Distributed Simulations

Some Experience

DSS Meeting, 22 October 2007

Petr Zelenka
pzeli@kiv.zcu.cz

Outline

- Master's Thesis Review
  - System Architecture
  - Synchronization Mechanism
  - Simulation Process Distribution
  - Using J-Sim
- Possible Further Steps
  - Components
  - Case Study

Master's Thesis Review

Some Basic Facts About the Thesis

Thesis Title:

Distributed Discrete-Time Simulation Using Java™ RMI

Thesis Supervisor:

doc. Ing. Stanislav Racek, CSc.

Thesis Examiner:

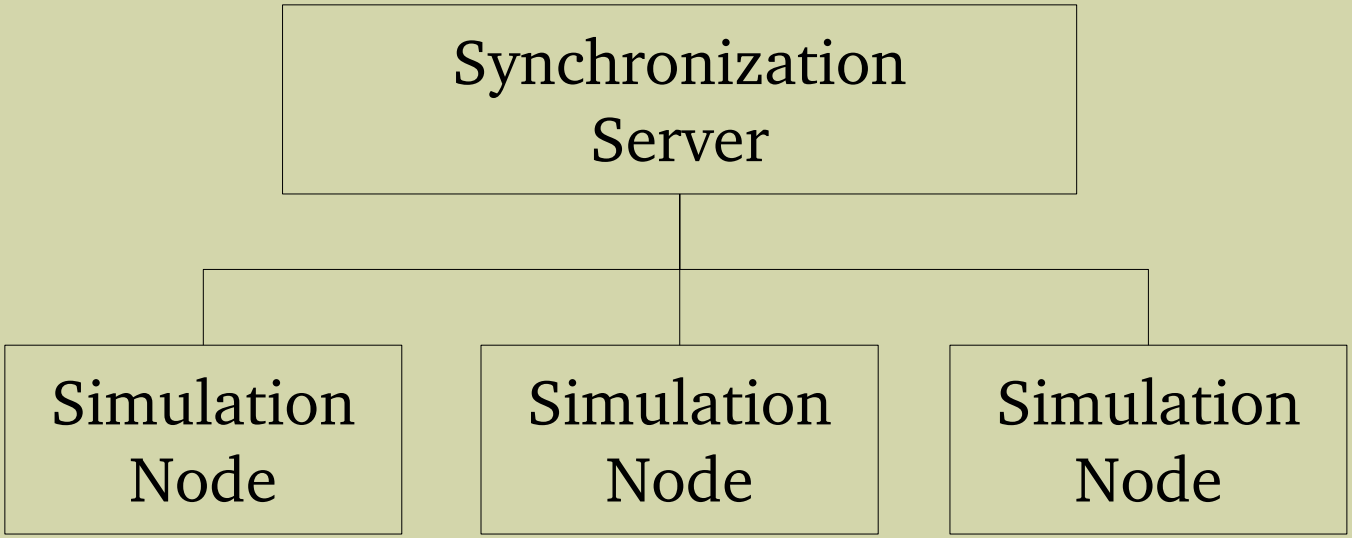
Ing. Tomáš Potužák

Thesis Problem

- To develop a system allowing synchronization of state variables among several simulation processes running in a distributed environment
  - Simulation processes should be written in Java™, possibly using the J-Sim library
  - Internal communication should be implemented using Java™ RMI
- Induced subproblems
  - Synchronization mechanism
  - Simulation process distribution mechanism

System Architecture (1)

- Based on the traditional client-server model
- Principal components
  - Synchronization server (server side)
  - Simulation node (client side)



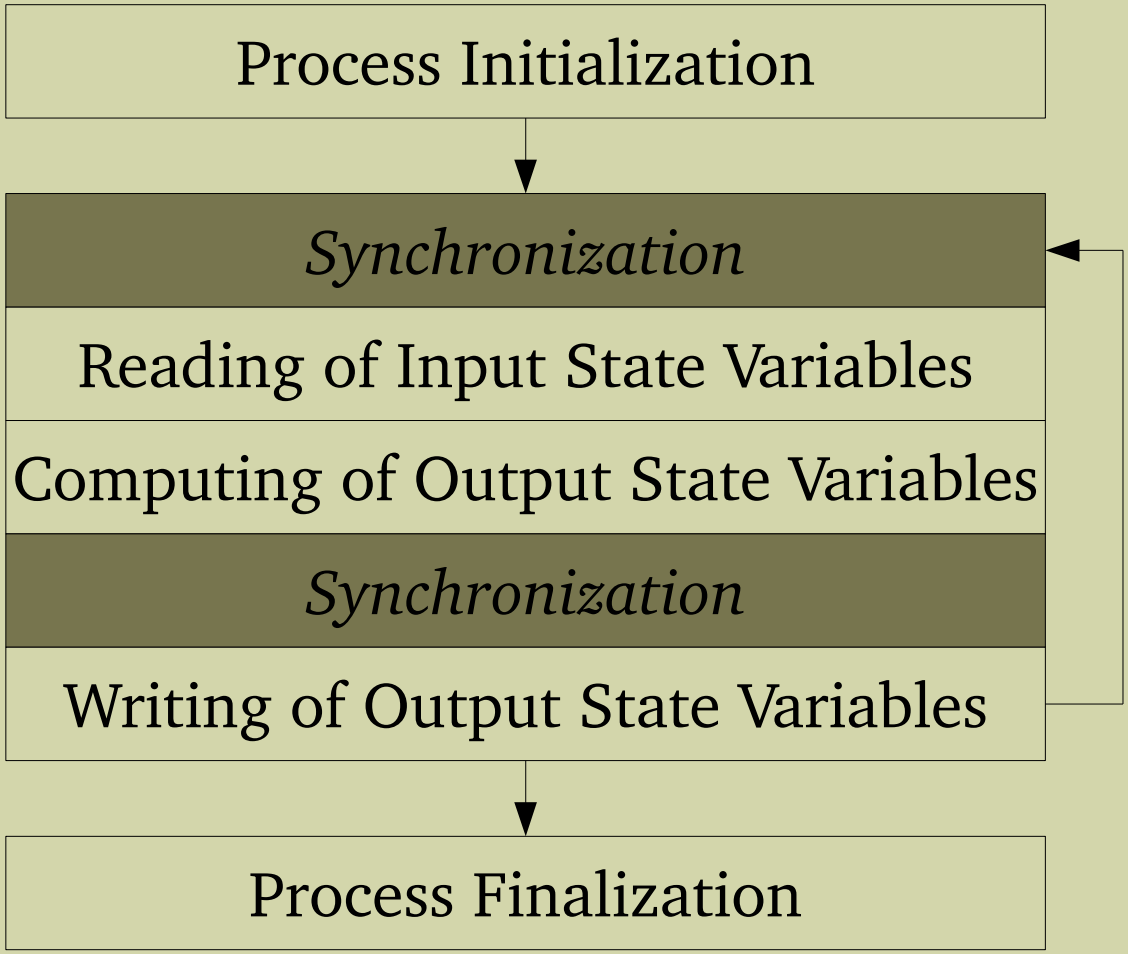
System Architecture (2)

- Simulation node
  - A container for a simulation process to run in
- Synchronization server
  - A repository for the state variables
- Simulation process
  - Is responsible for computing some of the state variables
  - Keeps data objects for input and output state variables and synchronizes them with the other processes

Synchronization Mechanism (1)

- Currently, there is only one synchronization mechanism implemented
  - Conservative synchronization for time-stepped simulations
  - Constant, a priori defined time step size
  - Each simulation process declares its maximum permissible time step size, the minimum of them is selected for the entire simulation
  - Suitable mainly for continuous concurrent systems

Simulation Process Lifecycle



Synchronization Mechanism (2)

- Synchronization over distributed shared memory using two barriers
  - Reading and writing operations on the state variables need to be synchronized using a barrier
- The barriers need to be dimensioned before the simulation begins
  - Ordinary simulation processes plus an optional process for reading and writing state variables for the user interface

Simulation Process Distribution (1)

- Performed through a dispatcher integrated into the synchronization server
  - Reverses the client-server relation at this stage
- Does not use any naming or directory services to discover available nodes, a simple configuration file is used instead
  - The standard RMI registry is not sufficient, an extension by a conifguration file would be neccessary
  - But a standalone configuration file solves this issue as well

Simulation Process Distribution (2)

- Some wrapper code for the simulation node listens at a port for incoming network connections and returns the stub to the simulation node itself
- The dispatcher
  - Parses the configuration file
  - Keeps lists of available and used simulation nodes
  - Registers the simulation processes passed to it
  - Gets the simulation node stubs as neccessary
  - Starts and terminates the simulation

Using J-Sim (1)

- The objects from the J-Sim library could be used inside the simulation process
  - The simulation process needs to be serializable because of its distribution to the simulation node
  - Most objects from the J-Sim library do not satisfy the criterions of serializability
- It would be impossible (or at least too difficult) to use J-Sim for optimistic synchronization mechanisms

Using J-Sim (2)

- Some modifications in the J-Sim library are required
  - JSimHead, JSimLink, JSimProcess
    - Suffices to implement java.io.Serializable
  - JSimSimulation
    - Uses collections for storing events
    - Needs to implement the readObject and writeObject methods or mark the corresponding attributes as transient
    - There are no events at the time of the simulation process distribution, this makes the serialization easier

Lessons Learned

- The simulation process can be (easily) distributed to the simulation nodes from a remote central point
- Using J-Sim in distributed simulations is somewhat complicated even for the easiest scenarios
  - J-Sim was developed for pseudoparallel simulation only
  - Some issues can be solved, some can be not

Possible Further Steps

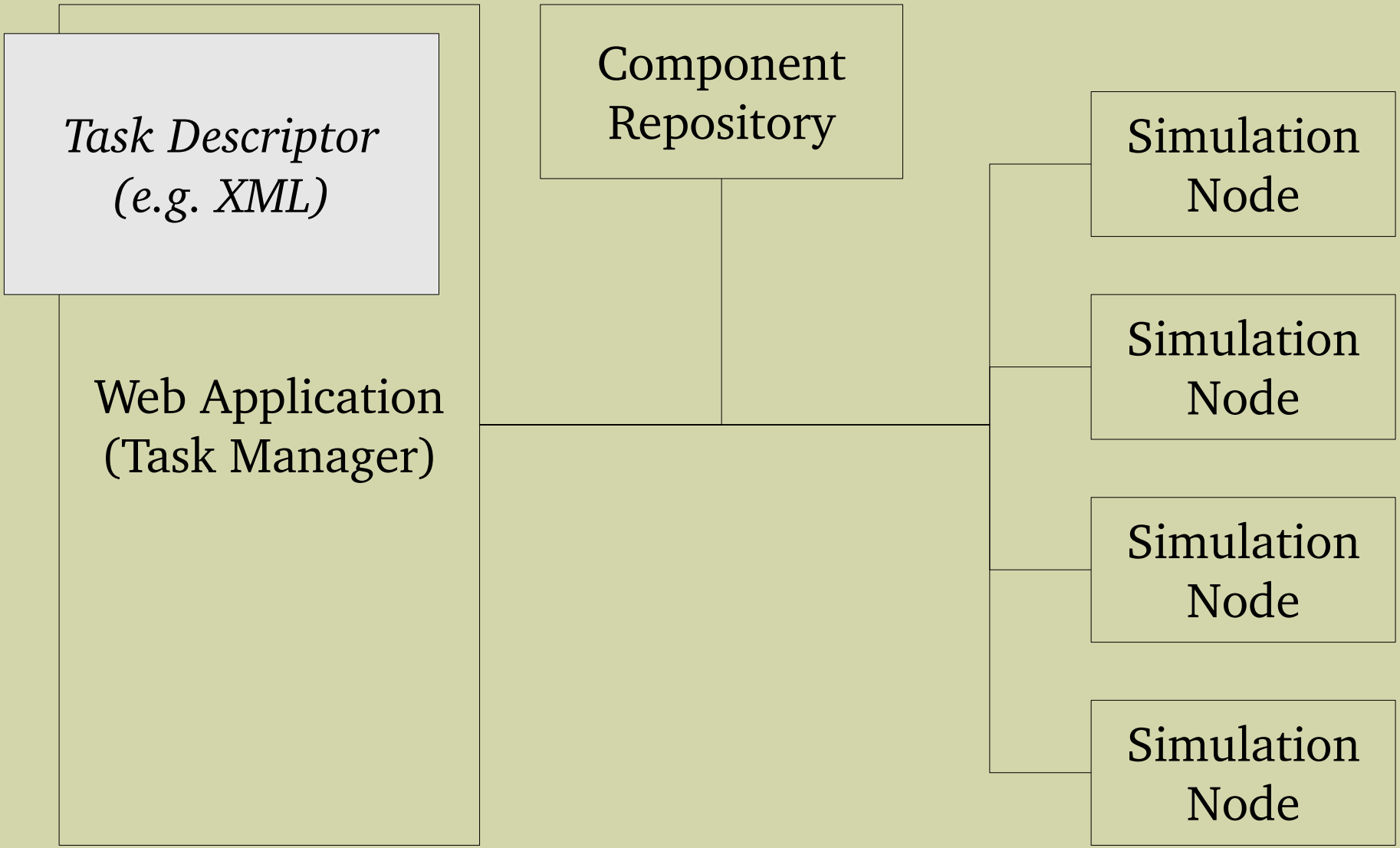
Components

- Software component (definition by Szyperski)

  A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.

- Software component can be deployable at runtime, without any specific knowledge of it at compile time

Case Study


Task Descriptor
(e.g. XML)
Web Application
(Task Manager)
Component
Repository
Simulation
Node
Simulation
Node
Simulation
Node
Simulation
Node

References

- Fujimoto, R.M.
  Parallel and Distributed Simulation Systems.
  John Wiley & Sons, 2000.
- Grosso, W.
  Java™ RMI.
  O'Reily & Associates, 2001.

That's all, folks! :-)

Questions and comments are appreciated