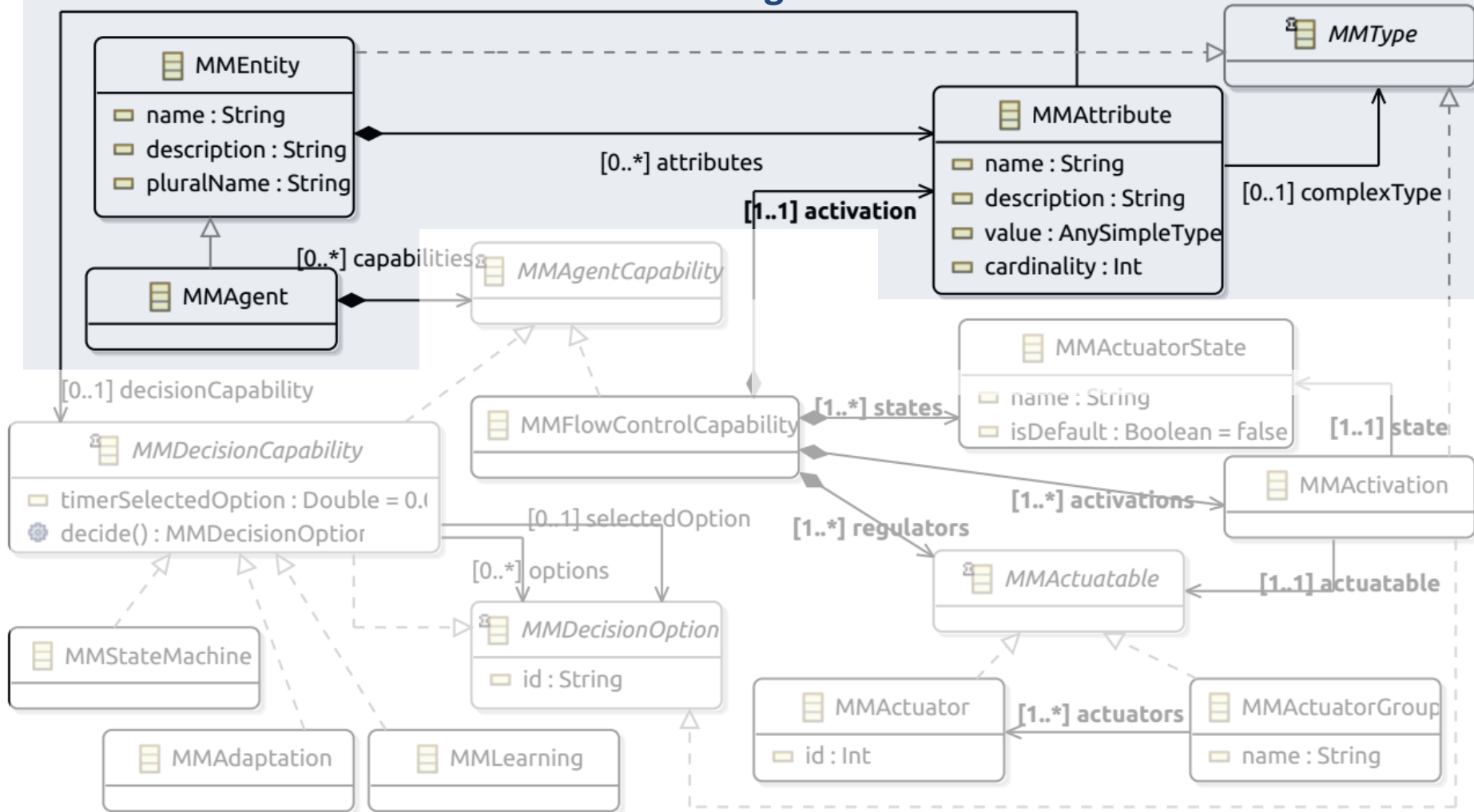
Metamodel



Step 4



Entities & Agents

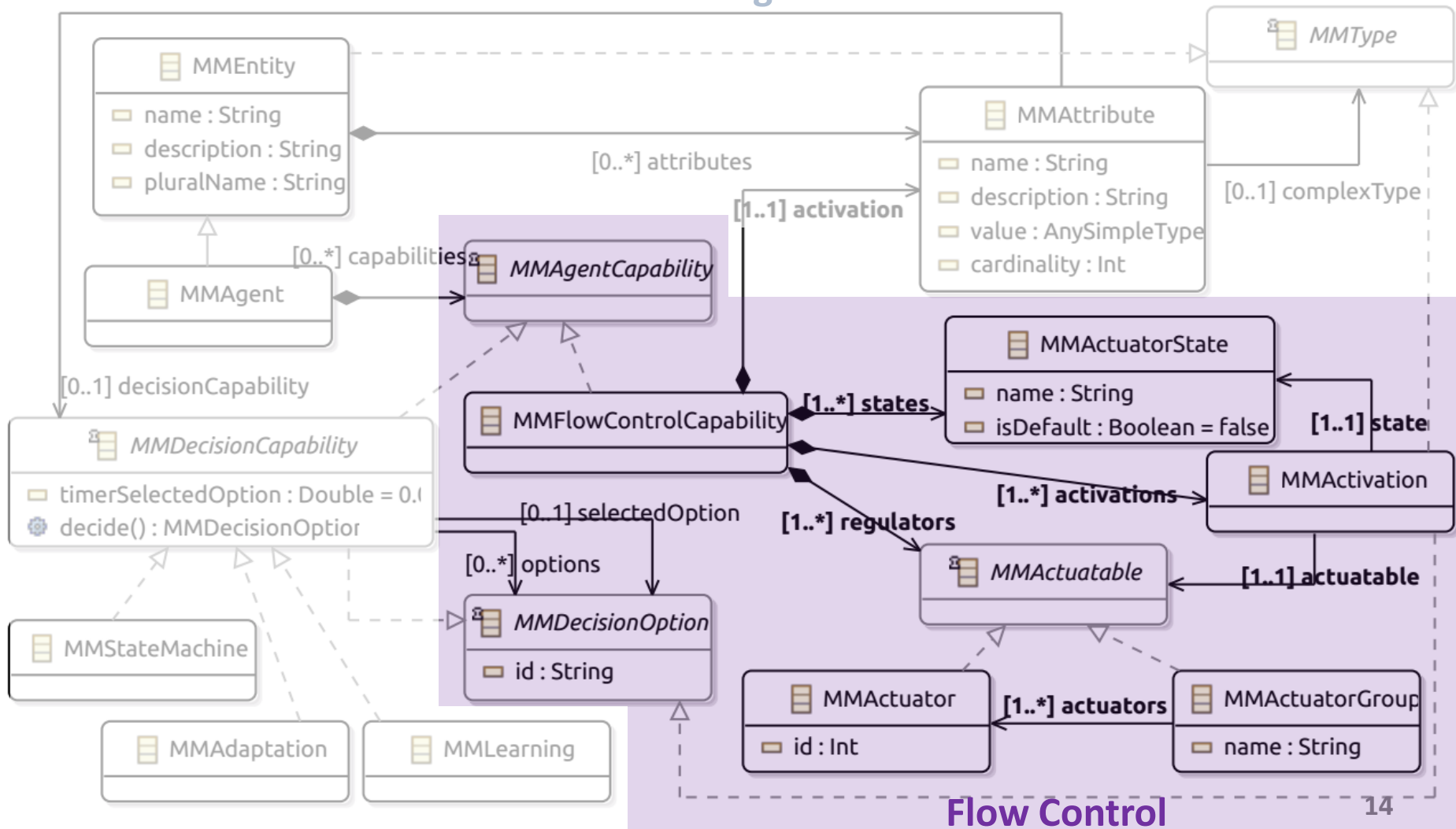


Metamodel

Step 4



Entities & Agents



MAS
LAB



```

classDiagram
    class MMEntity {
        name : String
        description : String
        pluralName : String
    }
    class MMAttribute {
        name : String
        description : String
        value : AnySimpleType
        cardinality : Int
    }
    class MMAgent {
    }
    class MMDecisionCapability {
        timerSelectedOption : Double = 0.1
        decide() : MMDecisionOption
    }
    class MMFlowControlCapability {
    }
    class MMActuatorState {
        name : String
        isDefault : Boolean = false
    }
    class MMActivation {
    }
    class MMActuable {
    }
    class MMActuator {
        id : Int
    }
    class MMActuatorGroup {
        name : String
    }
    class MMDecisionOption {
        id : String
    }
    class MMAdaptation {
    }
    class MMLearning {
    }
    class MMStateMachine {
    }

    MMEntity "0..*" --> "0..*" MMAttribute : attributes
    MMEntity <|-- MMAgent
    MMAgent "0..*" --> "0..*" MMDecisionCapability : capabilities
    MMDecisionCapability "0..1" --> "0..1" MMDecisionOption : selectedOption
    MMDecisionCapability "0..*" --> "0..*" MMDecisionOption : options
    MMFlowControlCapability <|-- MMDecisionOption
    MMFlowControlCapability "1..1" --> "1..1" MMActuatorState : states
    MMActuatorState "1..1" --> "1..1" MMActivation : state
    MMActivation "1..*" --> "1..*" MMActuable : activations
    MMActuable "1..1" --> "1..1" MMActuator : actuators
    MMActuable "1..1" --> "1..1" MMActuatorGroup : actuators
    MMActuatorGroup "1..*" --> "1..*" MMActuator : actuators
    MMActuatorGroup <|-- MMActuator
    MMActuator <|-- MMAdaptation
    MMActuator <|-- MMLearning
    MMStateMachine <|-- MMDecisionCapability
    MMStateMachine <|-- MMFlowControlCapability
    MMStateMachine <|-- MMActuatorState
    MMStateMachine <|-- MMActivation
    MMStateMachine <|-- MMActuable
    MMStateMachine <|-- MMActuatorGroup
    MMStateMachine <|-- MMAdaptation
    MMStateMachine <|-- MMLearning
  
```

Flow Control

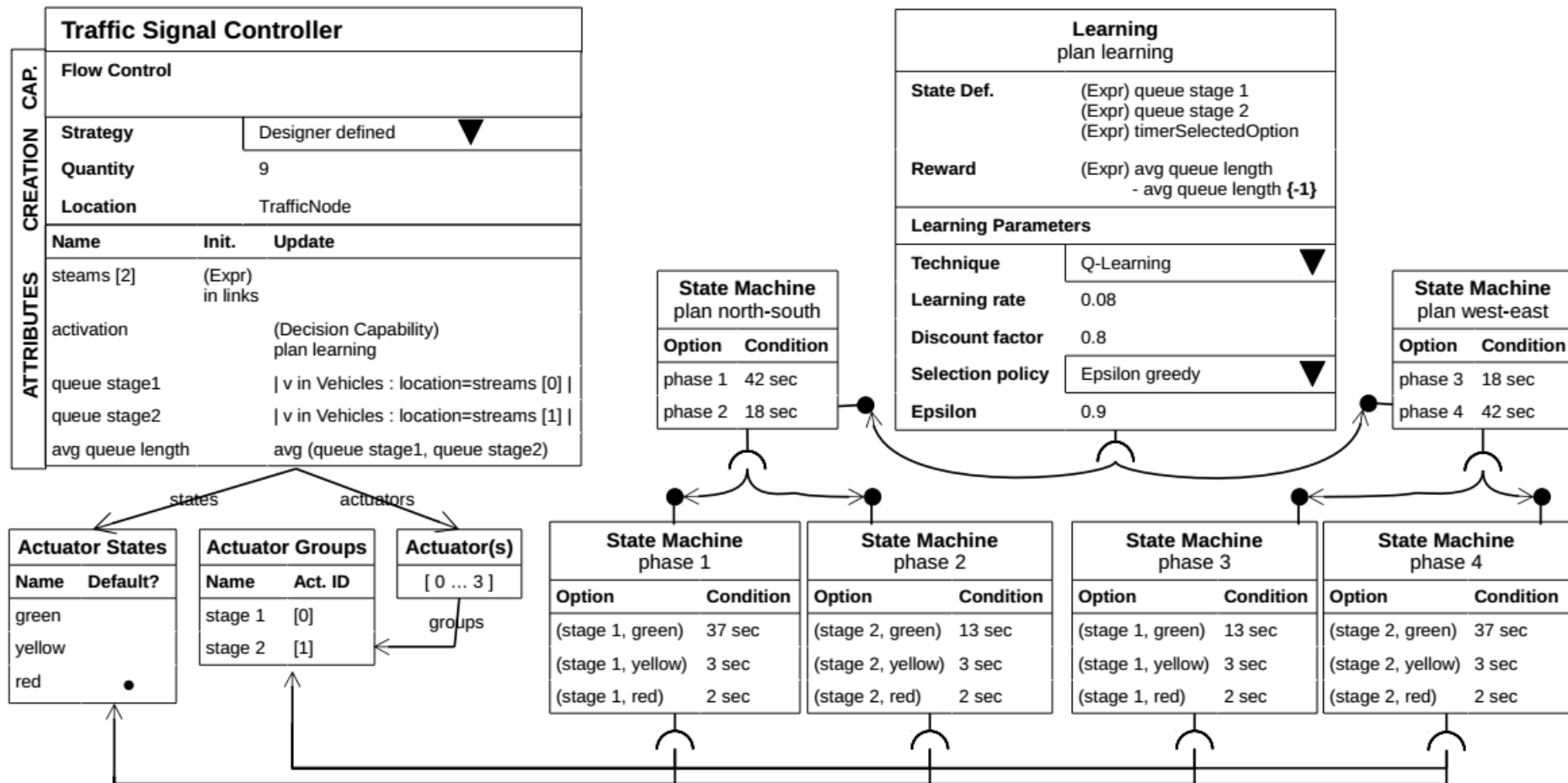
15

DSL4ABMS: Modeling Language

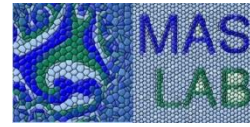
Concrete Syntax



- UML-inspired building blocks for ABMS elements



Model-to-code Transformations



- Production rules to generate NetLogo code. E.g.:

Production Rule	Transformations to NetLogo Code
Agent type	For each MMAgent → breed
RL capability	For each MMReinforcementLearning → • Qlearning init. • Qlearning reward def. • Qlearning update Qtable • Qlearning decision
Qlearning update QTable	executes Qlearning reward def. reporter to compute reward set statement for updating the Qtable $Q(s, a) = Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$

- Specified and implemented using XPand 

Evaluation: Questions



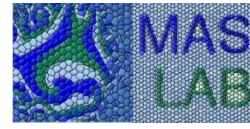
- Q1. MDD Effectiveness

Is our MDD approach able to produce **ready-to-run code** from DSL4ABMS models?

- Q2. MDD Benefits

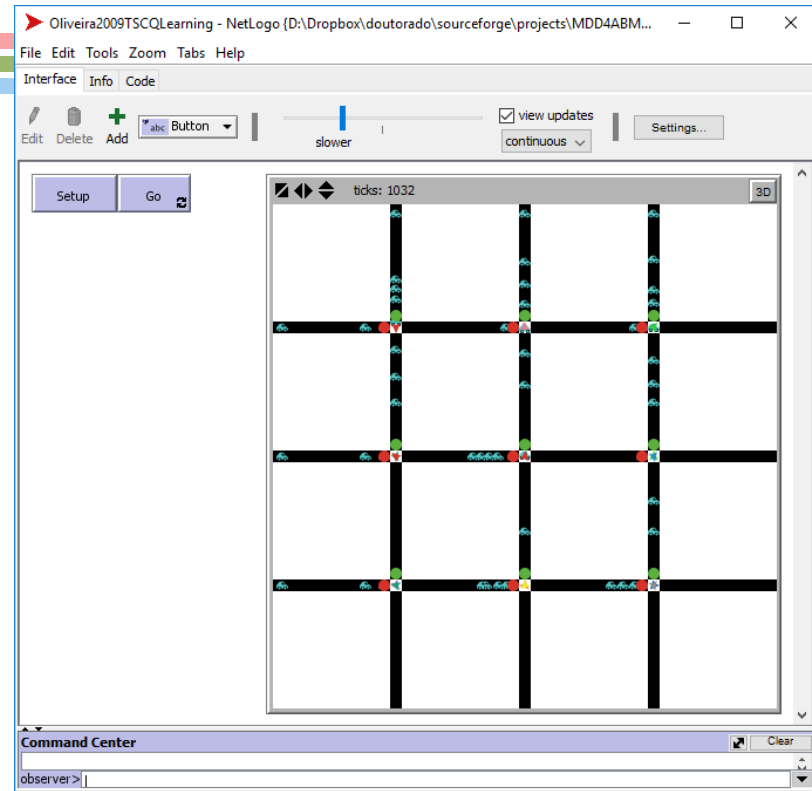
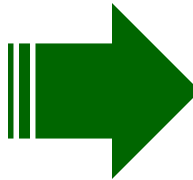
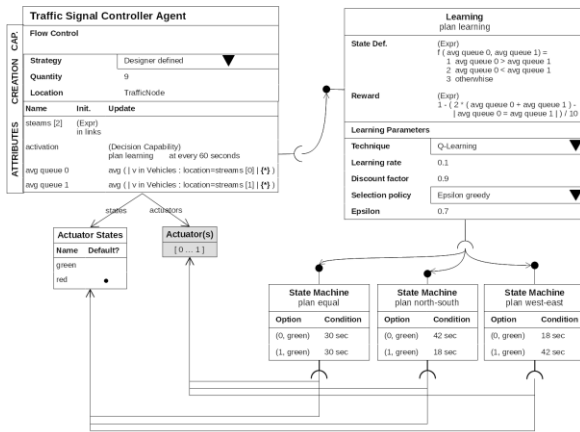
Does our MDD approach decrease the **effort** to develop agent-based simulations?

Evaluation: Selected Simulations



- One simulation was selected for each decision capability
- Fixed plans → State machines
 - U. Wilensky. NetLogo Traffic Grid model. 2003
<http://ccl.northwestern.edu/netlogo/models/TrafficGrid>.
- Self-organizing Traffic Lights → Adaptation
 - C. Gershenson. Self-organizing traffic lights. Complex Systems. 2005.
- Signal Plan Learning → Reinforcement Learning
 - D. de. Oliveira and A. L. C. Bazzan. Multiagent learning on traffic lights control: effects of using shared information. In Multi-Agent Systems for Traffic and Transportation,. 2009

Q1. Generated Simulations



- TSC-related ready-to-run code was fully automatically generated

```
to rl-plan_learning-update-qtable [s a s' r]
  let q_s_a 0
  let q_s'_max_a_value 0
  if table:length s_values_map > 0 and table:has-key? s_values_map a
    set q_s_a table:get s_values_map a
  ]

  if table:length s'_values_map > 0 [
    set q_s'_max_a_value table:get s'_values_map max-action-value
  ]

  let new_q_s_a ( q_s_a + rl_plan_learning_alpha *
    ( r + rl_plan_learning_gamma *
      q_s'_max_a_value - q_s_a ) )

  table:put s_values_map a new_q_s_a
end
```



Q2. Development Effort

- **Size metrics**, from cost estimation methods in SE
- **Lines of Code (LoCs)** is a key size measurement
 - Used in methods such as Function Points and COCOMO
 - Can be manually produced (MLoCs) or generated (GLoCs)
- **Atomic Model Element (AMEs)**
 - Used to measure graphical models
 - 1 AME *is a visual element that is equivalent to 1 LoC*
 - Counting rules are detailed in the paper

Q2. Results



Simulation	AMEs	MLoCs	Effort*	GLoCs
Fixed Plan / State Machine				
NetLogo	1	34	35	1
Our MDE approach	14	0	14	116
Self-organizing Traffic Lights / Adaptation				
NetLogo	1	82	83	1
Our MDE approach	15	0	15	103
Signal Plan Learning / Reinforcement Learning				
ITSUMO	0	257	257	0
Our MDE approach	37	0	37	297

↓ **60.0%**

↓ **81.9%**

↓ **85.6%**

*Effort = AMEs + MLoCs

Our MDE approach reduces the effort

Q2. Results



Simulation	AMEs	MLoCs	Effort*	GLoCs
Fixed Plan / State Machine				
NetLogo	1	34	35	1
Our MDE approach	14	0	14	116
Self-organizing Traffic Lights / Adaptation				
NetLogo	1	82	83	1
Our MDE approach	15	0	15	103
Signal Plan Learning / Reinforcement Learning				
ITSUMO	0	257	257	0
Our MDE approach	37	0	37	297

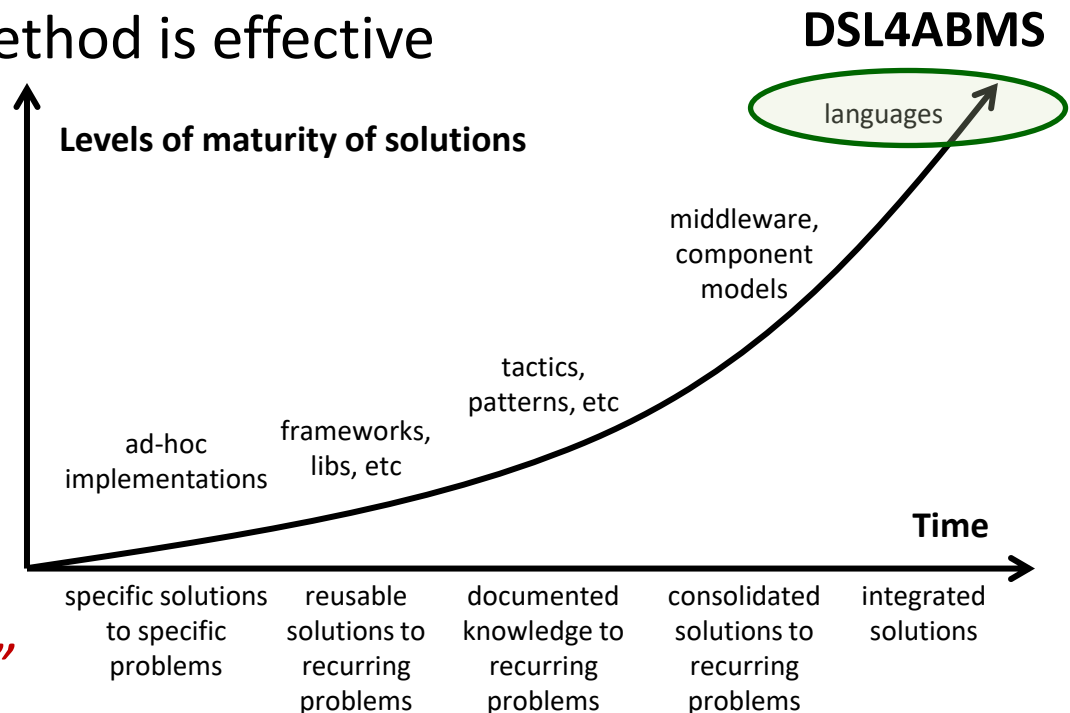
*Effort = AMEs + MLoCs

- Why does our MDD approach produce simulations with more LoCs than NetLogo/ITSUMO?
- It produces code for **reusable** domain-independent abstractions (e.g., state machines)
- But no **human effort** is required to produce it

Conclusion



- Our MDD approach reduces the effort to develop agent-based simulations in the ATSC domain
- The domain analysis method is effective



“most efforts are at lower levels of solution maturity”

Weyns, D., & Michel, F. (2015). Agent environments for multi-agent systems—a research roadmap. In *Agent Environments for Multi-Agent Systems IV* (pp. 3-21). LNCS/Springer.

Future Work



- Experiment with humans to evaluate subjective aspects of our MDD approach
 - E.g., usability, comprehensibility
- Incorporate additional simulation concepts
- Long-term goal: to use MDD to ease the development of agent-based simulations

Invitation: Demo Sessions



- Thursday, May 11
 - 10:20h - 11:20h
 - 16:10h - 17:10h



ABSTRACTME: Modularized Environment Modeling in Agent-based Simulations

Deividi Moreira¹, Fernando Santos^{1,2}, Matheus Barbieri¹, Ingrid Nunes^{1,3}, Ana L. C. Bazzan¹
¹UFRGS, Brazil; ²UDESC, Brazil; ³TU Dortmund, Germany
(dfsmoreira, fsantos, mbarbieri, ingridnunes, bazzan)@inf.ufrgs.br



1. Introduction

- Agent-based Simulations (ABS)
- Simulated environment, entities, and agents
- Recurrent elements
- Environment topology, model parameters, setup routines

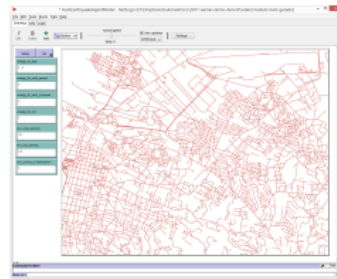


• DSL4ABMS: a Domain-specific Language (DSL)

- Expressively models the simulated environment in ABSs
- Towards a Model-driven Development approach for ABS
- Modularity: elements grouped by concern

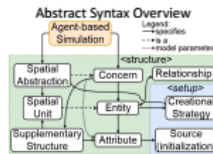
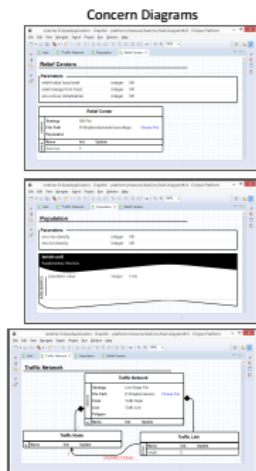
3. Code Generation

- Generates ready-to-use code to the NetLogo platform
- Spatial abstraction and creation and initialization of entities
- Visual components for manual inputs



2. The ABSTRACTME Tool

- Supports ABS development
- Provides an implementation to the DSL4ABMS language



Overview Diagram



- Grid, Cartesian Space, Graph
- File-based creation: GIS, SHP

4. Evaluation

- Modeling section
- Usefulness, Satisfaction, and Ease of Use questionnaire



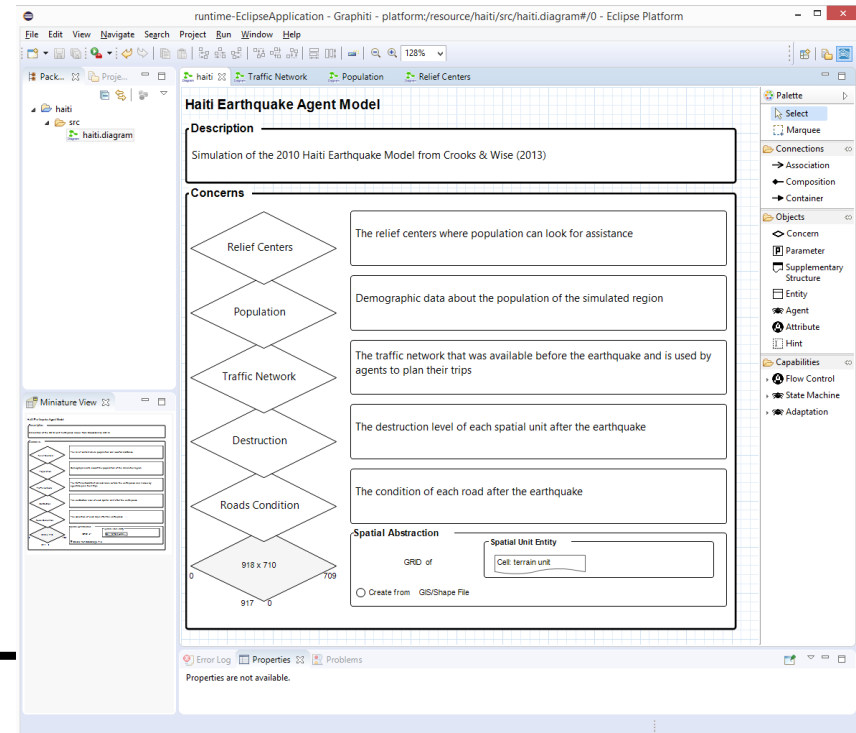
5. Demo Video

<https://youtu.be/Z4DeVDwdjVw>



References

DSL4ABMS.
<http://www.inf.ufrgs.br/prosew/resources/dsl4abms/>
Santos, F., Nunes, I., Bazzan, A. Supporting the Development of Agent-based Simulations: a DSL for Environment Modeling. In: COMPSAC 2017, Italy (to appear).
Weyns, D., & Michel, F. (2015). Agent environments for multi-agent systems: a research roadmap. In: Agent Environments for Multi-Agent Systems IV (pp. 3-21). Springer International Publishing.





A Case Study of the Development of an Agent-based Simulation in the Traffic Signal Control Domain using an MDD Approach

Fernando Santos^{1,2}, Ingrid Nunes^{1,3}, Ana L.C. Bazzan¹

¹ Instituto de Informática, UFRGS, Brazil

² Depto. de Engenharia de Software, UDESC, Brazil

³ TU Dortmund, Germany

{fsantos, ingridnunes, bazzan}@inf.ufrgs.br

May 8, 2017



Prosoft

