Resource Entity Action: A Generalized Design Pattern for RTS games

Mohamed Abbadi Francesco Di Giacomo Renzo Orsini
Aske Plaat Pieter Spronck Giuseppe Maggiore
E-mail: {mabbadi,fdigiacomo,orsini}@dais.unive.it,
{a.plaat,p.spronck}@uvt.nl, maggiore.g@nhtv.nl

Università Ca' Foscari DAIS - Computer Science, Venice, Italy
Tilburg University, Netherlands
NHTV University of Applied Sciences, Breda, Netherlands

Table of contents

Introduction to the problem

Encoding

Building real-time video-games, fast

Simple game definition through high level patterns

High performance through automated optimization

Save a lot of work, do not decrease quality

Especially important for serious/research games

Introduction to the problem

Casanova so far

- Declarative language
- Centered on (real-time) games
- Design philosophy: few and simple orthogonal concepts, all game-focused
  - Game world, entities
  - Rules
  - Scripts

Introduction to the problem

```
world BouncingBall = {
  Sprite   : Sprite
  Position : Vector2<m>
  Velocity : Vector2<m/s>

  rule Sprite.Position'(world:BouncingBall) =
    world.Position
  rule Position'(world:BouncingBall, dt:float<s>) =
    world.Position + world.Velocity * dt
  rule Velocity'(world:BouncingBall, dt:float<s>) =
    world.Velocity - Vector2.UnitY * 9.81<m/s^2> * dt
}
```

Introduction to the problem

```
let world =
  { Sprite = ...; Position = ...; Velocity = ... }

let main = return ()

let input =
  [
    wait_key_press Keys.Escape => quit()
    wait_key_down Keys.Space  =>
      co{
        world.Position := Vector2.Zero
        world.Velocity := Vector2.Zero
        wait_key_up Keys.Space || wait 0.2<s>
      }
  ]
```

Introduction to the problem

Our current focus

- RTS games
- Their high relevance
  - Large commercial adoption
  - Potential high impact outside entertainment
- Their issues: such games are quite hard to build

Introduction to the problem

Building RTS games

- Traditional engines/software engineering
- At most, reuse of the game core, plus restyling
- Not flexible
- Compared to other genres (FPS!), little code reuse

Generalizing RTS games

Generalizing RTS games

Building a simple algebra for entities and resources

Entities - a container of resources and other attributes

Resources - a (usually sparse) vector

Actions - a (usually sparse) matrix of resources conversion

Generalizing RTS games

Source e, targets $T = \{t_1, t_2, ..., t_n\}$, action A (a transformation matrix).

Each entity has resource vector $\mathbf{r_i} = (r_{i_1}, r_{i_2}, ..., r_{i_m})$

We compute $\mathbf{w_e} = (w_{e_1}, w_{e_2}, ..., w_{e_m}) = \mathbf{r_s} \times A \cdot dt$.

We compute the new target resource vectors $\mathbf{r_i'} = \mathbf{r_i} + \mathbf{w_e}\ \forall e_i \in E$.

Generalizing RTS games - example

Spaceship resources = { laser; life points }

Source ship is $r_s = (20, 500)$, targeted ship is $r_t = (15, 1000)$

Transformation matrix =

$$A = \begin{bmatrix} 0 & -1 \\ 0 & 0 \\ & \end{bmatrix}$$

Thus $w_e = r_s \times A \cdot dt = (20, 500) \times A \cdot dt = (0, -20) \cdot dt$. At this point, assuming $dt = 1$ second, we have $r'_t = r_t + w_e = (20, 1000) + (0, -20) \cdot dt = (20, 980)$.

Generalizing RTS games

About this generalization

Shows that a generalization is possible

Even with it, defining remaining game aspects is very verbose:

- Complex *action completed* responses (spawn a new unit, destroy current unit, etc.)
- Finding the current *target entities*

Generalizing RTS games

Towards an extension

Resources algebra is a good take

We need a mechanism for finding target entities

Our model

Extended model

- Resource algebra paired with SQL queries
- Support for optimizations of `joins` on spatial predicate

Our model

Extended model

Three kinds of actions

- Constant - just increase/decrease target attributes
- Mutable - convert from source to target
- Threshold - convert from source to target until a certain threshold

Our model

```
TARGET Infantry; RESTRICTION Owner <> Owner; RADIUS
    1000.0; TRANSFER CONSTANT Life - ArrowDamage;

TARGET Shipyard; RESTRICTION Owner = Owner; RADIUS
    150.0; TRANSFER MineralStash + Minerals;

TARGET Construction; RESTRICTION Owner = Owner; RADIUS
     10.0; TRANSFER CONSTANT Integrity + 1.0;
    THRESHOLD Integrity = 100.0; OUTPUT Completed :=
    true
```

Syntax

```
<Action> ::= TARGET <TARGET LIST> <RESTRICTION LIST>
    [<RADIUS CLAUSE>] <TRANSFER LIST>
   <INSERT LIST> [<THRESHOLD BLOCK>]
<TARGET LIST> ::= <ACTION ELEMENT>+
<ACTION ELEMENT> ::= Casanova Entity | Self
<RESTRICTION LIST> ::= {<RESTRICTION CLAUSE>}
<RESTRICTION CLAUSE> ::= RESTRICTION Boolean
    Expression of <SIMPLE PRED>
<SIMPLE PRED> ::= Self Casanova Entity Field (= | <>)
    Target Casanova Entity Field
<TRANSFER LIST> ::= {<TRANSFER CLAUSE>}
<TRANSFER CLAUSE> ::= (TRANSFER | TRANSFER CONSTANT)
(Target Casanova Entity Field) <Operator> ((Self
    Casanova Entity Field) | (Field Val)) [* Float Val
    ]
<Operator> ::= + | - | :=
```

Syntax

```
<RADIUS CLAUSE> ::= RADIUS (Float Val)
<INSERT LIST> ::= {<INSERT CLAUSE>}
<INSERT CLAUSE> ::= INSERT (Target Casanova Entity
    Field) -> (Self Casanova Entity Field List)
<THRESHOLD BLOCK> ::= <THRESHOLD CLAUSE>+
<OUTPUT CLAUSE>+
<THRESHOLD CLAUSE> ::= THRESHOLD
(Self Casanova Entity Field) Field Val
<OUTPUT CLAUSE> ::= OUTPUT
(Self Casanova Entity Field) <Operator> ((Self
  Casanova Entity Field) | (Field Val)) [* Float Val]
```

Our model

Semantics

Translation from above into SQL

Then lifting from relevant SQL literature!

Semantics for constant transfers - 1

```
SELECT t_i.id, SUM(s.a_j1} AS Σ_1,
SUM(s.a_j2} AS Σ_2,...,
SUM(s.a_jm} AS Σ_m

FROM Target t_i, Source s
WHERE <RESTRICTION LIST> [AND <RADIUS CLAUSE>]
GROUP BY t_i.id
```

Semantics for constant transfers - 2

```
WITH Transfer AS(
SELECT t_i.id, SUM(s.a_j1} AS Σ_1,
SUM(s.a_j2} AS Σ_2,...,
SUM(s.a_jm} AS Σ_m)

FROM Target t_i, Source s
WHERE [<RESTRICTION LIST>] [AND <RADIUS CLAUSE>]
GROUP BY t_i.id)
UPDATE Target t_i
SET t_i.a_t1 = u.Σ_1 | t_i.a_t1 = t_i.a_t1 + u.Σ_1 * dt | t_i.a_t1 =
t_i.a_t1 - u.Σ_1 * dt
...
FROM Transfer u
WHERE u.id = t_i.id
```

Our model

Other semantics

Omitted for brevity, but all feature:

- Fetching of targets and evaluation of predicates
- Computation of new resources
- Update of source and target resources

Evaluation

Evaluation

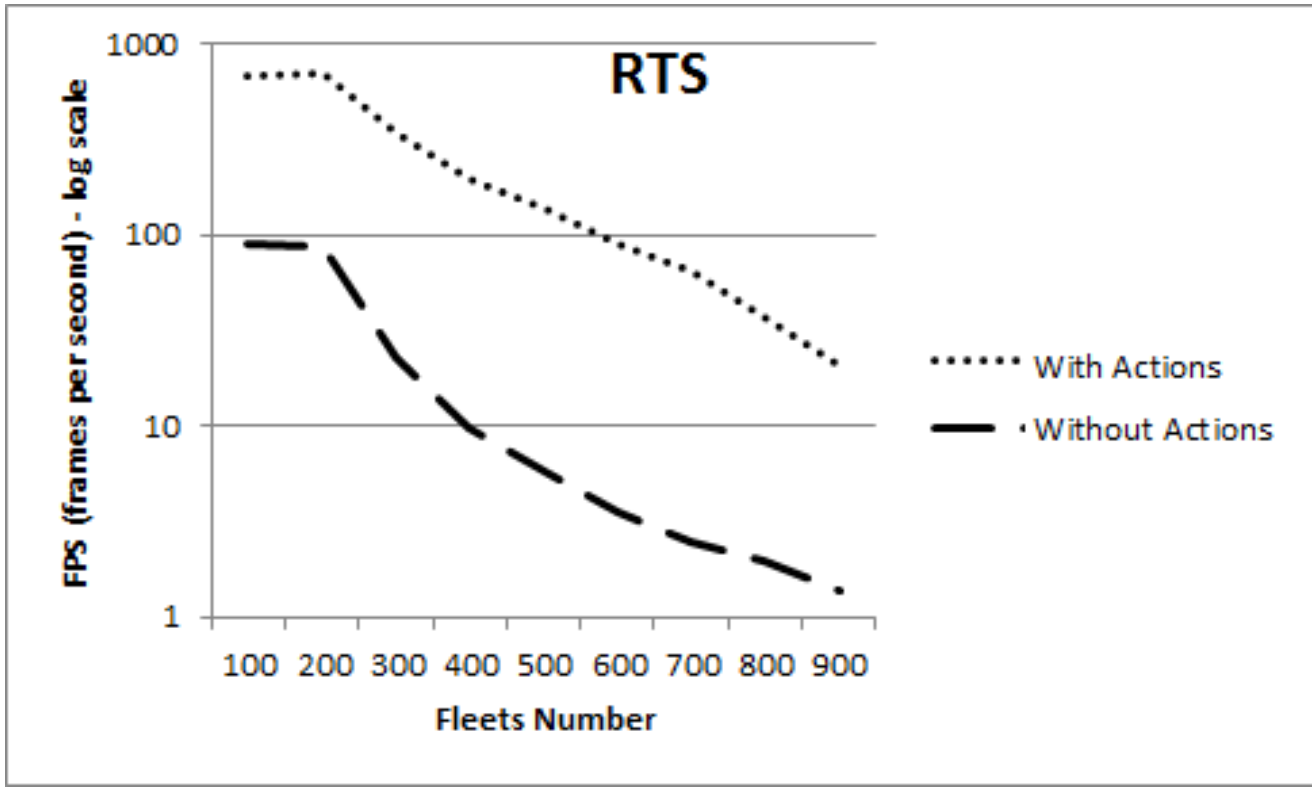
- Program length
- Performance

Evaluation

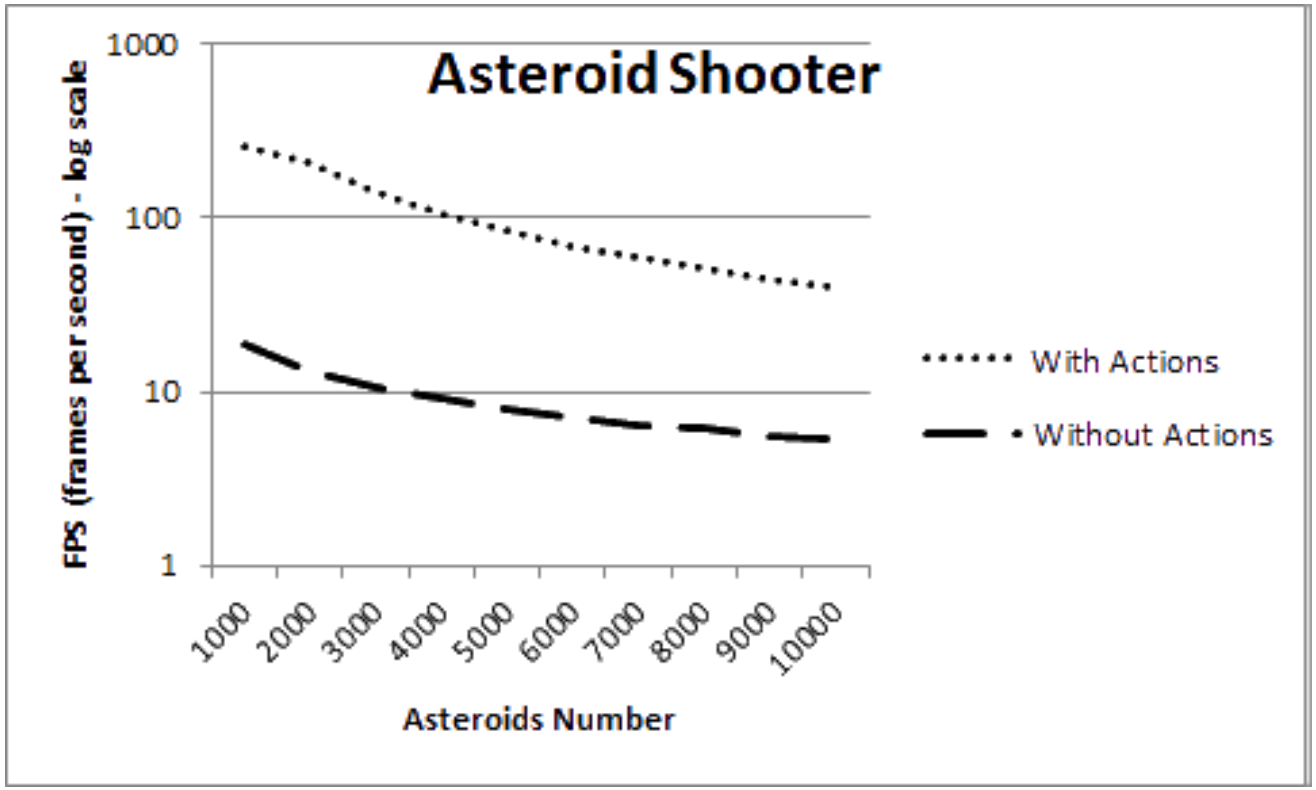
Table : CS (case study), Asteroid Shooter and Expanded CS code length

	Game Entities	Rules	Actions	Total
CS with REA	41	71	19	131
CS without REA	40	90	0	130
Asteroid shooter with REA	33	33	6	72
Asteroid Shooter without REA	34	44	0	78
Extended CS with REA	135	138	40	313
Extended CS without REA	135	328	0	463

Evaluation


RTS
1000
100
10
1
FPS (frames per second) - log scale
100 200 300 400 500 600 700 800 900
Fleets Number
With Actions
Without Actions

Evaluation


Asteroid Shooter
FPS (frames per second) - log scale
1000
100
10
1
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
Asteroids Number
With Actions
Without Actions

Evaluation

Beyond RTS games

Also works for other kinds of games

- Shooters
- RPGs
- Pac-Man
- Tetris
- ...

Closing

Discussion

We set out with the goal of reducing development effort for RTS games

The results are:

- Shorter, simpler sources
- Significantly faster execution, without hand-made optimizations

We have built a well working prototype of Casanova embedded in F#

Closing

Future work

We are currently working on Casanova 2.0

Syntactic and semantic integration of rules, coroutines, and SQL

- Everything is done with rules
- Every expression can interrupt, `yield` or `wait`
- SQL is a first class expression
- SQL queries are optimized with automated indices on `joins` and similar "hotspots"

That's it!

Thank you!

Questions?