



UNIVERSITY OF AMSTERDAM

UvA-DARE (Digital Academic Repository)

An agent based architecture for constructing Interactive Simulation Systems

Zhao, Z.

Publication date
2004

[Link to publication](#)

Citation for published version (APA):

Zhao, Z. (2004). *An agent based architecture for constructing Interactive Simulation Systems*. [Thesis, fully internal, Universiteit van Amsterdam].

General rights

It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations

If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: <https://uba.uva.nl/en/contact>, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.

List of Figures

1.1	Functional components and the data flow in a simulator. The time stepping routines define the actual behaviour of the simulator, and the control routines define the experiment performed on the simulation.	2
1.2	Computing power increases in the past decade. The figure shows the fastest (N=1) and the slowest (N=500) computer in the list, and the total performance of all computers in the list. The original information is from the website http://www.top500.org	3
1.3	A general data flow diagram of visualisation systems. The procedures in a visualisation pipeline constitute the core of the system. The generation of intuitive primitives for rendering, and the user interaction that controls the execution of pipeline can be performed in different machines.	4
1.4	A basic configuration of ISSs. Solid lines depict the simulation loop, and the dash lines depict the visualisation loop.	6
1.5	Basic components in SPLICE: application processes, each with an agent and a local data store.	9
1.6	Distributed federates and the Runtime infrastructure (RTI).	10
1.7	Summary of available architectures.	19
2.1	Basic architecture of an ISS-Conductor component.	25
2.2	A simple agent kernel.	26
2.3	An Actor and its ComA.	26
2.4	The basic architecture of MA.	26
2.5	A basic paradigm of assembling ISS-Conductor based components.	28
3.1	A logical view of the functional components in an MA.	32
3.2	A partial activity-transition graph of the example. In the description, actions: Start, DoStep and Stop do not have dependencies on data objects, the action InitSimulation requires a <i>Setting</i> object as its input, and the action <i>ExportData</i> has a <i>Result</i> object as its output. In the example, the action <i>DoStep</i> has a transition with a condition guard: $error > Setting.error$, which means the action will only be performed when the error is larger than a given bound.	33
3.3	A simple model of human interaction involved systems.	34
3.4	Capability modelling of the components involved with human interactions. A partial activity-transition graph of Fig. 3.3. The term <i>InState</i> describes the state of user activities.	35

3.5	A PT net based model of data production-consumption relation.	37
3.6	A sample scenario for roles Producer_A, Consumer_A and Consumer_B. More examples will be discussed in the next chapter.	39
3.7	A belief-transition graph for deriving the states of neighbour roles.	41
3.8	The action control between an Actor and a conductor, and its reflection in the world model.	42
3.9	The execution states of the actions.	43
3.10	A scenario fragment and its marking graph are shown on the left side. The right side shows a possible execution sequence of role A and B, when they do not apply any concurrency controls. The dashed arrows indicate the marking changes that are perceived by the peer role; the markings in Italic font are invalid.	44
3.11	Data structure for state synchronisation.	46
4.1	Routing spaces and their four default profiles defined for ComAs. Roles are {A, B, C}, the ComAs at Conductor are {A _c , B _c , C _c } and the ComA at Actor are {A _a , B _a , C _a }. The regions for the Conductors use solid lines; and for Actors use dash lines. XMIN, YMIN, XMAX and YMAX are the boundary of the entire region.	53
4.2	A detailed lifecycle of a ComA.	54
4.3	The snapshot of the GUI of a Conductor.	55
4.4	Generating run-time configuration files.	56
4.5	Two components constructed for benchmarking ISS-Conductor.	57
4.6	Action reasoning and update of shared objects in between run-time roles.	57
4.7	The delays of the local update and reflection and for the remote update of a shared object. The error bars indicate the standard deviation at each step.	58
4.8	Remote update of shared objects and the throughput of pure TCP sockets.	59
4.9	The basic architecture of DASII and the configurations of the experiment. The RTI is executed in fs1, fs2, and nics.	59
4.10	Update delay with different RTI locations. The error bars indicate the standard deviations.	59
4.11	The comparison of the T_R	60
4.12	The remote update of all eight consumers.	61
4.13	Compare the remote update delay of the 8th consumer and 8 times delay of the first consumer.	61
4.14	The remote update of the first Consumer in the federation in different configurations).	61
4.15	Passing messages to four receivers. The error bars indicate the standard deviations.	62
4.16	A scenario of sending data objects and messages between three component instances.	62
4.17	The influence between object distribution and message passing. The error bars show the standard deviation at each time step.	63
4.18	Remote update delay of different number of attributes. The error bars show the standard deviation at each time step.	63

4.19 Benchmark story. It has three nested scenario nets: T2 (scenario A), T3 (scenario B) and T4 (scenario C).	64
4.20 Scenario A (involved roles: <i>Producer_A</i> , <i>Consumer_A</i> and <i>Consumer_B</i>). Using the publish and subscribe mechanism, a data object can be simultaneously consumed by multiple consumers.	65
4.21 Scenario B (involved roles: <i>Producer_A</i> , <i>Producer_B</i> , <i>Consumer_A</i>). <i>SbT7</i> and <i>SbT8</i> are two critical transitions.	65
4.22 Scenario C (involved roles: <i>Producer_A</i> , <i>Consumer_A</i> , <i>Producer_C</i> and <i>Consumer_C</i>).	66
4.23 Topologies of the routing spaces of the involved roles. The roles that are not involved in the scenario set their routing spaces using the <i>away from the others</i> profile.	66
4.24 The total number of events and the time cost for processing an event.	67
4.25 The time cost by <i>Producer_A</i> for each scenario in different execution paradigms. The error bars show the standard deviations.	69
4.26 The total number of state-update messages received by the <i>Producer_A</i> in each scenario.	70
5.1 A scenario of simulation based operation planning.	74
5.2 The basic functionality of <i>Flow_Simulator</i>	77
5.3 The basic functionality of <i>Desktop_VRE</i>	78
5.4 A partial activity-transition graph of the <i>C.Flow_Simulator</i> component. See the definition of component capability in section 3.2.	79
5.5 Performance comparison between <i>ISS-Conductor</i> component and legacy implementation. The measurement shows the time cost for one iteration. The error bars indicate the standard deviations.	79
5.6 A layered vision of the functionality of the <i>Desktop_VRE</i> system.	80
5.7 A partial activity-transition of the <i>C.Desktop_VRE</i> component. The term <i>In-State</i> describes the user activity state.	81
5.8 An activity diagram for <i>Blood_Flow_Studying</i> scenario. The term <i>UserState</i> describes the activity state of a user.	82
5.9 A scenario net of the <i>Blood_Flow_Studying</i> scenario.	83
5.10 A screen snapshot of the <i>Blood_Flow_Studying</i> scenario. The left window shows the interface of the <i>Conductor</i> and the right one is the interface of the <i>C.Desktop_VRE</i> component. The flow boundary is a tube with $32 \times 32 \times 64$ lattices. The image in the window shows the velocity vectors of the calculated flow field.	84
5.11 Remote update of shared objects when the simulation is executed on multiple processes. The error bars indicate the standard deviations of 100 measurements.	85
5.12 Asynchronous data transmission between the <i>Surgeon_A</i> and <i>Blood_Simulator</i> component instances.	86
5.13 A comparison of the time interval between two invocations of <i>Compute</i> actions.	86
5.14 Basic time costs for updating a <i>Flow_Data</i> object.	88
5.15 The idle time between invoking <i>Refresh_Flow_Data</i> actions.	89

5.16	The total time for simulation to do 80 iterations computing.	89
5.17	Multiple users explore the <i>Blood Flow</i> object. The term <i>UserOpinion</i> describes the opinion choice selected by a user. When both surgeons accept the simulation results, or one of them does not accept, the simulation scenario ends. The high level scenario decides whether to make a new scenario or to finish the entire story.	94
5.18	User's opinions and the chatting interface.	95
5.19	The delay for remotely updating <i>Blood Flow</i> object with one, two and four <i>Surgeons</i> . The error bars indicate the standard deviations. The size of data object is 2M bytes.	95
5.20	A scenario template for collaboratively searching optimal bypasses. In the template, <i>SURGEON ROLE</i> and <i>SIMULATOR ROLE</i> are two general role names, which have to be replaced with concrete role names at run time. . . .	97
5.21	Switch shared classes between different instances of a scenario template. . .	98
6.1	A graphical representation of the experiment requirement for a bypass-validation experiment.	107
6.2	Different types of ontologies.	109
6.3	Developing Ontologies using Protégé.	109
6.4	The basic architecture of an agent.	111
6.5	A snapshot of the component management subsystem.	112
6.6	A snapshot of the experiment planning subsystem.	112
6.7	Searching different number of activities from the component collection. . . .	113