Centrum Wiskunde & Informatica

# Cascade: A Meta Language for Change, Cause and Effect

International Workshop on Live Programming (LIVE2022)

Dec. 6th 2022, Auckland New Zealand

**Riemer van Rozen**

NWO-I Centrum Wiskunde & Informatica (CWI)

Software Analysis & Transformation group (SWAT)

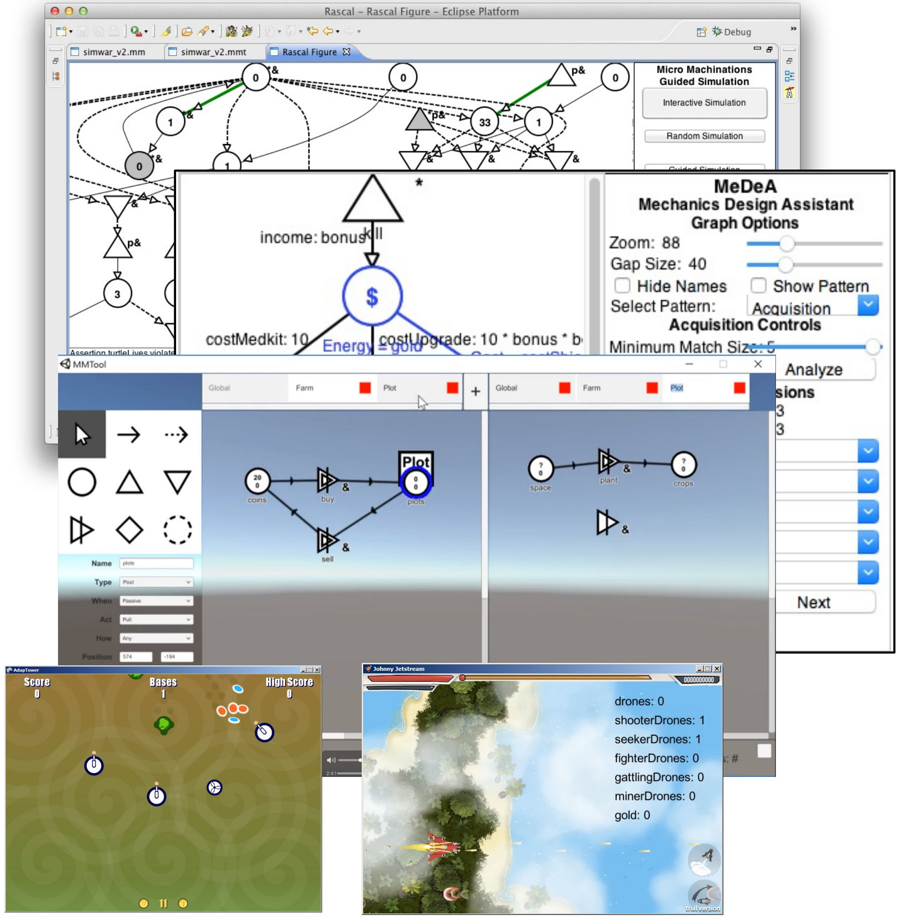
# Problem Statement

**Machinations**. Domain-Specific Language for game economies.

**Live Game Design.** Modifying a game's mechanics for prototyping, playtesting & fine-tuning gameplay.

**Problem.** There is a general lack of enabling technology for creating live programming languages and live programming environments.

**Objective.** Develop language-parametric enabling technology that powers live programming.

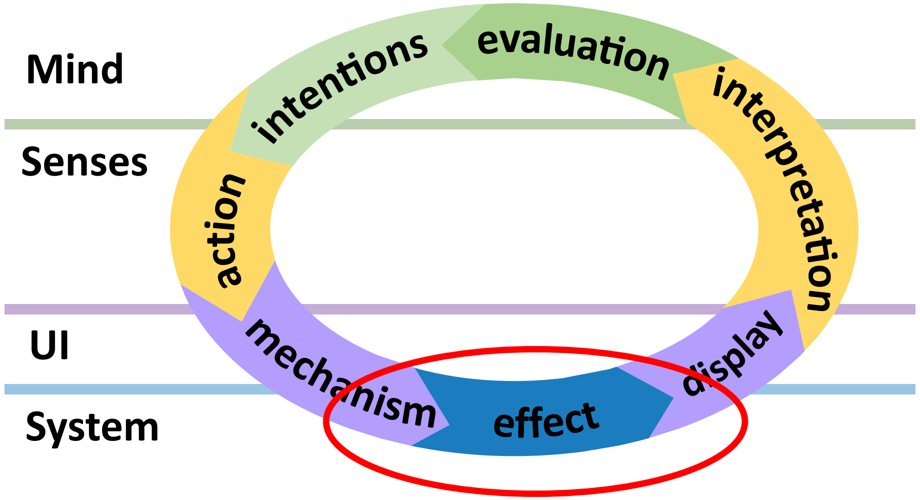


Zoom: 88
Gap Size: 40
□ Hide Names
□ Show Pattern
Select Pattern: Acquisition
Minimum Match Size: 5

drones: 0
shooterDrones: 1
seekerDrones: 1
fighterDrones: 0
gattlingDrones: 0
minerDrones: 0
gold: 0

# Enabling Live Language Technology

**Approach.** Develop Cascade, a meta-language for creating languages with interface- and feedback-mechanisms that drive live programming.



## Challenges addressed

- **Prototype live languages.** Obtain a REPL with mechanisms for coding, user interaction & live feedback.
- **Model languages.** Express abstract syntax and run-time state with meta-models.
- **Design side-effects.** Express a) *coding actions* that work on abstract syntax; and b) *user actions* that affect run-time state.
- **Account for run-time eventualities.** Express *cascading changes* in run-time scenarios (run-time state migration)

## References

- Donald A. Norman. "Cognitive engineering." *User centered system design* 31 (1986): 61.

# Live Programming Scenario

**TinyLiveSML.** Domain-Specific Language (DSL) for live state machines.

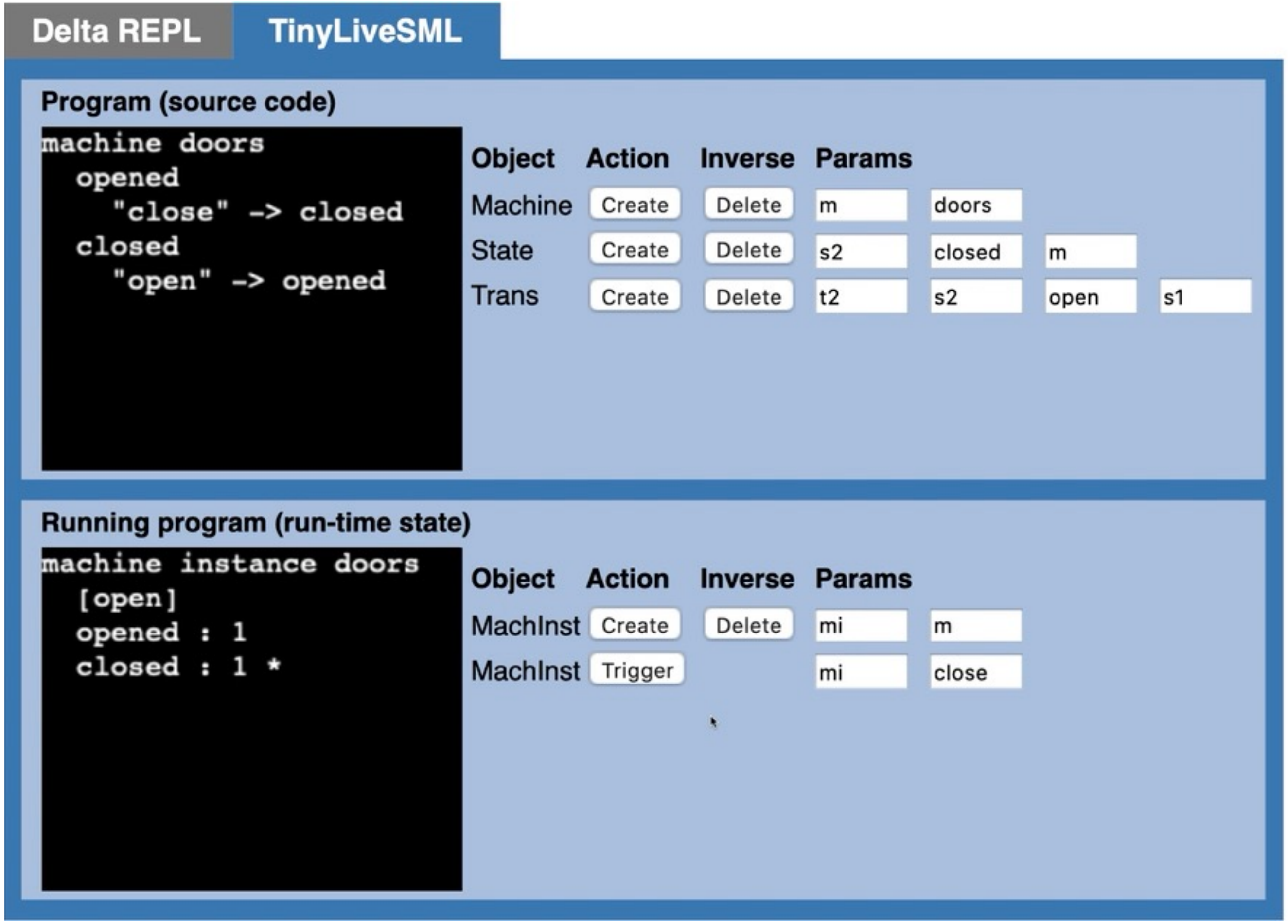
- Simultaneously program (top) and run programs (bottom)

**Programmer.** Creates

- machine doors
- state open, closed
- transition open, close

**User**

- Runs a machine doors. Initial state opened: 1 *
- Closes the door. Current state becomes closed: 1 *



Program (source code)

```
machine doors
  opened
    "close" -> closed
  closed
    "open" -> opened
```

| Object | Action | Inverse | Params | | | |
|---|---|---|---|---|---|---|
| Machine | Create | Delete | m | doors | | |
| State | Create | Delete | s2 | closed | m | |
| Trans | Create | Delete | t2 | s2 | open | s1 |

Running program (run-time state)

```
machine instance doors
  [open]
  opened : 1
  closed : 1 *
```

| Object | Action | Inverse | Params | |
|---|---|---|---|---|
| MachInst | Create | Delete | mi | m |
| MachInst | Trigger | | mi | close |

# Live Programming Scenario

**Programmer**

- Creates state locked

**TinyLiveSML**

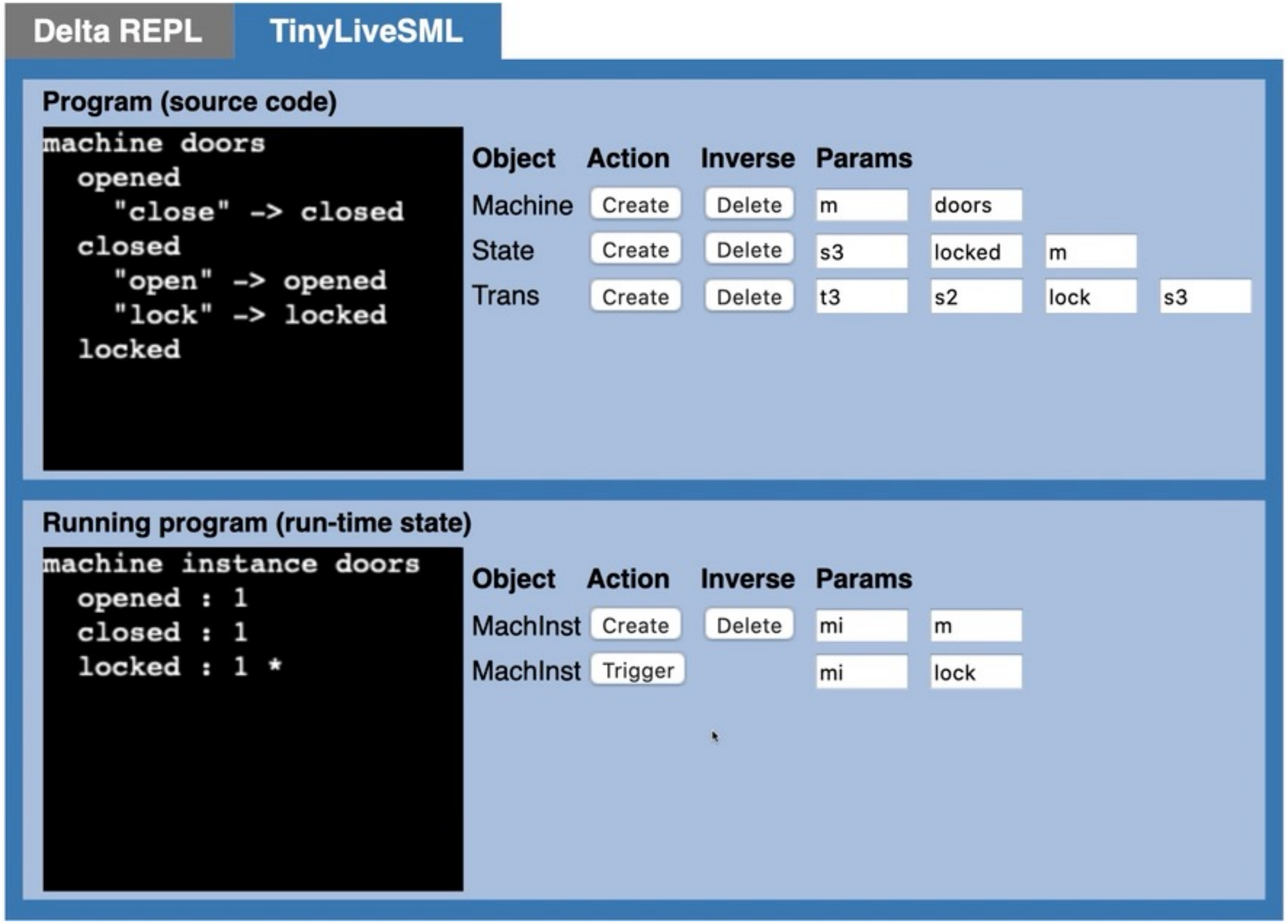
- Adds a state locked: 0 to running programs

**Programmer**

- Creates transition lock

**User**

- Locks the door. The current state becomes locked: 1 *



Program (source code)

```
machine doors
  opened
    "close" -> closed
  closed
    "open" -> opened
    "lock" -> locked
  locked
```

| Object | Action | Inverse | Params | | | |
|---|---|---|---|---|---|---|
| Machine | Create | Delete | m | doors | | |
| State | Create | Delete | s3 | locked | m | |
| Trans | Create | Delete | t3 | s2 | lock | s3 |

Running program (run-time state)

```
machine instance doors
  opened : 1
  closed : 1
  locked : 1 *
```

| Object | Action | Inverse | Params | |
|---|---|---|---|---|
| MachInst | Create | Delete | mi | m |
| MachInst | Trigger | | mi | lock |

# Live Programming Scenario

**Programmer**

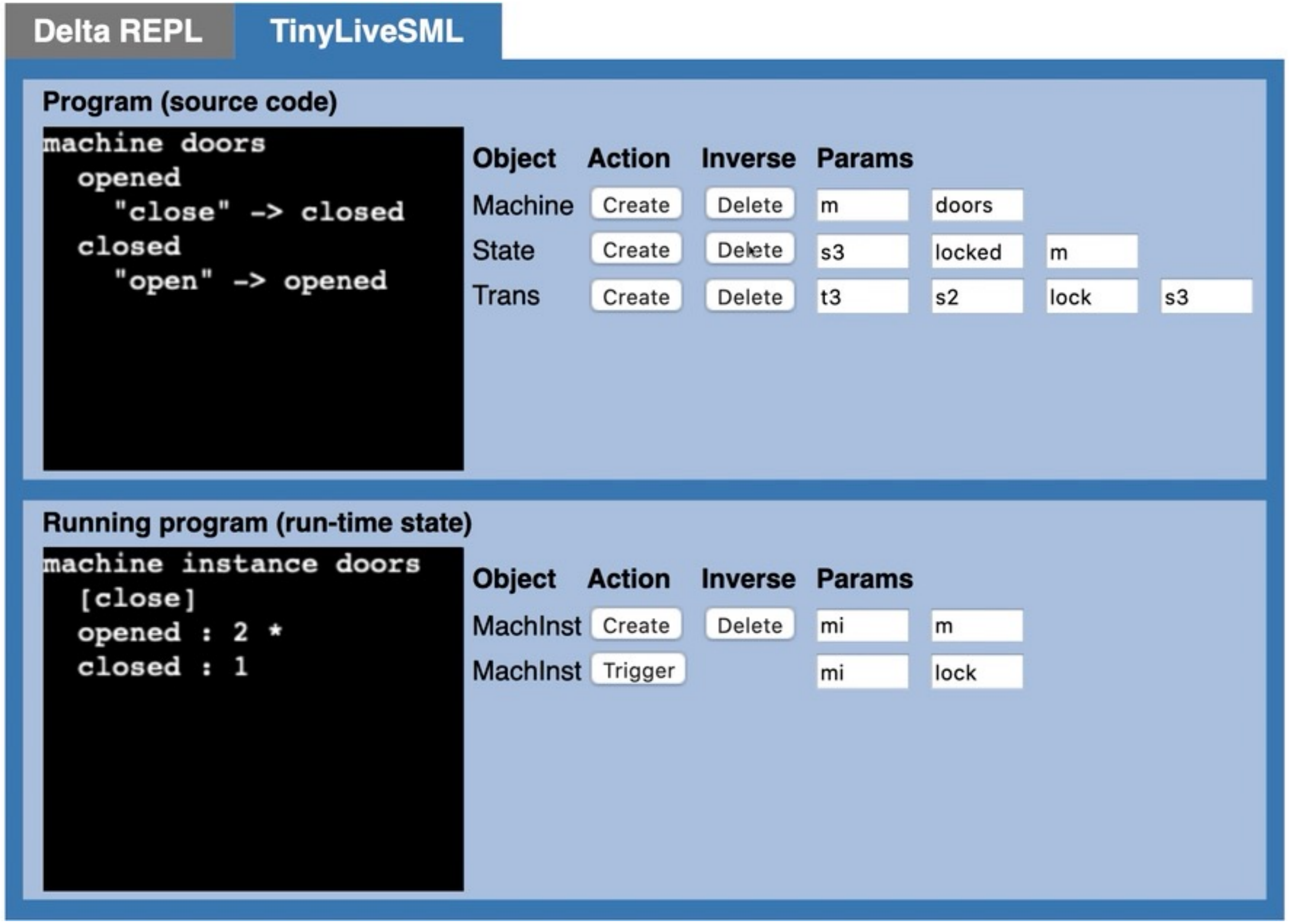
- Deletes state locked

**TinyLiveSML**

- Removes the locked state from all running program instances.
- Migrates the current state of our running doors program to opened: 2 *

**Implementation.**
TinyLiveSML counts 213 lines of code in Cascade.

- How does this work?



**Program (source code)**

```
machine doors
  opened
    "close" -> closed
  closed
    "open" -> opened
```

| Object | Action | Inverse | Params | | | |
|---|---|---|---|---|---|---|
| Machine | Create | Delete | m | doors | | |
| State | Create | Delete | s3 | locked | m | |
| Trans | Create | Delete | t3 | s2 | lock | s3 |

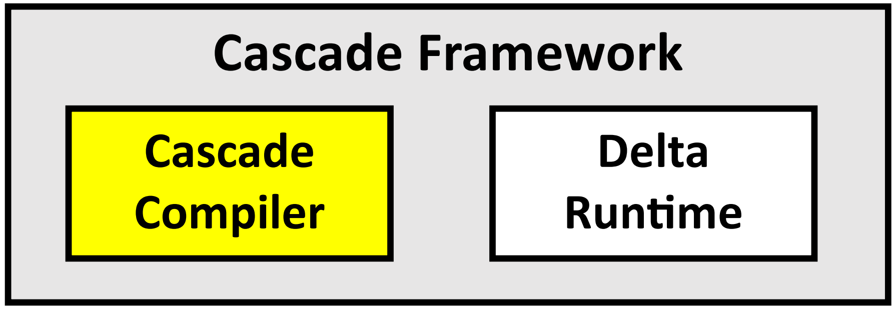
**Running program (run-time state)**

```
machine instance doors
  [close]
  opened : 2 *
  closed : 1
```

| Object | Action | Inverse | Params | |
|---|---|---|---|---|
| MachInst | Create | Delete | mi | m |
| MachInst | Trigger | | mi | lock |

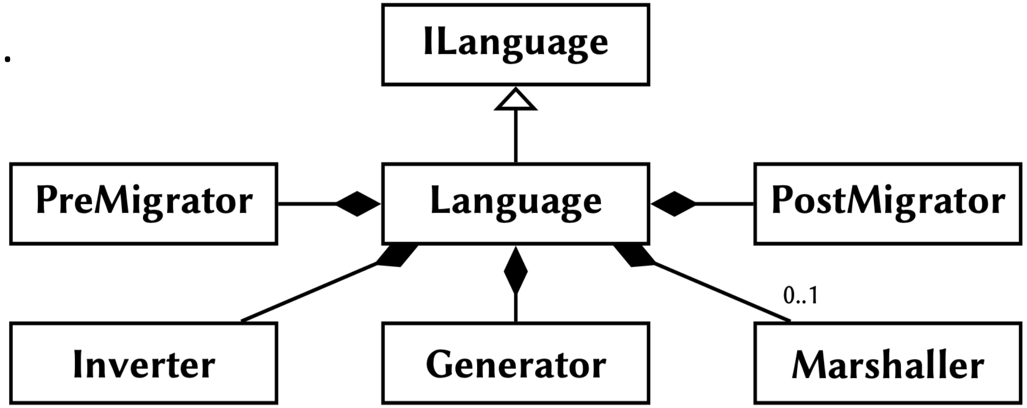
# Cascade Framework

**Prototype live languages.** Obtain a REPL with mechanisms for coding, user interaction & live feedback.



### Cascade compiler

- **Input.** Reads Cascade specs
- **Output.** Generates (C#) language prototypes that integrate with Delta.
- **Platform.** Implemented in Rascal.



### Generated Language Prototypes

- **Generator.**
  Generates transactions.
- **Pre- and Post- Migrators.**
  Migrates models according to Cascade specifications.
- **Inverter.** Inverts effects.

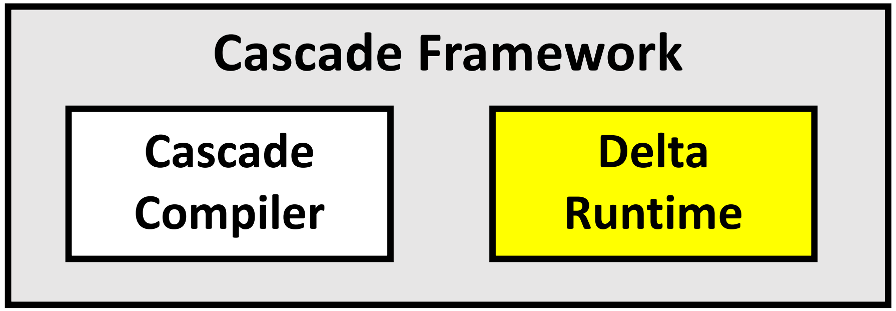
# Cascade Framework

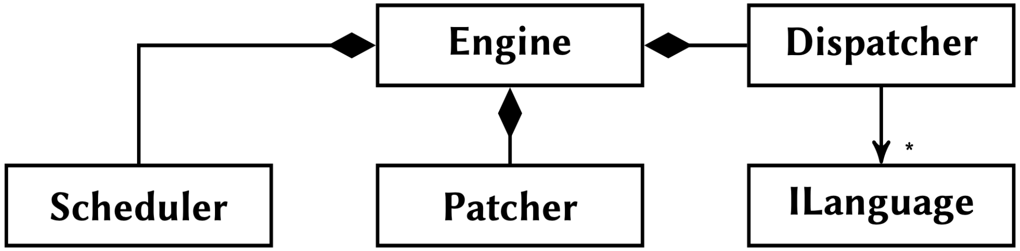
**Prototype live languages.** Obtain a REPL with mechanisms for coding, user interaction & live feedback.

**REPL language.** Enables making user- and coding actions.

**Delta.** Extensible engine (C#) that interprets events and enforces change.

- **Scheduler.** Schedules actions as events.
- **Patcher.** Commits transactions to history.
- **Dispatcher.** Relays events to the right language.





# TinyLiveSML in Cascade

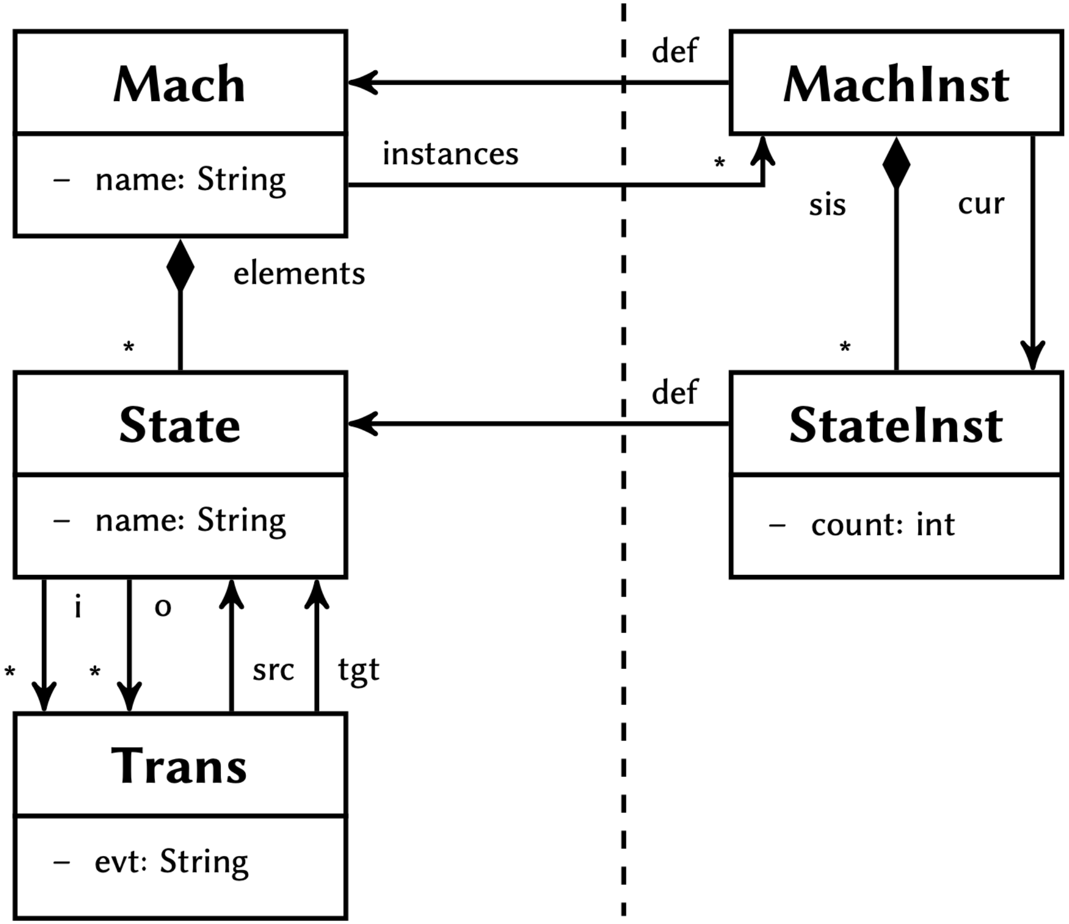
**Model languages.** Express abstract syntax and run-time state with meta-models.

**Static meta-model.** Defines Abstract Syntax Trees (ASTs) of LiveSML programs.

**Run-time meta-model.** Defines the run-time states of running state machines.

**Cascade notation.**

```
class Mach {
  String name;
  Set<State> states;
  Set<MachInst> instances;
```



# TinyLiveSML in Cascade

**Design side-effects.** Express a) *coding actions* that work on abstract syntax; and b) *user actions* that affect run-time state.

**Programmer**

- Creates machine doors
- Delta interprets the effect Create

**Effects.** Consist of edit operations.

**Bidirectionality.** Every effect has an inverse with the opposite effect.

**Side-effects.** Schedule side effects before (pre) or after (post) effects.

```
class Mach {
  …
  effect Create(future Mach m, String name) {
    m = new Mach();
    m.name = name;
    m.states = new Set<State>();
    m.instances = new Set<MachInst>();
  }
  inverse effect Delete(past Mach m,
        String name = m.name) {
    m.name = null;
    delete m.states;
    delete m.instances;
    delete m;
  }
 pre {
    foreach(State s in m.states) {
        State.Delete(s, s.name, m); }
    foreach(MachInst mi in m.instances) {
        MachInst.Delete(mi, m); }
  }
```

# TinyLiveSML in Cascade

**Account for run-time eventualities.** Express *cascading changes* in run-time scenarios,
e.g., run-time state migration.

### Programmer

- Deletes state locked

### TinyLiveSML

- Removes and deletes the locked state from all program instances.
- Reinitializes all instances, which results in migration if and only if a machine no longer has a current state.

```
class Mach {
  …

  side-effect AddState(Mach m, State s) {
    …
  }
  inverse side-effect RemoveState(Mach m,
      State s) {
    m.states.remove(s);
  }
  pre {
    foreach(MachInst mi in m.instances) {
      StateInst si = mi.sis[s];
      MachInst.RemoveStateInst(mi, si, s);
      StateInst.Delete(si, s);
      MachInst.Initialize(mi);
    }
  }
```

# Enabling Live Language Technology

**Objective.** Develop language-parametric enabling technology that powers live programming.

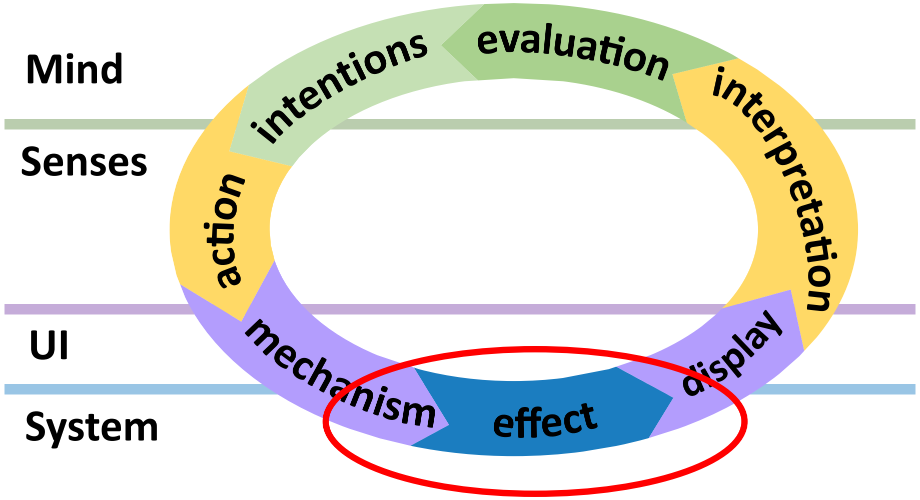
**Contribution.** Cascade, a meta-language for creating languages with interface- and feedback-mechanisms for live programming.

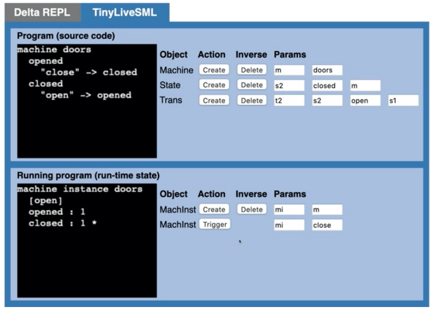
## Challenges addressed

- **Prototype live languages ✓**
- **Model languages ✓**
- **Design side-effects ✓**
- **Account for run-time eventualities ✓**

## Current and future work

- Machinations, behavior trees, questionnaire language
- Visual REPLs and generic live IDEs





Program (source code)

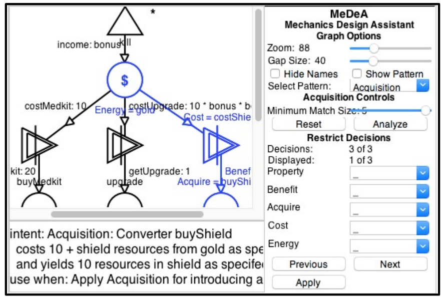
```
machine doors
  opened
    "close" -> closed
  closed
    "open" -> opened
```

| Object | Action | Inverse | Params | | | |
|---|---|---|---|---|---|---|
| Machine | Create | Delete | m | doors | | |
| State | Create | Delete | s2 | closed | m | |
| Trans | Create | Delete | t2 | s2 | open | s1 |

Running program (run-time state)

```
machine instance doors
  [open]
  opened : 1
  closed : 1 *
```

| Object | Action | Inverse | Params | |
|---|---|---|---|---|
| MachInst | Create | Delete | mi | m |
| MachInst | Trigger | | mi | close |



intent: Acquisition: Converter buyShield
 costs 10 + shield resources from gold as spe
 and yields 10 resources in shield as specife
use when: Apply Acquisition for introducing a

MeDeA
Mechanics Design Assistant
Graph Options
Zoom: 88
Gap Size: 40
□ Hide Names □ Show Pattern
Select Pattern: Acquisition
Acquisition Controls
Minimum Match Size: 5
Restrict Decisions
Decisions: 3 of 3
Displayed: 1 of 3
Property _
Benefit _
Acquire _
Cost _
Energy _

**Cascade is available under 3-clause BSD license**
https://github.com/vrozen/Cascade