Towards Scalable Multidimensional Execution Traces for xDSMLs

MoDeVVa 2014 Valencia

Erwan Bousse[1] Benoit Combemale[2] Benoit Baudry[2]

[1]University of Rennes 1 (IRISA), France

[2]Inria, France

October 2, 2014

Outline

Plan

Context – xDSMLs and traces

- Recently, a lot of effort in the executable Domain Specific Languages (xDSMLs) field
- Executability of models opens possibilities in terms of early verification and validation (V&V) of systems, including the possiblity to rely on **dynamic V&V approaches**

Central concept in dynamic V&V approaches: **execution traces!**

Examples of trace usages in dynamic V&V:

- **Debugging:** a trace can be replayed
- **Model checking:** counter example in the form of a trace
- **Runtime monitoring:** checks if a trace satisfies a property

Context – xDSMLs and traces

- Recently, a lot of effort in the executable Domain Specific Languages (xDSMLs) field
- Executability of models opens possibilities in terms of early verification and validation (V&V) of systems, including the possiblity to rely on **dynamic V&V approaches**

Central concept in dynamic V&V approaches: **execution traces!**

Examples of trace usages in dynamic V&V:

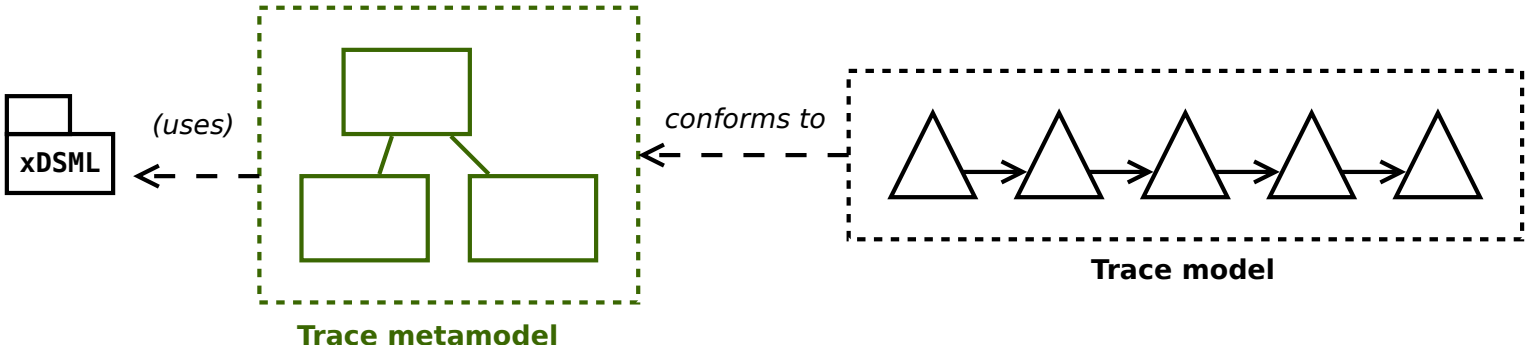
- **Debugging:** a trace can be replayed
- **Model checking:** counter example in the form of a trace
- **Runtime monitoring:** checks if a trace satisfies a property

Context – trace metamodel(s)

- We want to manipulate traces from various xDSMLs

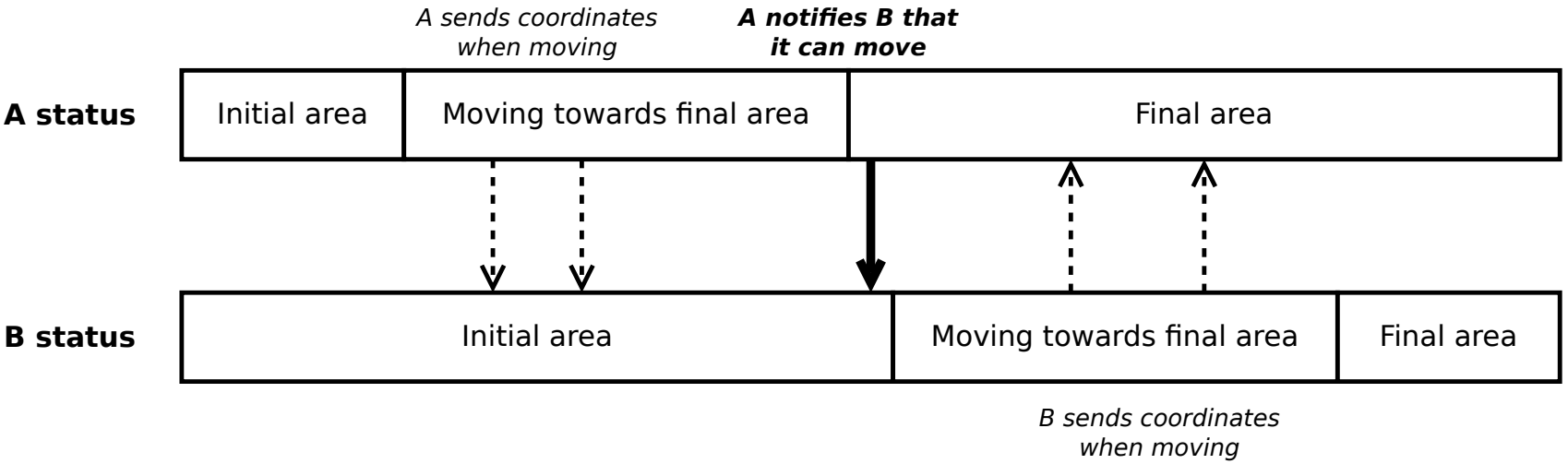
→ Need for **trace metamodel(s)**

Problem: what do we want in trace models, and thus in trace metamodels?



Motivating example: RoboML model and scenario

- RoboML: xDSML to model the behavior of robots
- Timed automata with domain concepts (e.g. GPS coordinates)



Two properties:

(a) B starts moving 5 seconds after A.

(b) Each time a robot covers 1 meter, it sends a message to the other one with its new coordinates.

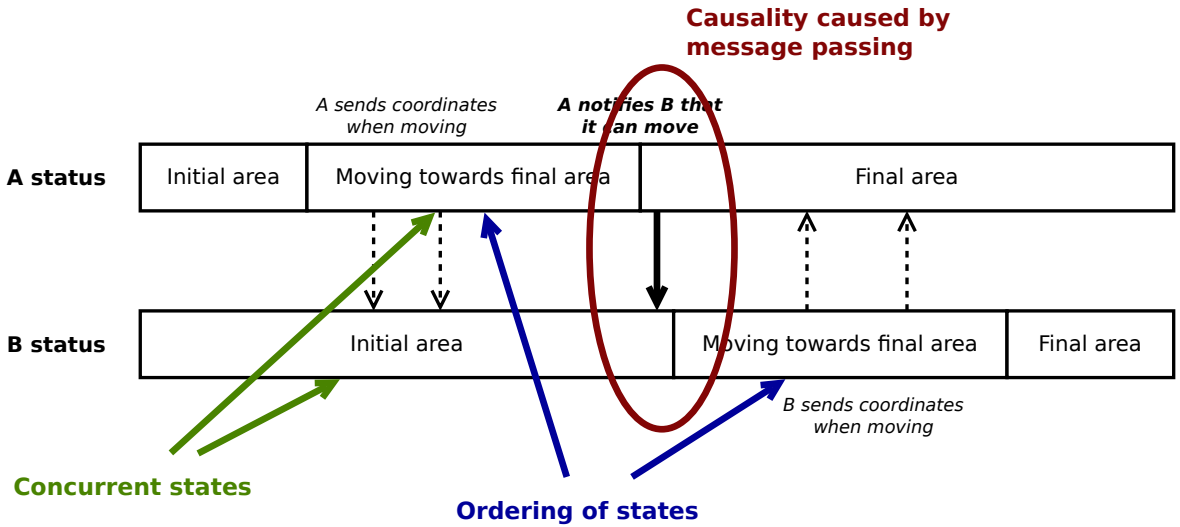
Plan

Pb. 1 – Concurrency modeling

Model execution can involve:

- One or multiple models, each with multiple variables
- Model states may be (partially) independent from one another
- More generally, **variables states may or may not be independent from other variables states**

RoboML scenario illustration



Pb. 2 – User-defined additional information

- During an execution, we have access to runtime data which depends on the xDSML semantics.
- Yet some properties may concern **derived or external data**, not available in the state of a given executed model

RoboML scenario illustration

(b) *Each time a robot* **covers 1 meter**, *it sends a message to the other one with its new coordinates.*

- "Covered meters" not part of the available runtime data
- Yet it can be derived from evolution of the coordinates of the robot and would belong in the trace to verify such properties.

Pb. 2 – User-defined additional information

- During an execution, we have access to runtime data which depends on the xDSML semantics.
- Yet some properties may concern **derived or external data**, not available in the state of a given executed model

RoboML scenario illustration

(b) *Each time a robot* **covers 1 meter**, *it sends a message to the other one with its new coordinates.*

- "Covered meters" not part of the available runtime data
- Yet it can be derived from evolution of the coordinates of the robot and would belong in the trace to verify such properties.

Pb. 3 – Scalability in space

- Traces can be **arbitrarily large**, as some complex systems are monitored continuously in case a failure occurs
- Scalability in space must be managed offline (file or database storage) or online (in memory)

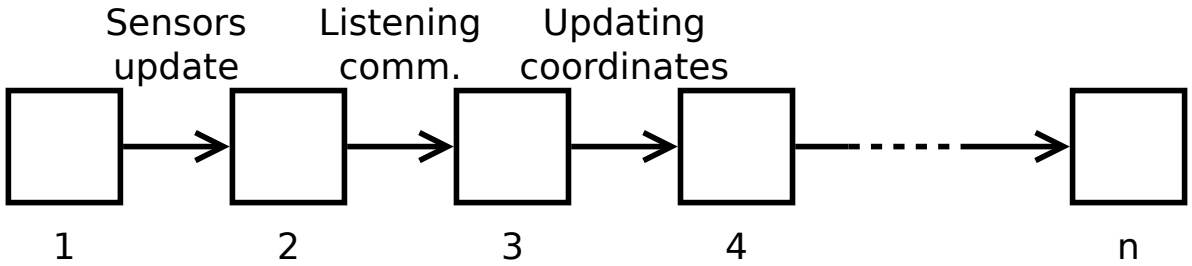
RoboML scenario illustration

A robot changes its internal state all the time to update its coordinates or listen to communications, leading quickly to a large amount of states.

Pb. 3 – Scalability in space

- Traces can be **arbitrarily large**, as some complex systems are monitored continuously in case a failure occurs
- Scalability in space must be managed offline (file or database storage) or online (in memory)

RoboML scenario illustration



A robot changes its internal state all the time to update its coordinates or listen to communications, leading quickly to a large amount of states.

Plan

Pb. 4 – Modularity

- To allow the removal/add of information within a trace and to allow traces with partial information
- Important in order to:
  - trace only a subset of the runtime data
  - extract a subset of the information of a trace
  - add information to the trace (e.g. derived variables)

RoboML scenario illustration

(a) *B starts moving 5 seconds after A.*

Only concerns the movement states of the robots and not their coordinates, thus extracting a trace with only the former information would be relevant to prove this property.

Pb. 4 – Modularity

- To allow the removal/add of information within a trace and to allow traces with partial information
- Important in order to:
  - trace only a subset of the runtime data
  - extract a subset of the information of a trace
  - add information to the trace (e.g. derived variables)

RoboML scenario illustration

(a) *B starts moving 5 seconds after A.*

Only concerns the movement states of the robots and not their coordinates, thus extracting a trace with only the former information would be relevant to prove this property.

Pb. 5 – Manipulation safety

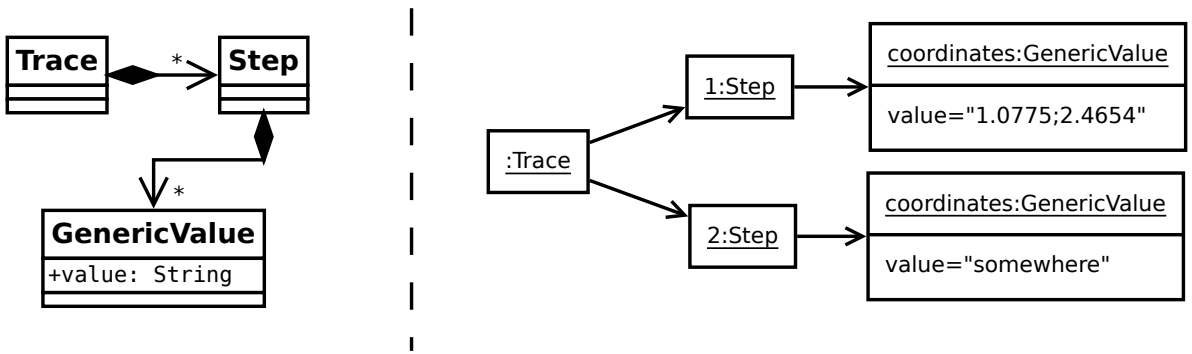
- **Generic trace metamodels** exist to model all kinds of traces for any executable language.
- However, constructing traces with such metamodels may lead to **inconsistent trace models**, since their genericity does not forbid one to create a trace whose states are not relevant to the concerned xDSML.

RoboML scenario illustration

Pb. 5 – Manipulation safety

- **Generic trace metamodels** exist to model all kinds of traces for any executable language.
- However, constructing traces with such metamodels may lead to **inconsistent trace models**, since their genericity does not forbid one to create a trace whose states are not relevant to the concerned xDSML.

RoboML scenario illustration



Pb. 6 – Reuse of trace manipulations

- Trace verification implies trace querying, thus **operations to manipulate traces**
- Important need: to be able to **reuse** such operations from a trace to another, from a system to another, or even from an xDSML to another

RoboML scenario illustration

Verifying property (a) requires an operator that checks all states that are found 5 seconds after a specific situation, which can be generalized in a *within* operator.

Pb. 6 – Reuse of trace manipulations

- Trace verification implies trace querying, thus **operations to manipulate traces**
- Important need: to be able to **reuse** such operations from a trace to another, from a system to another, or even from an xDSML to another

RoboML scenario illustration

Verifying property (a) requires an operator that checks all states that are found 5 seconds after a specific situation, which can be generalized in a *within* operator.

Pb. 7 – Scalability in time

- Analyzing traces may require to **iterate over all its steps**. The potentially large size of a trace compromises the capacity to make queries in a reasonable time.
- Moreover, if some variable referenced in a property only changes lately in a trace, we would still have to iterate through all steps before noticing that change.

RoboML scenario illustration

For property (a), would be interesting to avoid exploring the first half of the trace

Pb. 7 – Scalability in time

- Analyzing traces may require to **iterate over all its steps**. The potentially large size of a trace compromises the capacity to make queries in a reasonable time.
- Moreover, if some variable referenced in a property only changes lately in a trace, we would still have to iterate through all steps before noticing that change.

RoboML scenario illustration

For property (a), would be interesting to avoid exploring the first half of the trace

Plan

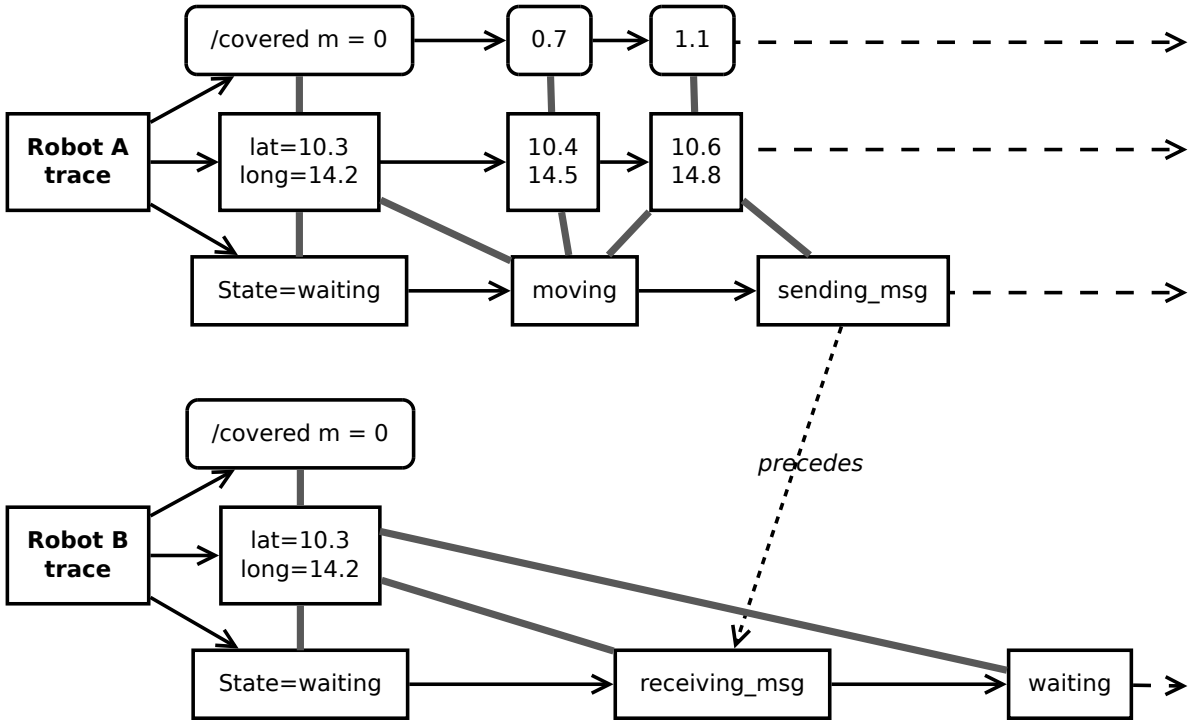
Idea 1 – Multiple dimensions

- Within a trace, very likely that only a subset of the variables really change from a state to another.
- Idea: consider **multiple dimensions in a trace**, each being a set of mutable elements of the executed model.
- A trace would then be a set of **subtraces**, each being the evolution of a specific dimension.

Problems it solves

- concurrency relationships can be defined (Pb. 1)
- adding or removing subtraces gives us modularity (Pb. 4)
- adding subtraces is a way to enrich the trace (Pb. 2)
- we can iterate separately on different dimensions, which could improve scalability in time (Pb. 7)

Example (sketch): a trace from the RoboML scenario



- 6 dimensions (3 per robot: covered meters, coord., state)
- precedence relationship with the message sending.
- Covered meters being not in the runtime data, it is added as a new dimension derived from coordinates

Idea 2 – Data sharing

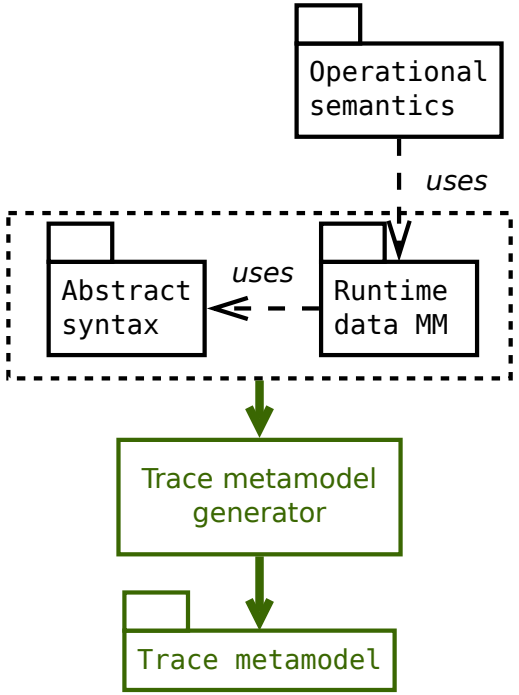
- From one step of a trace to another, there can be redundant data when dimensions values are used multiple times in the same run
- Idea: each dimension states are stored in a **dedicated storage structure** in order to be referenced by the steps of the same dimension.

Problems it solves

Helps with scalability in space (Pb. 3)

Idea 3 – Domain specific trace metamodels

- Given some xDSML semantics, we know what kind of runtime data is manipulated when executing a model
- Idea: a **domain-specific trace metamodel** that completely matches our xDSML, precisely defining the set of all possible traces of a given xDSML
- Drawback: requires a generative approach



Problems it solves

manipulation safety for trace models (Pb. 5),

Idea 4 – Trace API

- If we generate domain specific trace metamodels, then operations defined for one trace format are not compatible with another trace format
- Idea: a **common generic API** for all trace metamodels, whose operations can be automatically generated along with the metamodel
- Examples: *filter, merge, slice, during*, etc.

Problems it solves

operations reuse (Pb. 6), both those defined in the API and those using the API

Plan

Conclusion

- Verification and validation of executable models is a challenge that requires the modeling of execution traces.
- We identified a selection of problems that must be considered when modeling traces
- Our idea: **multidimensional traces** coupled with a trace manipulation API.
- Takes into account concurrency, scalability, modularity among other aspects.

Further work

A first implementation of the approach and its application to RoboML, along with memory/time measures to compare with "naïve" traces

Done!

Thank you for your attention ☺