Distributed simulation of situated multi-agent systems

Franco Cicirelli, Andrea Giordano, Libero Nigro

Laboratorio di Ingegneria del Software
http://www.lis.deis.unical.it

Dipartimento di Elettronica Informatica e Sistemistica
Università della Calabria
87036 Rende (CS) – Italy

{f.cicirelli,a.giordano}@deis.unical.it, l.nigro@unical.it
Goal

- proposing an approach and supporting framework, for **modelling and distributed simulation** of complex **situated multi-agent systems**

Presentation Outline

- **introducing** situated agents
- **discussing** distributed-simulation of situated agents
- **describing** the proposed approach and the developed framework
- **showing** achievable performance by means of a **TileWorld**-based simulation model
Situated agents

- are characterized by being embedded in a spatial environment (or territory) and by owning spatial coordinates.
- their behaviour is strongly influenced by the owned position into the environment.
- are able to move on, perceive and act-upon the territory.
- emerging properties of modelled systems may arise by agent-to-agent or agent-to-environment interactions.
Distributed simulation of situated agents

- situated agents are widely adopted for studying a broad range of phenomena and systems (e.g. in biology, sociology, wildfires)

- distributed simulation is often mandatory to cope with the high resource demand (both in terms of time and space) of such large models

- in a distributed context the environment becomes a huge shared variable of a concurrent system

- suitable environment partitioning-schemas and approaches regulating the access to the environmental data are required (e.g. for load balancing, data consistency, performance)
Distributing the environment (1)

A first scenario:

- the territory is modelled as a bi-dimensional grid
- limited sensing/control capabilities
- a hypothesis:
  - each cell may host at most one agent (a conflict occurs otherwise)
- sequential simulation and cooperative concurrency
A **second scenario:**

- distributed simulation (2 LPs)
- territory and agent population are split
- conflicts occur on red highlighted cells (real parallelism)
- remote communication is required between LPs
Distributing the environment: a solution (1)

Reducing remote communication:
- border areas are replicated (gray parts)
- their size depend on visibility radius
Avoiding conflicts:

- a *Conflict Free* execution order is enforced among agents residing on different LPs
- conflicting agents are not allowed to act concurrently
- no control messages are exchanged among LPs
- no locks are used
Avoiding conflicts:

- Cells on border areas are tagged with Conflict Free Numbers (CFNs).
- Two agents which are distant less than or equal to 2*action radius and belong to different LPs must be flagged with a different CFN.
- CFNs are used to define a conflict free execution order among agents.
- Agents on different LPs having the same CFNs can really act in parallel.
- Tagging schema depend on action radius.
Avoiding conflicts:
- a repetitive pattern, favouring CFN reuse, is used (shuffled from time to time)
- CFN reuse favours parallelism

- the assigning algorithm scans the border area from top to bottom and from left to right
- the same assignment (despite shuffling) is made by two neighbouring LPs without requiring any interaction by using the current logical time as the seed for the pseudo random number generators
Enforcing the conflict-free execution order: the composite logical time (CLT)

- CLT manages multiple events at the same virtual time for conflict resolution.
- Virtual time is the virtual time from the model point of view.
- Every time an event (message) is ready to be dispatched, its actual delivery occurs at a time which takes into account the CFN of the receiving agent.
- Actions triggered by such event cannot cause conflicts.
Distributing the environment: a solution (5)

- Territory and agent population are split among various LPs
- Each territory portion is called **region**
- The edge portions of adjacent regions (**borders**) are mirrored among LPs and kept updated
- **Action** and **visibility** radii are introduced (**neighborhood**)
- Such radii delimit the area within which an agent can **efficiently read** and **change** the status of the immediate surrounding territory
- Conflict resolution and data consistency rely on a **composite logical time** notion which makes use of CFN
- **Migration** is required when an agent moves among regions
Distributing the environment: the supporting framework

is added into the **Theatre** multi-agent architecture

relies on a lightweight **actor** model of computation

promotes the environment as **first-class entity** (ActorEnv)

allows the sharing of agent state

simplifies M&S tasks by making the modeller **unaware** of distribution concerns

within the agent **neighborhood**, it permits **synchronous** access to the environment (asynchronous otherwise)
The supporting framework: the Neighborhood interface

Neighbourhood n = ActorEnv.getMyNeighbourhood();

Actor createAndLocate(String actClass, Position p);
void moveActor(Actor act, Position p);
void removeActor(Actor act);
List<Actor> getCell(Position p, Class actClass);
boolean isCellEmpty(Position p);
List<Actor> getActors(Class actClass);
Position getPosition(Actor act);
void addShared(String name, Class type);
<T> T getShared(String name, Class<T> type, Actor act);
<T> void setShared(String name, Class<T> type, T value, Actor act);
A TileWorld Model as testbed

- agent’s mission: move around to find and pick-up a tile and then move to fill an hole and so forth until no more tiles exist
- an hole is characterized by its depth
- holes and tiles may appear and disappear dynamically
- different game configurations with a huge number of randomly placed TileWorldActor(s), TileActor(s), HoleActor(s), and static obstacles, were experimented
- different values for the action and visibility radii were considered
- the goal was not to compare agent strategies but only to check the achievable simulation performance
A code excerpt of the TileWorldActor

```java
//status MOVE_TO_HOLE
Neighbourhood n = ActorEnv.getMyNeighbourhood();
if( n.getShared("visible",Boolean.class,this.foundHole) ){
    nextPosition = makeAStepTowardAHole(this.foundHole);
    if (nextPosition.isReached( n.getPosition(this.foundHole) )){//fill the hole
        int d = n.getShared("depth",Integer.Class,this.foundHole);
        n.setShared("depth",Integer.Class,d-1,this.foundHole);
        if (n.getShared("depth",Integer.Class,this.foundHole)==0){
            n.setShared("visible",Boolean.Class,false,this.foundHole);
            this.score += n.getShared("score",Double.Class,this.foundHole);
            become(LOOK_FOR_TILE);//change status
        }
    }else
        if (!n.getCell(nextPosition,ObstacleActor.class).isEmpty())
            nextPosition = changeDirection();//avoid the obstacle
            n.moveActor(this,nextPosition);
}else{//Explore
    this.foundHole = null;
    for(HoleActor a:
        n.getActors(HoleActor.class))
        if (n.getShared("visible",Boolean.class,a)){
            this.foundHole = a;
            nextPosition = makeAStepTowardAHole(this.foundHole);
            n.moveActor(this,nextPosition);
            break;
        }
    if (this.foundHole == null)
        become(LOOK_FOR_HOLE);//change status
}
```
three LPs/federates allocated on three WinXP (32 bit) Intel i7 CPU 960, 1-core, 3.20 GHz, 3GB RAM, interconnected by a Gigabit Ethernet switch in the presence of HLA pRTI 1516

each federate hosts a region composed of 1200x900 cells (overall territory is 1200x2700 cells)
The **variable-load scenario**: is it appropriate?

![Graph 1](image1.png)

1. Normalized Wallclock Time [sec./f.u.]
   - 3LPA(250k)
   - 3LPA(100k)

![Graph 2](image2.png)

2. Speedup (3 LPS)
   - 250k TileWorldActor(s)

![Graph 3](image3.png)

3. Speedup (3 LPS)
   - 100k TileWorldActor(s)
Speedup vs. Action and Visibility radii (constant load)

1. Normalized Wallclock Time [sec./f.u.]
   - 150k TileWorldActor(s) - 3 LPs - AR=5
   - Graphs showing the relationship between normalized wallclock time and visibility radii (VR) for different action radii (AR).

2. Speedup (3 LPs)
   - Graphs showing speedup as a function of visibility radii for different action radii.

   - 150k TileWorldActor(s) - 3 LPs - VR=200
   - Similar graphs as in 1 but for VR=200.

4. Speedup (3 LPs)
   - Similar graphs as in 2 but for VR=200.
On-going and future work

- experimenting with the use of the infrastructure in large and highly dynamic systems (e.g. inspired by biology or social science)

- improving the ActorEnv interface (e.g. by allowing a more fine grained read/write control on shared data and by providing a more complex pattern-matching schema for neighbourhood exploration)

- specializing the approach so as to exploit the potential of modern multi-core hardware

- generalizing territory management (e.g. toward hexagonal space cells, continuous spaces, diffusive spaces and n-dimensional spaces)