The new Icobjs Framework

Christian Brunette

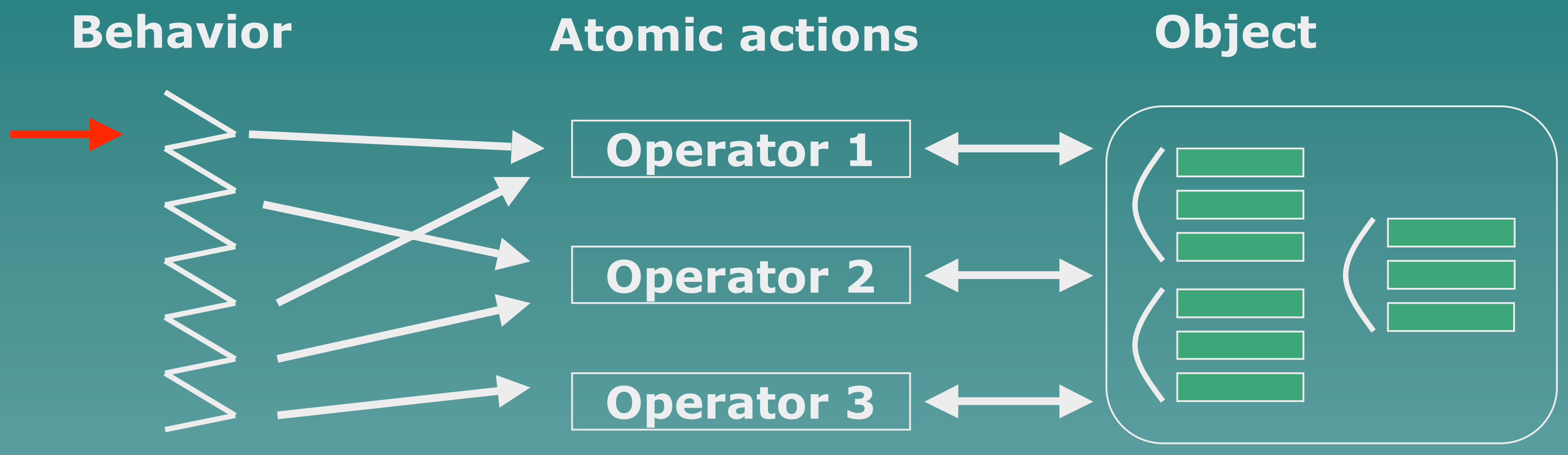
INRIA/ENSMP - MIMOSA

Icobjs in the past

Icobjs

- Means “Iconic Objects”
- Is a means to build at runtime entity behaviors by graphical combination
- Based on Junior, a Java API

Problems



- The data structure was totally sealed (basically due to OO approach)
- There were no clear API of an icobj

Needs

- A clear API
- A means to modify icobj behaviors after graphical construction
- A means to save created simulations and created icobjs
- Some new instructions
- Some optimizations of the reactive engine

Outline

- The model
- The reactive engine
- The framework
- Experimentations

Icobjs model

- An icobj is a graphical reactive entity composed of:
  - 2 identifiers: one for the entity and one for its “container”
  - Graphical information: appearance and space taken in its “container”
  - Behavior information: Cloneable and not-Cloneable
  - A hash table to store other fields

Some advices

- Initialize Icobj fields with their behaviors
- Access Icobj fields only through atomic actions
- Do not keep states in atomic actions

  ➔ use Icobj fields

Workspace

"Workspaces are to icobjs what reactive machines are to reactive instructions"

- Is the container of icobj
- Executes icobj behaviors
- Manages the graphical part
- Deals with interactions with "external world" (end-user, network…)
- Events are local to the Workspace
- Is an icobj

Migration

- Migration =

  exiting + transferring + entering
- Two kinds of migration:
  - Local = from a local workspace to another one (same thread)
  - Through the network (different threads)
- Need to wait the end of instant
- Take at least one instant

Outline

- The model
- The reactive engine
- The framework
- Experimentations

Reactive engine

- Junior instructions:
  - Nothing, Stop, Seq, Par, Loop, Repeat, If
  - Await, Generate, Until, Control, Freezable, Local, When
  - Link
- Based on the Storm implementation of Junior (J-F Susini)
  - 4 status: SUSP, TERM, STOP, WAIT
  - “zap precursor” algorithm

Added Instructions

- Run: evaluates at runtime and executes a reactive program

  ex: local migration

- Scanner: executes an atomic action associated to each occurrence of a valued event

  ex: interactions with “external world” (mouse, keyboard…)

Added Instructions

- Kill: weak preemption (SL)
  - More regular/modular than the Until instruction in Junior
  - Until still exists…
- IcobjThread:
  - add new instructions dynamically to the dedicated icobj
  - make the remove/migration of icobj behaviors faster

Engine modification

- LONGWAIT:
  - New instruction status
  - inter-instant waiting
- SeqN/ParN:
  - one control of sequential/parallel instructions
  - to clean terminated instructions

Event management

- An event is added to the environment when:
  - It is generated (internally or externally)
  - An instruction waits for it
- Keep events and values during 2 instants after their generations
- Need a mechanism to remove unused events from the environment
  - Faster event search
  - Less memory used

Outline

- The model
- The reactive engine
- The frameworks
- Experimentation

Behavior inspector

- To inspect the behaviors of icobj
- No direct access to instructions executed in the reactive engine
- No modification during a reaction
- Change behaviors after construction
  - Changing the behavior fields of icobj
  - Removing the executing behavior from the reactive engine
  - Loading the new behavior

Introspection

- Allow to modify values of icobjs fields at runtime
- Fields are only changed between two reactions
- User must implements on each icobjs Field class:

  Paremeter[] getParemeter(Icobj self)

  Serializable getValue(String fieldName)

  void setValue(String fieldName, Serializable value)

Load/Save

- Load/save = migration
- Saving =

  exiting + serialization in a file
- Loading =

  deserialization + entering
- These operations are controlled by the Workspace

Outline

- The model
- The reactive engine
- The framework
- Experimentations

Experimentation (1)

- Physics (cf. A. Samarin)
  - a physical reaction = 2 engine reactions
    - one instant to gather all physical events
    - one instant to compute the result
  - behaviors synchronized by an event
  - Remaining problems
    - loss of precision: due to data types
    - not very modular: the computation has not to exceed one instant

Experimentation (2)

- Multi-clock simulation
  - 2 reactive engines in the same Workspace
  - Each Workspace reaction consists in:
    - 4 reactions of the physical engine
    - 1 reaction of the basic engine
  - Events generated in each reactive engine are local to it
  - Events generated in the workspace are generated in the two reactive engines

Conclusion

- A new model and dedicated API for Icobjs
- New reactive engine with new instructions
- A framework to create/inspect icobjs
- Some experimentations on physical and multi-clock simulations
- Website:

  http://www.inria.fr/mimosa/rp/Icobjs/

Future works

- Implements migration through network
- Integrate the distribution in the framework
- Interface with a 3D engine