Trace Management Facilities to Support V&V in Executable DSLs

DiverSE SLE Seminar

Erwan Bousse\textsuperscript{1}  Benoit Combemale\textsuperscript{2}  Benoit Baudry\textsuperscript{2}

\textsuperscript{1}University of Rennes 1 (IRISA), France

\textsuperscript{2}Inria, France

May 28, 2015
Outline

1. Problem Statement
2. Scalable Model Cloning
3. Rich Domain-Specific Trace Metamodels
4. Rich Omniscient Debugging
5. Conclusion
Plan

1. Problem Statement
2. Scalable Model Cloning
3. Rich Domain-Specific Trace Metamodels
4. Rich Omniscient Debugging
5. Conclusion
Recently, a lot of effort in the field of executable Domain Specific Languages (xDLSs)

From a Verification and Validation (V&V) point of view, need for **dynamic V&V approaches** in order to analyse the behaviors of executable models

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**: checks if a trace satisfies a property
Recently, a lot of effort in the field of executable Domain Specific Languages (xDSLs)
From a Verification and Validation (V&V) point of view, need for **dynamic V&V approaches** in order to analyse the behaviors of executable models

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**: checks if a trace satisfies a property
Definition: Execution Trace

- An alternate sequence of states and small steps
- A state contains the values of all the mutable parts of a model
- A small step is the application of a transformation rule from the operational semantics
- A big step is a sequence of steps
Concerns when Managing Traces

What kind of data structure for a trace?
How to handle a (potentially large) trace?

Concerns we focused on

- **Usability**, e.g. making information accessible without searching and without using type checks / casting
- **Scalability in space**, both offline (file or database storage) or online (in memory)
- **Scalability in time**, e.g. browsing a large trace
Observation: mutable subset of a metamodel

- Operational semantics only access (read) and change (write) model elements corresponding to a subset of the considered metamodel.
- We call the subset concerned by write operations the **mutable subset of a metamodel**.

```
fire(Transition t) {
   ....
}
```
Intuition: focus on mutable subset

- When constructing the trace of an execution, we can focus on these mutable parts, and store only once the immutable parts.
- In other words, we can reduce redundancies within execution traces.
- This can be done both:
  - by working on the runtime representation of traces, using a memory data structure that avoids redundancy.
  - by working on traces metamodels, in order to provide efficient representations of traces.
Example: a metamodel, a model, and runtime counterparts
Overview of the contributions

- Scalable model cloning (MODELS 2014)
- Rich domain-specific trace metamodel generation (ECMFA 2015)
- Rich Omniscient debugging (work in progress)

Application to V&V

- Runtime representation of traces
  - Scalable model cloning (MODELS 2014)
  - Rich domain-specific trace metamodel generation (ECMFA 2015)
  - Rich Omniscient debugging (work in progress)

- Trace metamodels
  - Rich domain-specific trace metamodel generation (ECMFA 2015)
  - Reification of some ideas into generative approach
  - Application to V&V

Direct application
Plan

1. Problem Statement
2. Scalable Model Cloning
3. Rich Domain-Specific Trace Metamodels
4. Rich Omniscient Debugging
5. Conclusion
Context and motivation

Definition: clone

A clone is a model that is, when created, identical to an existing model. Both models conform to the same metamodel and are independent from one to another.

- New MDE activities which rely on the production of large quantities of models and variations of a set of models, that can be obtained through model cloning.
- In particular, cloning can be used to construct execution traces very conveniently.
Problem: efficient model cloning

- Need for the ability to clone a model
- Already possible using the most convenient cloning implementation: deep cloning (see EcoreUtil.Copier class)
- *deep cloning* ≡ duplicating the model in memory
Problem: efficient model cloning

- Need for the ability to **clone a model**
- Already possible using the most convenient cloning implementation: **deep cloning** (see `EcoreUtil.Copier` class)
- **deep cloning** ≡ duplicating the model in memory

**Problem:** **deep cloning** has very poor memory performances
Approach: *static* identification of safely shareable parts
Approach: *static* identification of safely shareable parts
Approach: *static* identification of safely shareable parts

![Diagram](image-url)

- **Operations**
- **Metamodel**
  - **Mutable elements** (runtime data, parts to optimize, etc.)
  - **Mutable elements** with shareable elements tagged
  - **Metamodel with shareable elements** tagged
  - **Shared data** (based on shareable elements of the MM)

**In memory**

- **Runtime repr. of a model**
- **Runtime repr. of a clone**
- **Cloning while safely sharing shareable data**
Evaluation

Experiment

- **data set**: 100 randomly generated metamodels
- **memory measures**: gain as compared to *deep cloning*, after cloning the model 1000 times
- **performance measures**: loss of time as compared to the original model, when navigating 10 000 times through each object of the model while accessing all properties

Results

- **memory**: the more shareable parts, the more memory gain
- **performance**: worst median overhead is 9.5% when manipulating clones with fields sharing
Evaluation

Experiment

- **data set**: 100 randomly generated metamodels
- **memory measures**: gain as compared to *deep cloning*, after cloning the model 1000 times
- **performance measures**: loss of time as compared to the original model, when navigating 10 000 times through each object of the model while accessing all properties

Results

- **memory**: the more shareable parts, the more memory gain
- **performance**: worst median overhead is 9.5% when manipulating clones with fields sharing
Plan

1. Problem Statement
2. Scalable Model Cloning
3. Rich Domain-Specific Trace Metamodels
4. Rich Omniscient Debugging
5. Conclusion
Problem: generic trace metamodels are not good enough

Example of Generic trace metamodel (≡ sequence of clones)

- **Efficiency issues**: sequential structure ⇒ to navigate in a trace, each execution state has to be visited
- **Usability issues**: 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 trace metamodel (≡ sequence of clones)

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

1. Generative approach to automatically derive a domain-specific trace metamodel for a given xDSL
   - *Domain-specific*: domain concepts are directly accessible
   - *Automation*: Save language engineers the design of a complex metamodel, which is time-consuming and error-prone

2. **Enrichment** with additional navigation facilities

Expectations:

- Usability of traces is improved
- Efficiency of trace manipulations is improved
Example of trace metamodel generation

Abstract Syntax

Net
- places
  - input
    - name: string
    - initialTokens: int
  - output
    - 1..*

Transition
- name: string
- transitions
- 1..*

Execution Metamodel
- Place
  - tokens: int
- TracedPlace
  - tokensTrace
- TokensValue
  - tokens
- Trace
  - tokensValues
- precedingState
- executionStates
- TracedPlace
- tracePlaces

Step Metamodel
- Transition
  - name: string
- ExecutionState
  - executionStates
- Events
  - followingEventOcc
- SmallStep
  - Fire

Trace Management Facilities to support V&V
Example of trace metamodel generation
Example of Domain Specific Trace

```
: Trace
s1 : ExecutionState
    p1 : TracedPlace
    p2 : TracedPlace
    p3 : TracedPlace
    p4 : TracedPlace
    executionState
    tokensValues
    tokens = 1
    tokens = 0
    tokens = 0
    tokens = 1
    tokens = 0
    fire(t1) fire(t2)

s2 : ExecutionState
    p1 : TracedPlace
    p2 : TracedPlace
    p3 : TracedPlace
    p4 : TracedPlace
    executionState
    tokensValues
    tokens = 0
    tokens = 0
    tokens = 1
    tokens = 0
```

Erwan Bousse, Benoit Combemale, Benoit Baudry
Evaluation: performance and usability

**Case study:** semantic differencing (ie. trace comparison)

Efficiency in time improvement

Usability improvement

<table>
<thead>
<tr>
<th>Elements</th>
<th>Generic</th>
<th>Rich Domain-Specific</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of code</td>
<td>136</td>
<td>55</td>
<td>60%</td>
</tr>
<tr>
<td>Statements</td>
<td>58</td>
<td>21</td>
<td>64%</td>
</tr>
<tr>
<td>Operation calls</td>
<td>32</td>
<td>13</td>
<td>60%</td>
</tr>
<tr>
<td>Loops</td>
<td>5</td>
<td>4</td>
<td>40%</td>
</tr>
<tr>
<td>Type checks</td>
<td>4</td>
<td>0</td>
<td>100%</td>
</tr>
</tbody>
</table>
Plan

1. Problem Statement
2. Scalable Model Cloning
3. Rich Domain-Specific Trace Metamodels
4. Rich Omniscient Debugging
5. Conclusion

Erwan Bousse, Benoît Combemale, Benoît Baudry

Trace Management Facilities to support V&V
Traditionnal (forward-only) debugging

Also notion of frame (e.g. a method in Java) into which we can step or from which we can step out.
Omniscient debugging

- A debugger only gives services to go **forward in time**
- Omniscient debugging aims at providing operators that can go **backwards in time** as well
  - **jump**: to go back to a chosen previous state
  - **step back into**: go back one step
  - **step back over**: go back one step over a frame
  - **step back out**: go back out of a frame
  - **play backwards**: goes back though all previous states
  - **visualization of history**: a representation of the previous states (counter, trace, etc.)

Omniscient debuggers usually rely on an **execution trace** to store previous states and implement these services.
Omniscient debugging

- A debugger only gives services to go **forward in time**
- Omniscient debugging aims at providing operators that can go **backwards in time** as well
  - **jump**: to go back to a chosen previous state
  - **step back into**: go back one step
  - **step back over**: go back one step over a frame
  - **step back out**: go back out of a frame
  - **play backwards**: goes back though all previous states
  - **visualization of history**: a representation of the previous states (counter, trace, etc.)

Omniscient debuggers usually rely on **an execution trace** to store previous states and implement these services.
Overview of the contribution (work in progress)

A generative approach for **rich omniscient debugging** for xDSLs

- rich because of **new jump services**, e.g. jump to a specific field value
- allows the reuse of domain-specific traces in other domain-specific trace activities
Implementation prototype in the GEMOC Studio
## Plan

1. Problem Statement
2. Scalable Model Cloning
3. Rich Domain-Specific Trace Metamodels
4. Rich Omniscient Debugging
5. Conclusion
Conclusion

■ xDSLs brings a lot of possibilities: simulation, dynamic V&V
■ Dynamic V&V requires execution traces, which can be large and difficult to store/use
■ Two studied approaches:
  - at the runtime representation level: Scalable Model Cloning
  - at the trace metamodel level: Rich domain-specific Trace metamodel generation
■ Application to rich omniscient debugging for executable DSLs

 Perspectives

■ Enhancing customisation of trace metamodels
■ Experiment other V&V activities on top of generated trace metamodels
Done!

Thank you for your attention! 😊