Distributed Simulation of RePast Models over HLA/Actors

Franco Cicirelli, Angelo Furfaro, Andrea Giordano, Libero Nigro

Laboratorio di Ingegneria del Software
Dipartimento di Elettronica Informatica e Sistemistica
Università della Calabria, 87036 Rende(CS) - Italy
Email: {f.cicirelli,a.furfaro,a.giordano}@deis.unical.it, l.nigro@unical.it
Web: http://www.lis.deis.unical.it

13th IEEE/ACM International Symposium on
Distributed Simulation and Real Time Applications (DS-RT 2009)
October 25-28, Singapore
Goal

- Distributed simulation of RePast agent-based models initially designed for centralized execution, thus

Approach

Adoption of HLA as middleware for distributed simulation and time management

Use of technologies such as Aspect Oriented Programming and Java Meta-data (Annotations) in order to minimize “code intrusions” in the original model

Mapping RePast models on to Theatre actors thus enabling the exploitation of migration mechanisms and efficient communications
Goal

- Distributed simulation of RePast agent-based models initially designed for centralized execution, thus
- allowing to exploit the computational potentials (in space and time) of networked contexts for large MAS models

Approach
Goal

- Distributed simulation of RePast agent-based models initially designed for centralized execution, thus
- allowing to exploit the computational potentials (in space and time) of networked contexts for large MAS models

Approach

- Adoption of HLA as middleware for distributed simulation and time management
Outline

Goal

- Distributed simulation of RePast agent-based models initially designed for centralized execution, thus
- allowing to exploit the computational potentials (in space and time) of networked contexts for large MAS models

Approach

- Adoption of HLA as middleware for distributed simulation and time management
- Use of technologies such as Aspect Oriented Programming and Java Meta-data (Annotations) in order to minimize “code intrusions” in the original model
Outline

Goal
- Distributed simulation of RePast agent-based models initially designed for centralized execution, thus
- allowing to exploit the computational potentials (in space and time) of networked contexts for large MAS models

Approach
- Adoption of HLA as middleware for distributed simulation and time management
- Use of technologies such as Aspect Oriented Programming and Java Meta-data (Annotations) in order to minimize “code intrusions” in the original model
- Mapping RePast models on to Theatre actors thus enabling the exploitation of migration mechanisms and efficient communications
RePast modelling concepts

RePast entities

A RePast system is made up of:

- **a model** object, containing general information about the system (e.g. configuration information)
A RePast system is made up of:

- a **model** object, containing general information about the system (e.g. configuration information)
- a collection of **agents**
A RePast system is made up of:

- a **model** object, containing general information about the system (e.g. configuration information)
- a collection of **agents**
- a set of **spaces** objects, modelling the environment within which the agents are situated.
A RePast system is made up of:

- a **model** object, containing general information about the system (e.g. configuration information)
- a collection of **agents**
- a set of **spaces** objects, modelling the environment within which the agents are situated.

The state of a system is scattered among agents, spaces and model objects.
RePast runtime scenario

RePast events are modelled at runtime as *action* objects (instances of `BasicAction` class)

![Diagram of RePast runtime scenario]

- Display
- Application
- Space 0, Space 1, Space M
- Agent 0, Agent 1, Agent N
- Scheduler
- MODEL
- Executive
- Controller

Actions are scheduled to occur at specified simulation times (ticks). Action dispatch triggers the execution of a method chain which causes state changes in agents, spaces or the model.

Space objects (envs) can be:
- *data* (diffusive)
- *object* (agent)
RePast runtime scenario

RePast events are modelled at runtime as *action* objects (instances of `BasicAction` class)

Actions are scheduled to occur at specified simulation times (*ticks*)
RePast runtime scenario

- RePast events are modelled at runtime as *action* objects (instances of `BasicAction` class)
- Actions are scheduled to occur at specified simulation times (*ticks*)
- Action dispatch triggers the execution of a method chain which causes state changes in agents, spaces or the model

### Diagram

- **Application**
  - Space 0, Space 1, Space M
  - Agent 0, Agent 1, Agent N
- **Scheduler**
- **Controller**
- **Executive**
- **Display**

**Interactions**:
- Schedule/Dispatch
- Get/put
- Execute

**Spaces**:
- Data (diffusive) spaces
- Object (agent) spaces
RePast runtime scenario

RePast events are modelled at runtime as action objects (instances of BasicAction class).

Actions are scheduled to occur at specified simulation times (ticks).

Action dispatch triggers the execution of a method chain which causes state changes in agents, spaces or the model.

Space objects (envs) can be:
RePast runtime scenario

RePast events are modelled at runtime as *action* objects (instances of `BasicAction` class)

Actions are scheduled to occur at specified simulation times (*ticks*)

Action dispatch triggers the execution of a method chain which causes state changes in agents, spaces or the model

Space objects (envs) can be:
- data (*diffusive*) spaces
RePast events are modelled at runtime as action objects (instances of BasicAction class).

Actions are scheduled to occur at specified simulation times (ticks).

Action dispatch triggers the execution of a method chain which causes state changes in agents, spaces or the model.

Space objects (envs) can be:
- data (diffusive) spaces
- object (agent) spaces
RePast runtime scenario (diffusive spaces)

**Runtime scenario**

**Diffuse-Modify-Update loop**

- **Diffuse** – diffusive spaces update themselves synchronously

**Diffuse**

- Diffusive spaces

**Modify**

- Agents can introduce further changes to the data spaces but changes are stored into a temporary copy of the environment

**Update**

- Changes made in the modify phase are actualized on the environment

---

F. Cicirelli, A. Furfaro, A. Giordano, L. Nigro

Distributed Simulation of RePast Models over HLA/Actors

IEEE/ACM DS-RT 2009
RePast runtime scenario (diffusive spaces)

Runtime scenario

- **Display**
- **Scheduler**
- **Controller**
- **MODEL**
- **SimModel Interface**
- **Agent 0**
- **Agent 1**
- **Agent N**
- **Space 0**
- **Space 1**
- **Space M**
- **Application**

Diffuse-Modify-Update loop

- **Diffuse** – diffusive spaces update themselves synchronously
- **Modify** – agents can introduce further changes to the data spaces but changes are stored into a temporary copy of the environment

F. Cicirelli, A. Furfaro, A. Giordano, L. Nigro
Distributed Simulation of RePast Models over HLA/Actors
IEEE/ACM DS-RT 2009 5 / 23
RePast runtime scenario (diffusive spaces)

Runtime scenario

Diffusion-Modify-Update loop

- **Diffuse** – diffusive spaces update themselves synchronously
- **Modify** – agents can introduce further changes to the data spaces but changes are stored into a temporary copy of the environment
- **Update** – changes made in the modify phase are actualized on the environment
RePast runtime scenario (object spaces)

**Runtime scenario**

- **Application**
- **Space 0**
- **Space 1**
- **Space M**
- **Display**
- **Agent 0**
- **Agent 1**
- **Agent N**
- **Scheduler**
- **MODEL**
- **SimModel Interface**
- **Controller**
- **Executive**

**Object Spaces**

- Agents access object spaces using *get* and *put* operations.
RePast runtime scenario (object spaces)

**Runtime scenario**

**Object Spaces**

- Agents access object spaces using `get` and `put` operations.
- The effect of these operations on object spaces is "immediate".
The approach: ACTOR_REPAST over HLA

A RePast model is partitioned into a collection of LPs
The approach: ACTOR_REPAST over HLA

A RePast model is partitioned into a collection of LPs
Each LP hosts a portion of the environment and a subset of agents
The approach: ACTOR_REPAST over HLA

- A RePast model is partitioned into a collection of LPs
- Each LP hosts a portion of the environment and a subset of agents
- The LP maintains the “global view” of the environment
Actors are basic execution units (simulation objects) whose behavior is modeled by a finite (possibly hierarchical) state machine.
Actors are basic execution units (simulation objects) whose behavior is modeled by a finite (possibly hierarchical) state machine.

Actors are reactive objects which encapsulate a data state and communicate to one another by asynchronous message passing.
Actors are basic execution units (simulation objects) whose behavior is modeled by a finite (possibly hierarchical) state machine.

Actors are reactive objects which encapsulate a data state and communicate to one another by asynchronous message passing.

Message processing represents the unit of scheduling and dispatching in a Theatre (logical processor).
Actors are basic execution units (simulation objects) whose behavior is modeled by a finite (possibly hierarchical) state machine.

Actors are reactive objects which encapsulate a data state and communicate to one another by asynchronous message passing.

Message processing represents the unit of scheduling and dispatching in a Theatre (logical processor).

Actors can migrate from a Theatre to another.
Theatre Architecture

**Control Machine**
- Is in charge of scheduling and dispatching of local messages
- Message execution happens in the context of the thread of control of this component

**Network Class Loader**
- Handles dynamic code loading

**Transport Layer**
- Handles distributed concerns like delivering messages addressed to actors located on other theatres and actor migrations

**Local Actor Table**
- Copes with actor-naming and reference issues

---

F. Cicirelli, A. Furfaro, A. Giordano, L. Nigro
Distributed Simulation of RePast Models over HLA/Actors
IEEE/ACM DS-RT 2009
Mapping design rationale

Transformation relies on a software engineering process whose major goal is transparency: minimizing “code intrusions” in the original model.

Mapping issues
Mapping design rationale

- Transformation relies on a *software engineering process* whose major goal is **transparency**: minimizing “code intrusions” in the original model.
- The approach combines Java techniques (meta-data annotations) and Aspect Oriented Programming (as supported by AspectJ).

Mapping issues
Mapping design rationale

- Transformation relies on a *software engineering process* whose major goal is **transparency**: minimizing “code intrusions” in the original model.
- The approach combines Java techniques (meta-data annotations) and Aspect Oriented Programming (as supported by AspectJ).

Mapping issues

- State space representation
Mapping design rationale

- Transformation relies on a *software engineering process* whose major goal is *transparency*: minimizing “code intrusions” in the original model.
- The approach combines Java techniques (meta-data annotations) and Aspect Oriented Programming (as supported by AspectJ).

Mapping issues

- State space representation
- Scheduling system
Mapping design rationale

- Transformation relies on a *software engineering process* whose major goal is *transparency*: minimizing “code intrusions” in the original model.
- The approach combines Java techniques (meta-data annotations) and Aspect Oriented Programming (as supported by AspectJ).

Mapping issues

- State space representation
- Scheduling system
- Conflict resolution on shared variables
Environmental spaces are partitioned into regions over distinct LPs.
Environmental spaces are partitioned into regions over distinct LPs.

Each agent can perceive only the part of the environment which constitutes its *neighbourhood*. 

F. Cicirelli, A. Furfaro, A. Giordano, L. Nigro

Distributed Simulation of RePast Models over HLA/Actors

IEEE/ACM DS-RT 2009
Environmental spaces are partitioned into regions over distinct LPs.

Each agent can perceive only the part of the environment which constitutes its *neighbourhood*.

Region boundaries introduce shared zones: an agent located near a division boundary perceives agents and data located on another LP.
Partitioning and region boundaries

Each LP has

- a *local* border zone whose modifications can be perceived by agents located in other LPs.
Each LP has

- a *local* border zone whose modifications can be perceived by agents located in other LPs
- a *mirror* border which represents the part of the environment handled by other LPs but potentially perceived by local agents
Aspects and Annotations

Aspect-oriented programming

- AOP is a paradigm that increases modularity by allowing the separation of cross-cutting concerns.
- In this context, developing a RePast model and executing it are viewed as two different concerns that should be dealt with separately.
- AspectJ is a state-of-art aspect-oriented extension to the Java programming language, which introduces the notion of pointcuts, i.e. points in the code where various aspects may be involved.
- AspectJ allows to introduce modifications in the definition of classes and interfaces, e.g. adding new instance fields or changing the inheritance relationships.
Aspects and Annotations

Aspect-oriented programming

- AOP is a paradigm that increases modularity by allowing the separation of cross-cutting concerns.
- In this context, developing a RePast model and executing it are viewed as two different concerns that should be dealt with separately.
- AspectJ is a state-of-art aspect-oriented extension to the Java programming language, which introduces the notion of pointcuts, i.e. points in the code where various aspects may be involved.
- AspectJ allows to introduce modifications in the definition of classes and interfaces, e.g. adding new instance fields or changing the inheritance relationships.

Java Meta-data (Annotations)

- Annotations are used to decorate the source code without directly affecting its semantics.
- They are handled by customizable annotation processing tools.
- Special annotations are introduced for automating the generation of suitable aspects from RePast models.
Model and agent classes are respectively annotated with the `@Model` and `@Agent` tags, and respectively become heirs of `ACTOR_MODEL` and `ACTOR_AGENT`.
Scheduling issues and actors

- Model and agent classes are respectively annotated with the `@Model` and `@Agent` tags, and respectively become heirs of `ACTOR_MODEL` and `ACTOR_AGENT`.

- Action objects become messages and get scheduled by the theatre control machine (the RePast controller is bypassed).
Scheduling issues and actors

Model and agent classes are respectively annotated with the @Model and @Agent tags, and respectively become heirs of \texttt{ACTOR\_MODEL} and \texttt{ACTOR\_AGENT}.

- Action objects become messages and get scheduled by the theatre control machine (the RePast controller is bypassed).
- Interactive actions are still handled by the RePast controller.
Method invocations on remote agents are intercepted by aspects and transformed into messages sent to the remote LP.
Interceptors and aspects

Once received such messages are handled by invoking the original methods on the actual destination agents.
Space concerns

Each Theatre LP hosts a special actor (\texttt{ACTOR\_ENV}) which mediates all the interactions between agents and spaces. In particular, it is in charge to handle modifications of shared object spaces.

F. Cicirelli, A. Furfaro, A. Giordano, L. Nigro

Distributed Simulation of RePast Models over HLA/Actors

IEEE/ACM DS-RT 2009
Conflict resolution on shared objects

- Conflict resolution is handled by adapting the conservative simulation algorithm of Theatre (without resorting to native HLA mechanisms).
- The time notion exposed to RTI is a triple: \( \langle \text{virtual-time}, \text{generation}, \text{step} \rangle \)

where:
Conflict resolution on shared objects

- Conflict resolution is handled by adapting the conservative simulation algorithm of Theatre (without resorting to native HLA mechanisms).
- The time notion exposed to RTI is a triple: `<virtual-time, generation, step>`
  where:
  - `virtual-time` is the model’s simulation time.
Conflict resolution on shared objects

- Conflict resolution is handled by adapting the conservative simulation algorithm of Theatre (without resorting to native HLA mechanisms)
- The time notion exposed to RTI is a triple: `<virtual-time, generation, step>` where:
  - `virtual-time` is the model’s simulation time
  - `generation` is a field used to control concurrent accesses to shared objects: conflicting agents can operate only at different generations
Conflict resolution is handled by adapting the conservative simulation algorithm of Theatre (without resorting to native HLA mechanisms).

The time notion exposed to RTI is a triple: \(<\text{virtual-time}, \text{generation}, \text{step}>\)

where:

- \text{virtual-time} is the model’s simulation time
- \text{generation} is a field used to control concurrent accesses to shared objects: conflicting agents can operate only at different generations
- \text{step} is one-bit field used to guarantee that update operations on borders cells are actualized on all LPs before the next generation and to ensure consistency of agent migrations.
Conflict resolution is handled by adapting the conservative simulation algorithm of Theatre (without resorting to native HLA mechanisms).

The time notion exposed to RTI is a triple: `<virtual-time, generation, step>` where:
- `virtual-time` is the model’s simulation time
- `generation` is a field used to control concurrent accesses to shared objects: conflicting agents can operate only at different generations
- `step` is a one-bit field used to guarantee that update operations on borders cells are actualized on all LPs before the next generation and to ensure consistency of agent migrations.

The assignment algorithm produces identical effects on the two borders of two adjacent LPs, and improves the concurrency degree.
Generation assignment

Conflict resolution

- Different generations are assigned to conflicting agents
Generation assignment

Conflict resolution

- Different generations are assigned to conflicting agents
- The algorithm scans the border area from top to bottom and from left to right
Generation assignment

Conflict resolution

- Different generations are assigned to conflicting agents
- The algorithm scans the border area from top to bottom and from left to right
- For each agent a random generation is chosen from a set that does not contain the numbers already assigned to potentially conflicting agents

F. Cicirelli, A. Furfaro, A. Giordano, L. Nigro
Distributed Simulation of RePast Models over HLA/Actors
IEEE/ACM DS-RT 2009
Generation assignment

Conflict resolution

- Two agents are considered potentially conflicting when they are located on different LPs and are less than two cells far each other.
Generation assignment

Conflict resolution

- Two agents are considered potentially conflicting when they are located on different LPs and are less than two cells far each other.
- The same assignment is made by two neighbouring LPs without requiring any interaction by using the current logical time as the seed for the random number generators.

Legend:

- ▲ generation already assigned
- ● generation not yet assigned
- □ potential conflicts with X
- X current agent considered
A TileWorld Model as testbed

Agent’s *mission*: move around to find a tile and then move to fill an hole and so forth until no more tiles exist.

Agent’s mission: move around to find a tile and then move to fill an hole and so forth until no more tiles exist.
Agent’s *mission*: move around to find a tile and then move to fill an hole and so forth until no more tiles exist.

One single tile is enough to fill an hole which, once covered up, disappears along with the tile.
A TileWorld Model as testbed

Agent’s *mission*: move around to find a tile and then move to fill an hole and so forth until no more tiles exist.

One single tile is enough to fill an hole which, once covered up, disappears along with the tile.

The goal is not to compare agent strategies but only to check the achievable simulation performance.
Agent’s *mission*: move around to find a tile and then move to fill an hole and so forth until no more tiles exist

One single tile is enough to fill an hole which, once covered up, disappears along with the tile.

The goal is not to compare agent strategies but only to check the achievable simulation performance.

A scalable version with a huge number of (randomly placed) agents, tiles and holes, and static obstacles, was experimented.
TileWorldAgent model

TileWorldAgent

Distributed Simulation of RePast Models over HLA/Actors
Simulation Experiments (3LP)

- A territory of 825x275 cells split on 3 LPs (275x275 cells for each node)
A territory of 825x275 cells split on 3 LPs (275x275 cells for each node)
Simulation time=150 tu. $r=$#tiles/#agents
A territory of 825x275 cells split on 3 LPs (275x275 cells for each node)

Simulation time = 150 tu. \( r = \frac{\text{#tiles}}{\text{#agents}} \)

Three Pentium IV, 2 GHz, 256MB RAM WinXP platforms with RePastJ 3.0 interconnected by a Gigabit Ethernet switch in the presence of HLA pRTI 1516
SpeedUp

Simulation SpeedUp

![Graph showing the relationship between number of agents and speedup]

- Number of agents
- Speedup

F. Cicirelli, A. Furfaro, A. Giordano, L. Nigro
Distributed Simulation of RePast Models over HLA/Actors
IEEE/ACM DS-RT 2009
Distributed simulation of Repast models is achieved by requiring minimal changes in the original sequential models. The approach is based on:

- state-of-art software engineering concepts, namely Java text annotations and AOP
Distributed simulation of Repast models is achieved by requiring minimal changes in the original sequential models. The approach is based on:

- state-of-art software engineering concepts, namely Java text annotations and AOP
- a lean infrastructure relying on lightweight agents and message passing
Conclusions

Distributed simulation of Repast models is achieved by requiring minimal changes in the original sequential models. The approach is based on:

- state-of-art software engineering concepts, namely Java text annotations and AOP
- a lean infrastructure relying on lightweight agents and message passing
- HLA as standard middleware for distributed execution.
Conclusions

Distributed simulation of Repast models is achieved by requiring minimal changes in the original sequential models. The approach is based on:

- state-of-art software engineering concepts, namely Java text annotations and AOP
- a lean infrastructure relying on lightweight agents and message passing
- HLA as standard middleware for distributed execution.

On-going work is geared at

F. Cicirelli, A. Furfaro, A. Giordano, L. Nigro
Conclusions

Distributed simulation of Repast models is achieved by requiring minimal changes in the original sequential models. The approach is based on:

- state-of-art software engineering concepts, namely Java text annotations and AOP
- a lean infrastructure relying on lightweight agents and message passing
- HLA as standard middleware for distributed execution.

On-going work is geared at

- improving HLA_ACTOR_REPAST and particularly the interplay between Java meta-data and AOP as supported by AspectJ
Conclusions

Distributed simulation of Repast models is achieved by requiring minimal changes in the original sequential models. The approach is based on:

- state-of-art software engineering concepts, namely Java text annotations and AOP
- a lean infrastructure relying on lightweight agents and message passing
- HLA as standard middleware for distributed execution.

On-going work is geared at

- improving HLA_ACTOR_REPAST and particularly the interplay between Java meta-data and AOP as supported by AspectJ
- extending the Tileworld model example so as to test the system in the evaluation of complex agent strategies which combine perception, cognitive and deliberative aspects
Distributed simulation of Repast models is achieved by requiring minimal changes in the original sequential models. The approach is based on:

- state-of-art software engineering concepts, namely Java text annotations and AOP
- a lean infrastructure relying on lightweight agents and message passing
- HLA as standard middleware for distributed execution.

On-going work is geared at

- improving HLA_ACTOR_REPAST and particularly the interplay between Java meta-data and AOP as supported by AspectJ
- extending the Tileworld model example so as to test the system in the evaluation of complex agent strategies which combine perception, cognitive and deliberative aspects
- studying different organizations of the environment component of multi-agent systems with the goal of evaluating effective techniques for distributing (transparently) spatial environments.
Thanks!

Any questions?