Create and Play your PacMan Game with the GEMOC Studio

Dorian Leroy ¹ Erwan Bousse ² Manuel Wimmer ² Benoit Combemale ³
Wieland Schwinger ¹

¹JKU Linz
²TU Wien
³University of Toulouse (UT2J)

September 17th 2017
Behavioral models often need to interact with the outside world during their execution, e.g. to process incoming domain-level event occurrences.

- Adds complexity to the operational semantics of a DSL:
  - impacts content and scheduling of execution rules,
  - requires an interruption mechanism,
  - requires an interface allowing external actors to send events.

Tedious and error-prone task that must be repeated for each executable DSL.
Context

- Behavioral models often need to interact with the outside world during their execution, e.g., to process incoming domain-level event occurrences.
- Adds complexity to the operational semantics of a DSL:
  - impacts content and scheduling of execution rules,
  - requires an interruption mechanism,
  - requires an interface allowing external actors to send events.

Tedious and error-prone task that must be repeated for each executable DSL.
Behavioral models often need to **interact with the outside world** during their execution, e.g., to process incoming domain-level event occurrences.

- Adds complexity to the operational semantics of a DSL:
  - impacts content and scheduling of execution rules,
  - requires an interruption mechanism,
  - requires an interface allowing external actors to send events.

**Tedious and error-prone** task that must be repeated for each executable DSL.
Example: The PacMan DSL

- **Abstract Syntax**
  - Board
  - Entity
  - Tile
  - WallTile
  - PassableTile

- **Execution Metamodel**
  - AbstractTile
  - PassableTile
  - SuperPellet
  - Pellet
  - Ghost
  - Pacman

- **Execution Rules**
  - run(Board: board)
  - update(Board: board, int: deltaTime)
  - update(Entity: entity, int: deltaTime)
  - enterNextTile(Entity: entity)
  - modifySpeed(Entity: entity, int: speed)
  - activate(Ghost: ghost)
  - energize(Pacman: pacman)
  - up(Pacman: pacman)
  - down(Pacman: pacman)
  - left(Pacman: pacman)
  - right(Pacman: pacman)
Example: The PacMan DSL

**Abstract Syntax**

- **entities**: 0..*
- **tiles**: 0..*
- **Board**: AbstractTile
- **PassableTile**: WallTile
- **Entity**: Pacman, Ghost
  - +initialTile
- **Tile**: GhostHouseTile
- **Execution Rules**
  - run(Board: board)
  - update(Board: board, int: deltaTime)
  - enterNextTile(Entity: entity)
  - modifySpeed(Entity: entity, int: speed)
  - up(Pacman: pacman), down(Pacman: pacman), left(Pacman: pacman), right(Pacman: pacman)

**Execution Metamodel**

- **AbstractTile**: 0..1
- **targetTile**: 1
- **currentTile**: 0..1
- **pellet**: 0..1
- **Entity**: PassableTile, Ghost
  - +speed: int
  - +initialTile
- **Tile**: SuperPellet, Pellet
- **Pacman**: +pelletsEaten: int, +lives: int, +energized: boolean
- **Ghost**: +energized: boolean

**How to safely call these execution rules, while main execution loop of the game is ongoing?**
Problem and Idea

How to avoid rewriting operational semantics to define *domain-specific events* that may safely *interrupt* the execution flow?

Idea

Take the tedious and repetitive part out of the hands of the language engineer by providing:

- An annotation mechanism to easily define events,
- The generation of an interface to send events at runtime,
- Generic event management reusable across DSLs.
Problem and Idea

How to avoid rewriting operational semantics to define **domain-specific events** that may safely **interrupt** the execution flow?

Idea

Take the tedious and repetitive part out of the hands of the language engineer by providing:

- An annotation mechanism to easily define events,
- The generation of an interface to send events at runtime,
- Generic event management reusable across DSLs.
Summary of the Approach

- A generative approach to obtain a **domain-specific event language** and its interpreter from annotated execution rules,
- A **generic event queue manager**, incorporating event queuing and dispatch into the execution loop.
Event Metamodel and Interpreter Generation

- The generative approach relies on annotated execution rules to locate event handlers and event preconditions.
- Event metaclases are generated from event handlers to populate the event metamodel.
- An event interpreter mapping instances of event metaclases to event handler and precondition calls is generated.
Event Metamodel and Interpreter Generation

- The generative approach relies on annotated execution rules to locate event handlers and event preconditions.
- Event metaclasses are generated from event handlers to populate the event metamodel.
- An event interpreter mapping instances of event metaclasses to event handler and precondition calls is generated.
The generative approach relies on annotated execution rules to locate event handlers and event preconditions.

Event metaclasses are generated from event handlers to populate the event metamodel.

An event interpreter mapping instances of event metaclasses to event handler and precondition calls is generated.
Demo
Conclusion & Future Work

- Adapting semantics to event handling is difficult
- Proposed solution:
  - non-intrusive annotation of execution rules and generation of a
  - generation of an interface to inject event occurrences
  - reuse of an event queue and interruption mechanism

- Eclipse Research Consortium GEMOC: sustains the GEMOC studio as a research platform to experiment on the globalization of, possibly executable and heterogeneous, modeling languages
- Contributors are welcome!

http://gemoc.org/
https://github.com/eclipse/gemoc-studio-modeldebugging
Conclusion & Future Work

- Adapting semantics to event handling is difficult
- Proposed solution:
  - non-intrusive annotation of execution rules and generation of an interface to inject event occurrences
  - reuse of an event queue and interruption mechanism

- **Eclipse Research Consortium GEMOC**: sustains the GEMOC studio as a research platform to experiment on the globalization of, possibly executable and heterogeneous, modeling languages
- Contributors are welcome!

http://gemoc.org/
https://github.com/eclipse/gemoc-studio-modeldebugging