A Generative Approach to Define Rich Domain-Specific Trace Metamodels

ECMFA'15

Erwan Bousse [1] Tanja Mayerhofer [2] Benoit Combemale [3] Benoit Baudry [4]

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

[2]Vienna University of Technology, Austria

[3]Inria, France

July 22, 2015

Outline

Outline

Context: xDSMLs and traces

- Recently, a lot of effort in the field of executable Domain Specific (Modeling) Languages (xDSMLs)
- From a Verification and Validation (V&V) point of view, need for **dynamic V&V approaches** to analyse the behaviors of executable models, ie. temporal properties

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

Examples of trace usages in dynamic V&V:

- **Omniscient Debugging:** a trace is used to step backward
- **Model checking:** counter example in the form of a trace
- **Runtime monitoring:** to check if a trace satisfies a property
- **Semantic differencing:** trace comparison of different models

Context: xDSMLs and traces

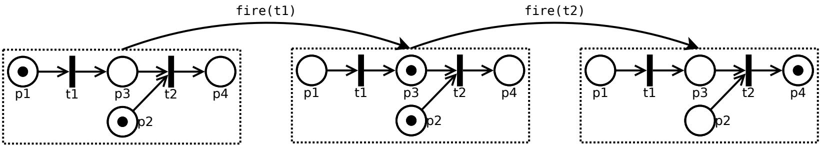
- Recently, a lot of effort in the field of executable Domain Specific (Modeling) Languages (xDSMLs)
- From a Verification and Validation (V&V) point of view, need for **dynamic V&V approaches** to analyse the behaviors of executable models, ie. temporal properties

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

Examples of trace usages in dynamic V&V:

- **Omniscient Debugging:** a trace is used to step backward
- **Model checking:** counter example in the form of a trace
- **Runtime monitoring:** to check if a trace satisfies a property
- **Semantic differencing:** trace comparison of different models

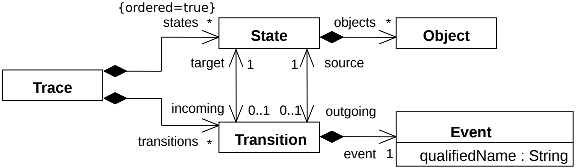
Execution Trace



- An alternate sequence of *states* and *events*
- A *state* contains the values of all the mutable parts of a model
- An *event* is the application of a transformation rule (focus on *operational semantics*)

Problem: generic trace metamodels are not good enough

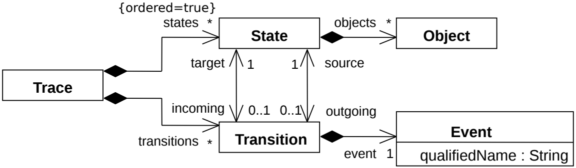
Example of generic clone-based trace metamodel:



- **Scalability in time issue**: sequential structure ⇒ to navigate in a trace, each execution state has to be visited
- **Usability issue**: domain-specific trace analyses have to handle domain-specific data that may be arbitrarily complex, and a generic set of objects is not convenient

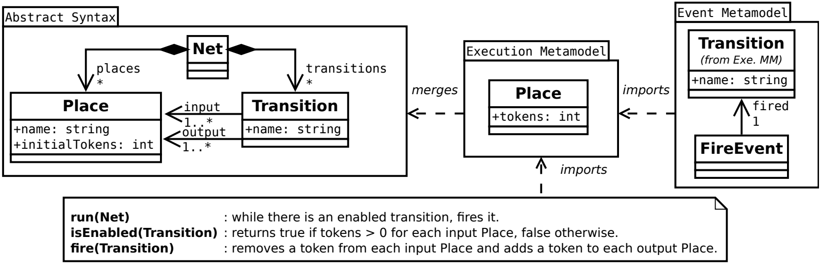
Problem: generic trace metamodels are not good enough

Example of generic clone-based trace metamodel:



- **Scalability in time issue**: sequential structure ⇒ to navigate in a trace, each execution state has to be visited
- **Usability issue**: domain-specific trace analyses have to handle domain-specific data that may be arbitrarily complex, and a generic set of objects is not convenient

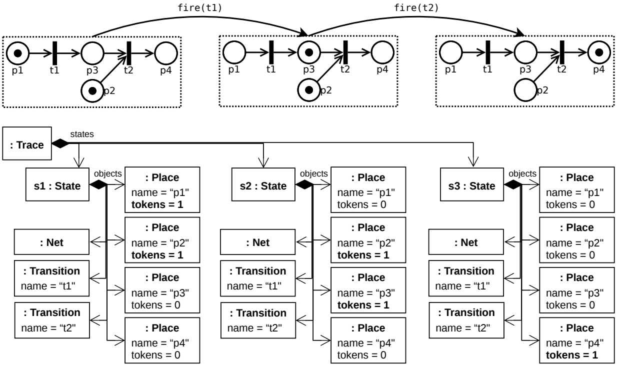
Example: Petri net xDSML



Operational Semantics (set of transformation rules)

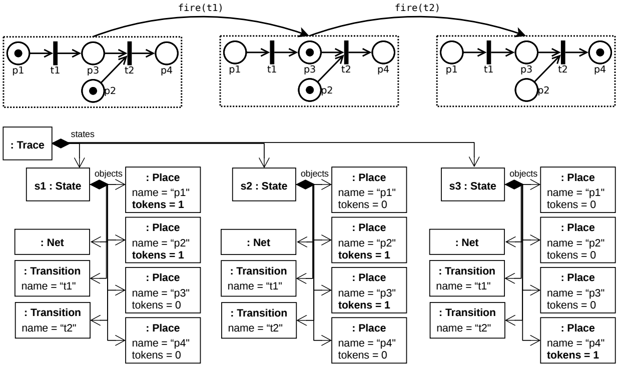
- Properties of the *abstract syntax* (e.g. `initialTokens`) are said **immutable**
- Properties of the *execution metamodel* (`tokens`) are said **mutable**

Example: generic Petri net trace



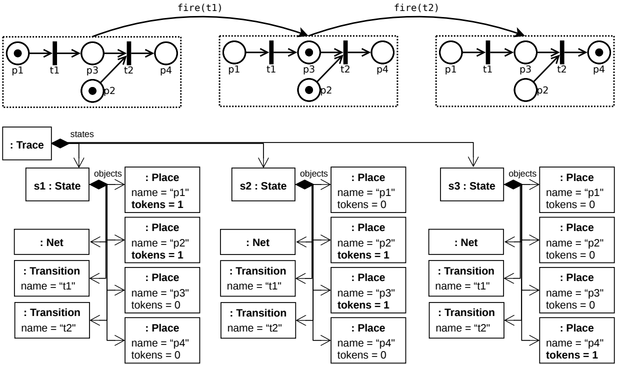
1. Generic types require **type checks and casting**
2. **Redundancy**, both with immutable and mutable data
3. Trace can be only queried by **visiting all states**

Example: generic Petri net trace



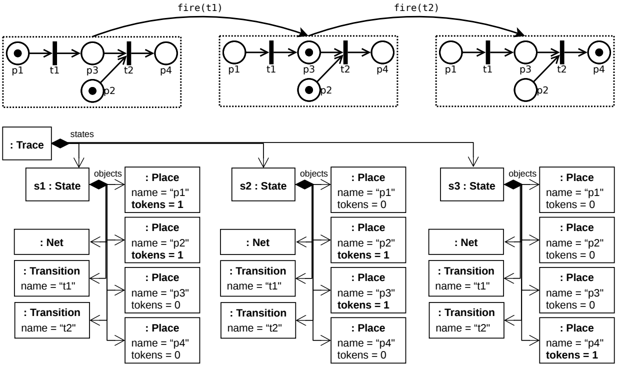
1. Generic types require **type checks and casting**
2. **Redundancy**, both with immutable and mutable data
3. Trace can be only queried by **visiting all states**

Example: generic Petri net trace



1. Generic types require **type checks and casting**
2. **Redundancy**, both with immutable and mutable data
3. Trace can be only queried by **visiting all states**

Example: generic Petri net trace



1. Generic types require **type checks and casting**
2. **Redundancy**, both with immutable and mutable data
3. Trace can be only queried by **visiting all states**

Outline

Approach: generating a domain-specific trace metamodel

To provide usability

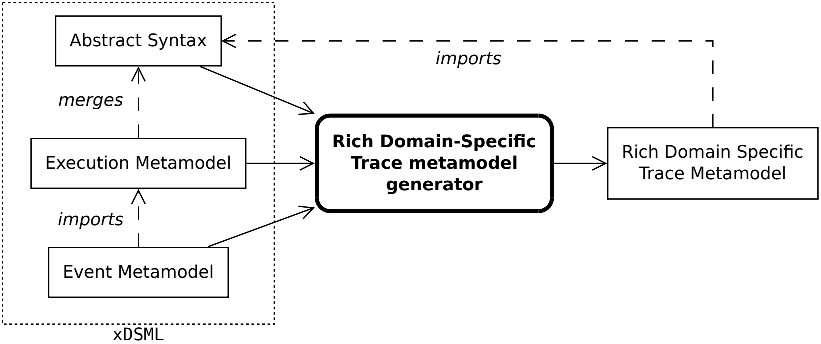
Generative approach to automatically derive a **domain-specific trace metamodel** for a given xDSML

- *Domain-specific*: domain concepts are directly accessible
- *Automation*: save language engineers the design of a complex metamodel, which is time-consuming and error-prone

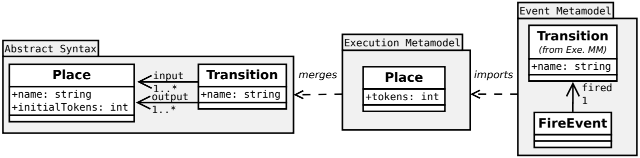
To provide scalability in time

Rich navigation facilities, e.g. browsing a trace according to the values reached by a specific mutable element

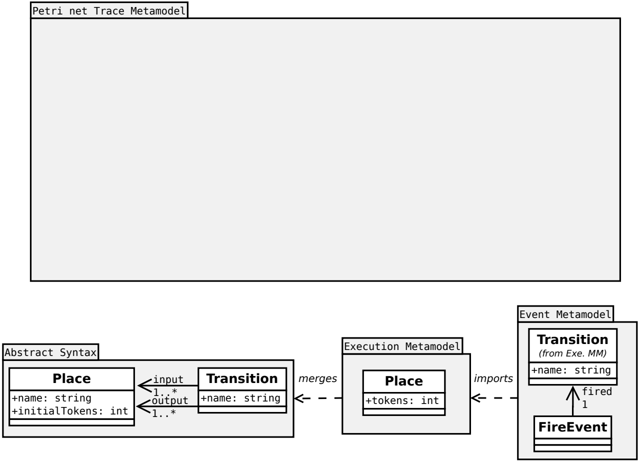
Overview


Abstract Syntax
merges
Execution Metamodel
imports
Event Metamodel
xDSML
imports
Rich Domain-Specific Trace metamodel generator
Rich Domain Specific Trace Metamodel

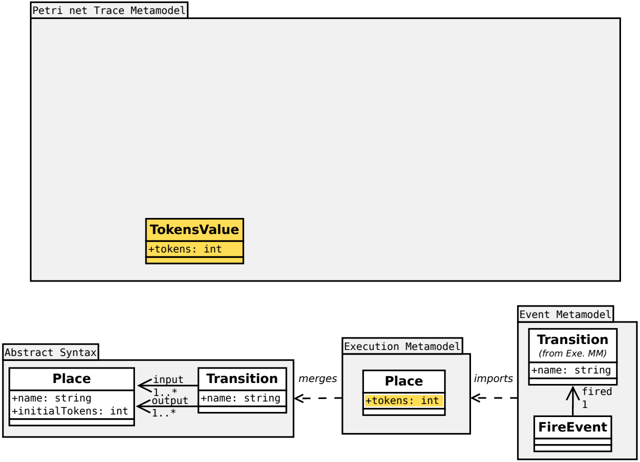
Trace metamodel generation (1)



Trace metamodel generation (1)

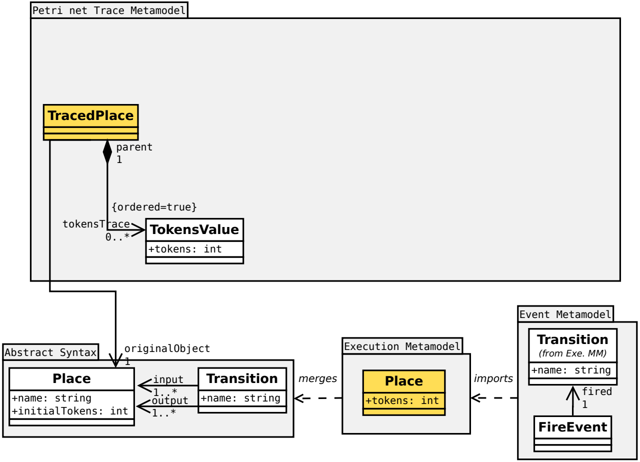


Trace metamodel generation (2)



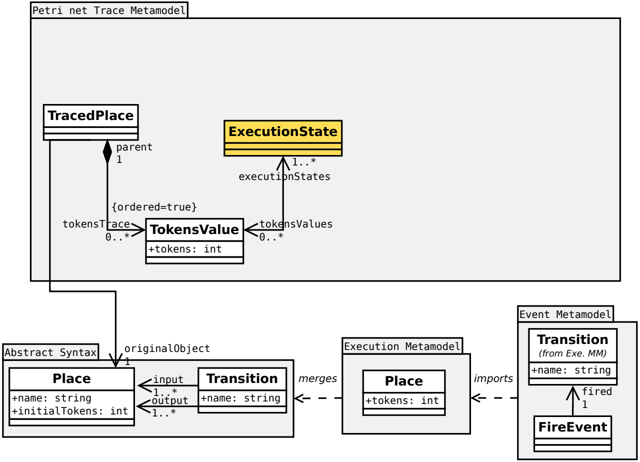
- **Reification** of mutable properties into classes

Trace metamodel generation (3)



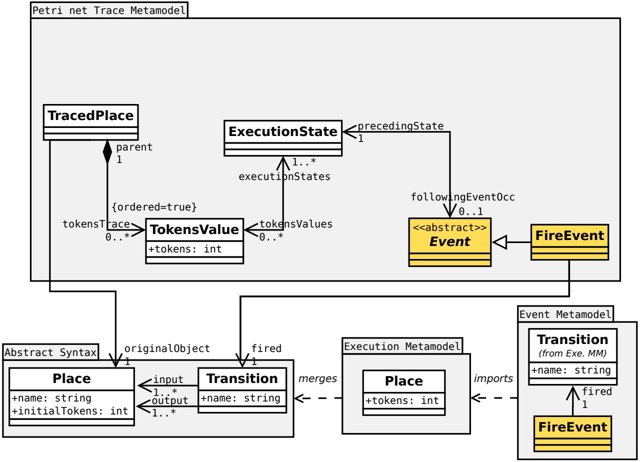
- Class for **value sequences** of mutable data of objects

Trace metamodel generation (4)



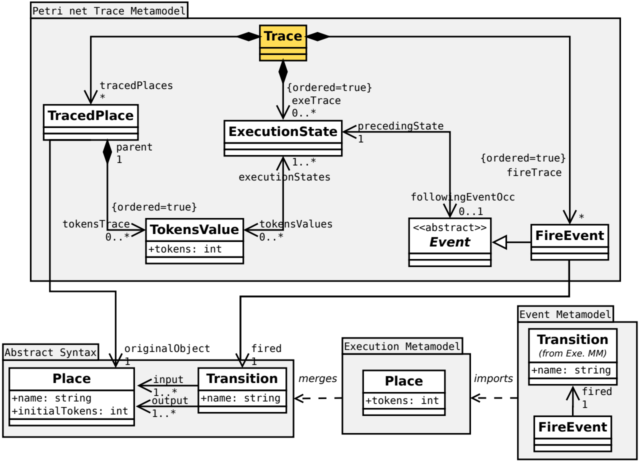
- Class for the **model state** $\equiv$ **tuple of all mutable values**

Trace metamodel generation (5)



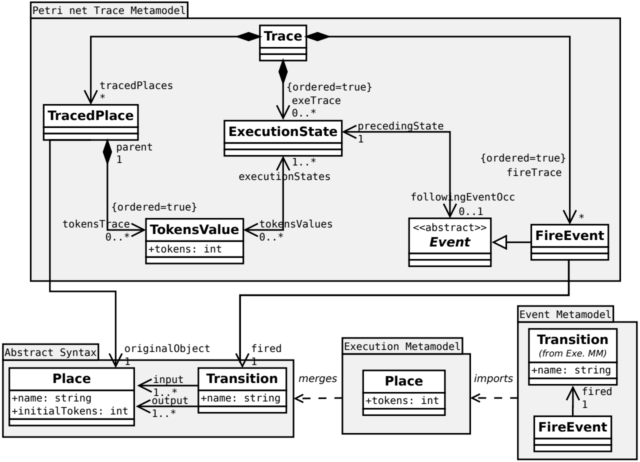
- Class for each **event**, with one event following each state

Trace metamodel generation (6)

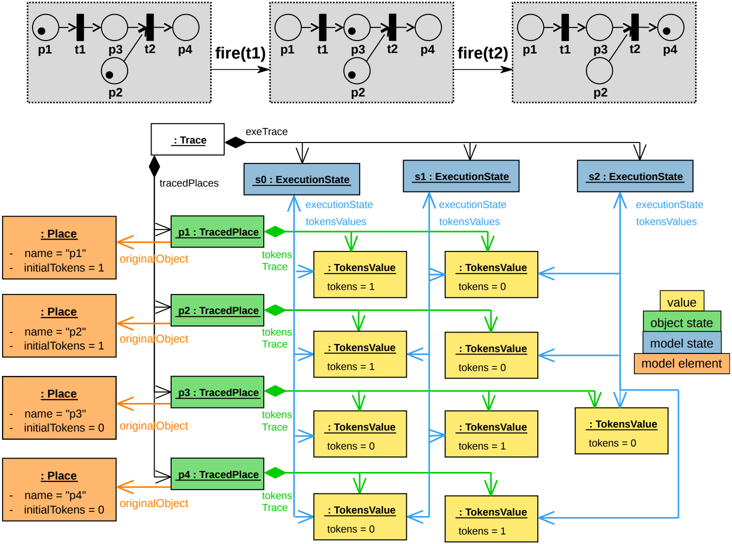


- Class for the **trace**, which includes **event sequences**

Trace metamodel generation (6)



Example of Domain Specific Trace



Implementation

Implementation available for EMF

- **Trace metamodel generator** (generic)
  - Input abstract syntax: Ecore
  - Input execution semantics: xMOF or Kermeta
  - Output trace metamodel: Ecore
- **Trace builder** (generic)
  - Instruments xMOF/fUML virtual machine
  - Produces traces conforming to generated rich domain-specific trace metamodel
- **Git repository**:
  `https://gforge.inria.fr/projects/lastragen/`

Also part of the **GEMOC Studio**: `http://gemoc.org/studio`

Outline

Case study

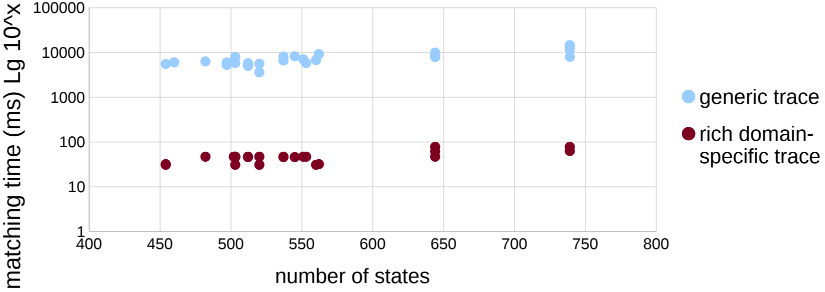
Trace manipulations: Semantic Differencing

- Consists in comparing traces of different versions of a model according to a set of **semantic differencing rules**
- A set of semantic differencing rules is:
  - specific to an xDSML, hence **domain-specific**
  - specific to a given trace metamodel
  - written using the *Epsilon Comparison Language (ECL)*

xDSML: fUML

- Real world xDSML, subset of UML
- Used models: activity diagrams taken from [Maoz etal. 2011]

Evaluation: scalability in time



- **Result:** Match rules on rich domain-specific traces are between 170 and 400 times faster (on average 250 times)
- **Reason:** Rich structure of traces allow efficient querying of state changes of particular model elements

Evaluation: usability

Elements	Generic	Rich Domain-Specific	Gain
Lines of code	136	55	60%
Statements	58	21	64%
Operation calls	32	13	60%
Loops	5	4	40%
Type checks	4	0	100%

- **Result:** Complexity of rules reduced between 40% and 100%
- **Reasons:**
  - Rich structure of traces allow efficient querying of state changes of particular model elements
  - Domain-specific structure makes type checks and casting obsolete

Outline

Conclusion

- xDSMLs bring a lot of possibilities: simulation, dynamic V&V
- Dynamic V&V uses traces, and hence a trace data structure
- Generic trace metamodels have two weaknesses: usability and scalability in time
- To cope with these issues, generation of rich domain-specific trace metamodels

Perspectives

- Enhancing customisation of trace metamodels
- Experiment other V&V activities on top of generated trace metamodels

Done!

Thank you for your attention! ☺