JiST:
Java in Simulation Time

for

Scalable Simulation of Mobile Ad hoc Networks

Rimon Barr
barr@cs.cornell.edu
Wireless Network Laboratory
Advisor: Prof. Z. J. Haas

MURI Demo
26 August 2003

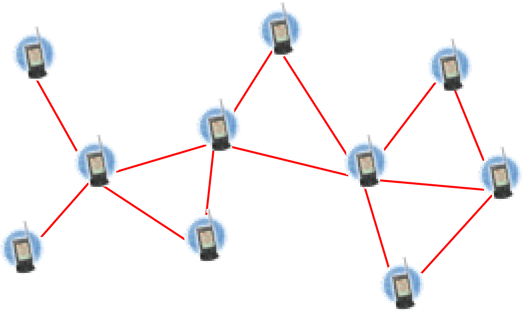
http://www.cs.cornell.edu/barr/repository/jist/

the world today...

- ***Transparent*** **Parallel and** ***Optimistic*** **Execution of Discrete Event** ***Simulations*** **of MANETs in** ***Java***

- **discrete event simulations are useful and needed**
- **but, most published ad hoc network simulations**
  - **lack network *size*** **~250 nodes; or**
  - **compromise *detail*** **packet level; or**
  - **curtail *duration*** **few minutes; or**
  - **are of sparse *density*** **tens of nodes / km^2; or**
  - **etc...**

  - **i.e. limited simulation scalability**



the world today... in perspective

- A university *campus*
  - Cornell students ~ 30,000
  - Wireless devices per student average ~1
  - Main campus < 4 km^2.

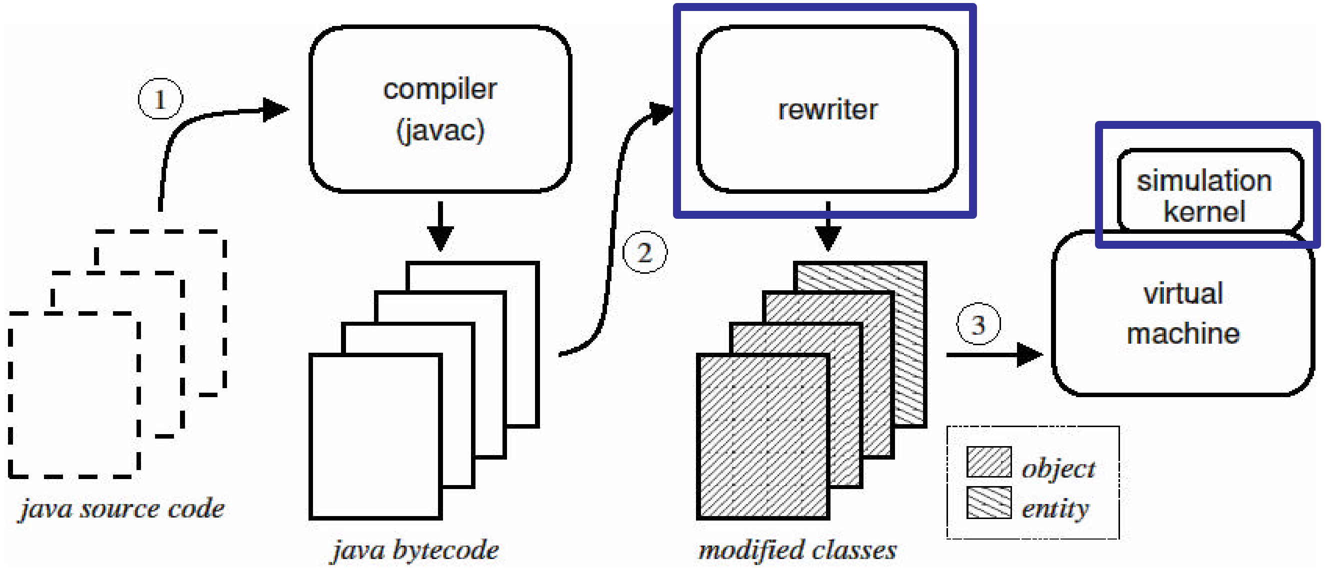
- The United States *military*
  - Troops deployed in Iraq 100-150,000 (in clusters)
  - Wireless devices per soldier ???
  - Territory 400,000km^2

- And, predictions of
  - smaller devices, better radios and chips
  - smart dust, wearable/disposable/ubiquitous computing

Simulation scalability is important.

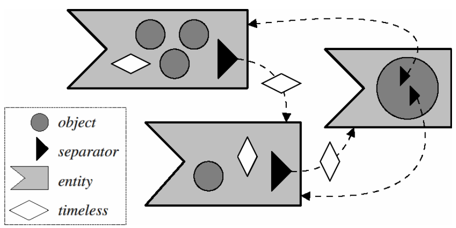
introduction to jist

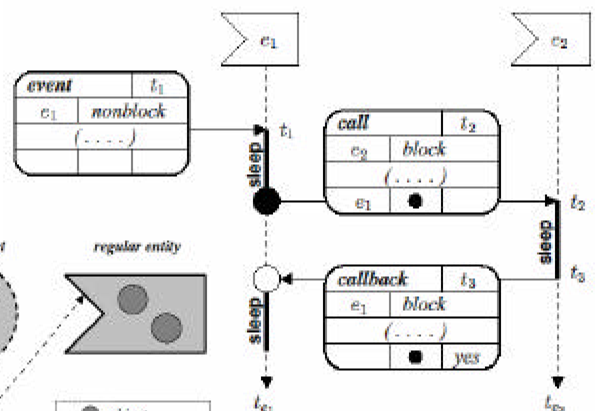
- JiST – Java in Simulation Time
  - *extends* object model and execution semantics
    - simulations written in plain Java
  - … to run discrete event simulations *efficiently*
    - reduces serialization and context-switching overhead
    - allows parallel and speculative simulation execution
  - merges modern language and simulation semantics

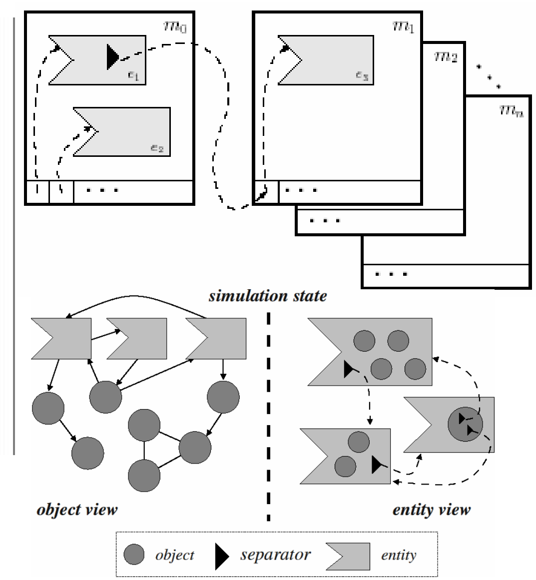


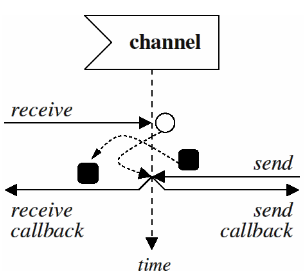
jist functionality

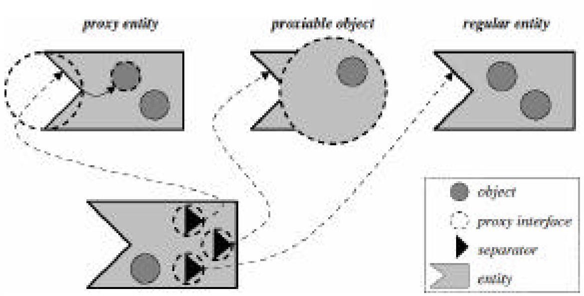
- **entities**: extend object model with simulation time components
- **simulation time invocation**: event-based invocation
- **timeless objects**: pass-by-reference to avoid copy
- **proxy entities**: interface-based entity creation
- **continuations**: call and callback, blocking methods
- **concurrency**: channel, threads, monitors, locks...
- **distribution**: separators track entities across machines
- **scripting**: embed engines for Java, Python, Tcl, etc...











a basic example

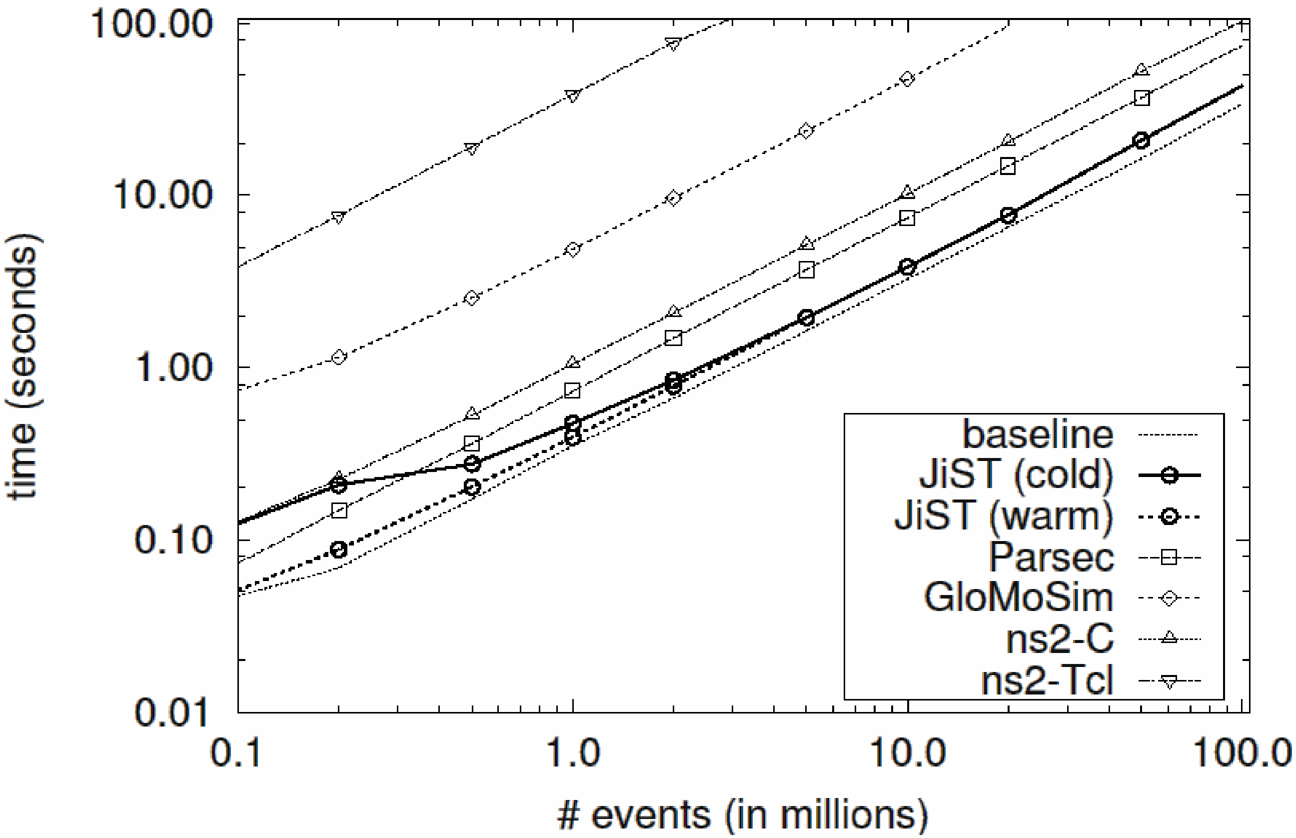
- **the "hello world" of event simulations**

```
class HelloWorld implements JistAPI.Entity
{
  public void hello()
  {
    JistAPI.sleep(1);
    hello();
    System.out.println("hello world, " +
       "time=" + JistAPI.getTime() );
  }
}
```

- **demo!**

Java	JiST
Stack overflow @hello	hello world, time=1 hello world, time=2 hello world, time=3 etc.

performance: event throughput

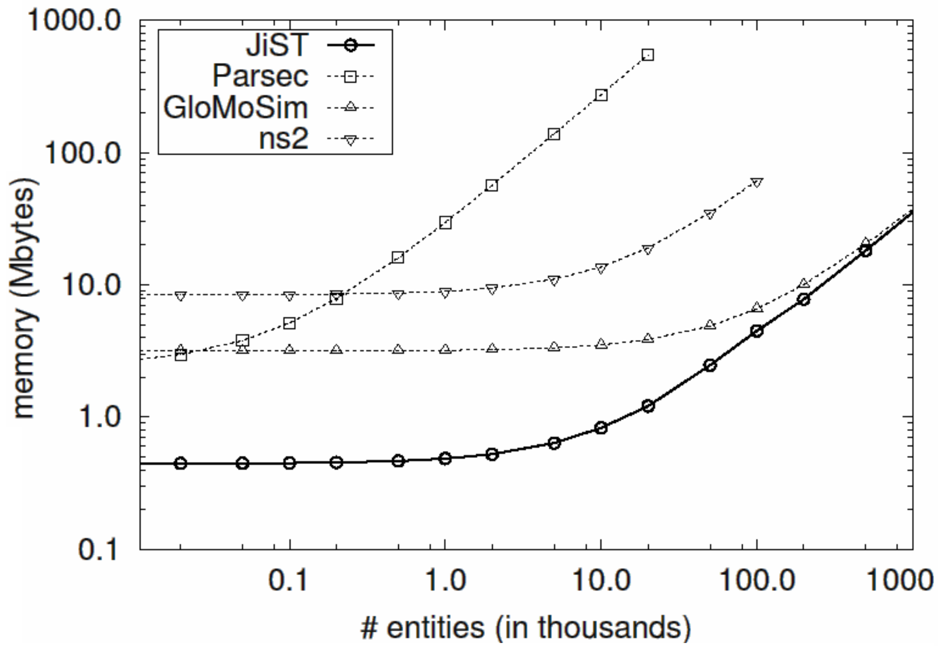


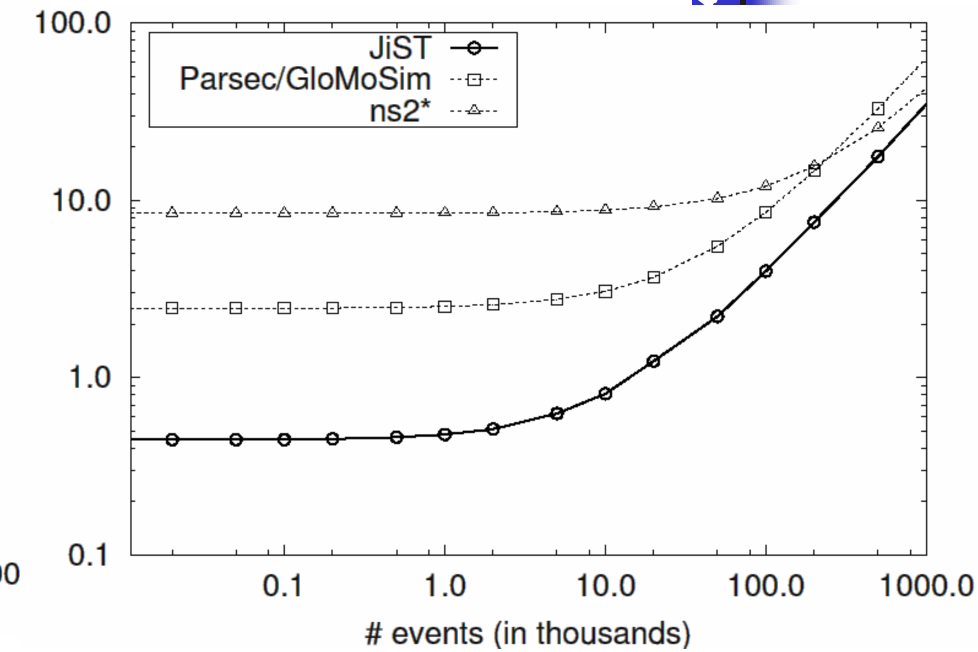
# events	JiST	GloMoSim	Ratio
10^5	0.044s	0.435s	10%
10^6	0.262s	2.938s	9%
10^7	2.301s	28.04s	8%
10^8	22.48s	278.4s	8%

serial throughput increase of 12x

5×10^6 events	time (sec)	vs. baseline	vs. JiST
baseline	1.640	1.0x	0.8x
JiST	**1.957**	**1.2x**	**1.0x**
Parsec	3.705	2.3x	1.9x
ns2-C	5.151	3.1x	2.6x
GloMoSim	23.720	14.5x	12.1x
ns2-Tcl	160.514	97.9x	82.0x

performance: memory overhead





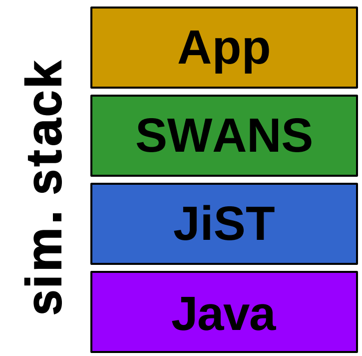
memory	entity	event	10K nodes sim.
JiST	**36 B**	**36 B**	**21 MB**
GloMoSim	36 B	64 B	35 MB
ns2	544 B	36 B*	72 MB*
Parsec	28536 B	64 B	2885 MB

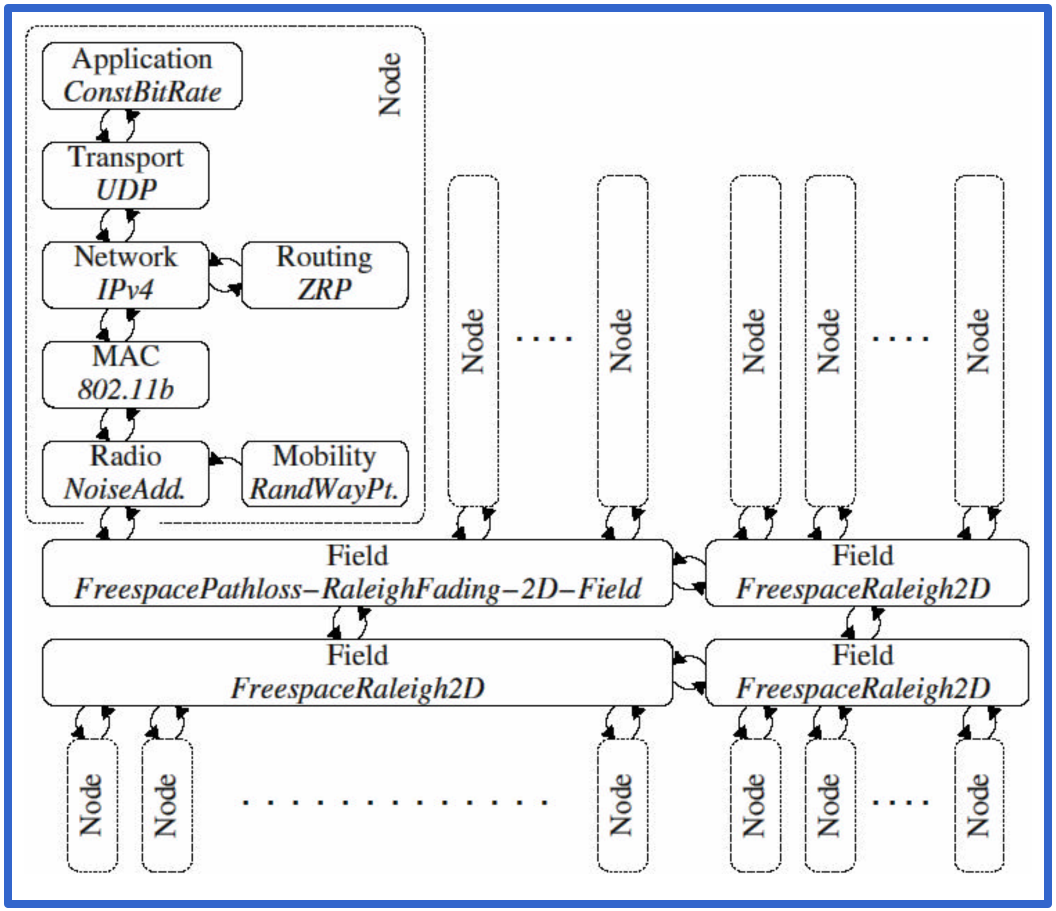
	Memory	Limit
JiST	36 bytes	> 10^6 entities
Parsec	28536 bytes	~ 10^4 entities

JiST scales to more entities per process

SWANS

- **Scalable Wireless Ad hoc Network Simulator**
  - **runs standard Java network applications**
  - **allows vertical *and* horizontal aggregation**





	files	classes	lines
JiST	26	65	9278
SWANS	52	115	12871
Other	16	26	2042
	94	**206**	**24191**

- larger than JiST code-base
- simpler than GloMoSim and ns2 implementations
- less than 3 months

performance: SWANS

- simulation configuration
  - **field** 5x5km^2; free-space path loss; no fading
  - **radio** additive noise; standard power, gain, etc.
  - **link** 802.11b
  - **network** IPv4
  - **transport** UDP
  - **mobility** random waypoint: v=2-5, p=10
  - **application** heartbeat neighbor discovery
- ran on:
  - PIII 1.1GHz laptop
  - 384 MB RAM
  - Sun JDK 1.4.2
- memory consumption:
  - 1.2KB per simulated node!
  - demo!

	nodes		
	1,000	10,000	100,000
ns2	✓	✗	✗
GloMoSim	✓	✓	✗
SWANS	✓	✓	✓

JiST:
Java in Simulation Time

for

Scalable Simulation of Mobile Ad hoc Networks

THANKS!

existing alternatives

ns2 **is the *gold standard***
- C++ with Tcl bindings, $O(n^2)$
- used extensively by community
- written for TCP simulation
- modified for ad hoc networks
- processor and memory intensive
- sequential; max. ~500 nodes

PDNS **– parallel distributed ns2**
- event loop uses RTI-KIT
- needs fast inter-connect
- distribute memory, ~1000 nodes

OpNet **– popular commercial option**
- good modeling capabilities
- poor scalability

custom-made **simulators**
- fast, specialized computation
- lack sophisticated execution and also *credibility*

GloMoSim
- implemented in Parsec, a custom C-like language
- entities are memory intensive
- requires "node aggregation," which imposes conservative parallelism, loses Parsec benefits
- shown ~10,000 nodes on NUMA machine (SPARC 1000, est. $300k)

SWAN
- implemented atop the parallel, distributed DaSSF framework
- similar to GloMoSim

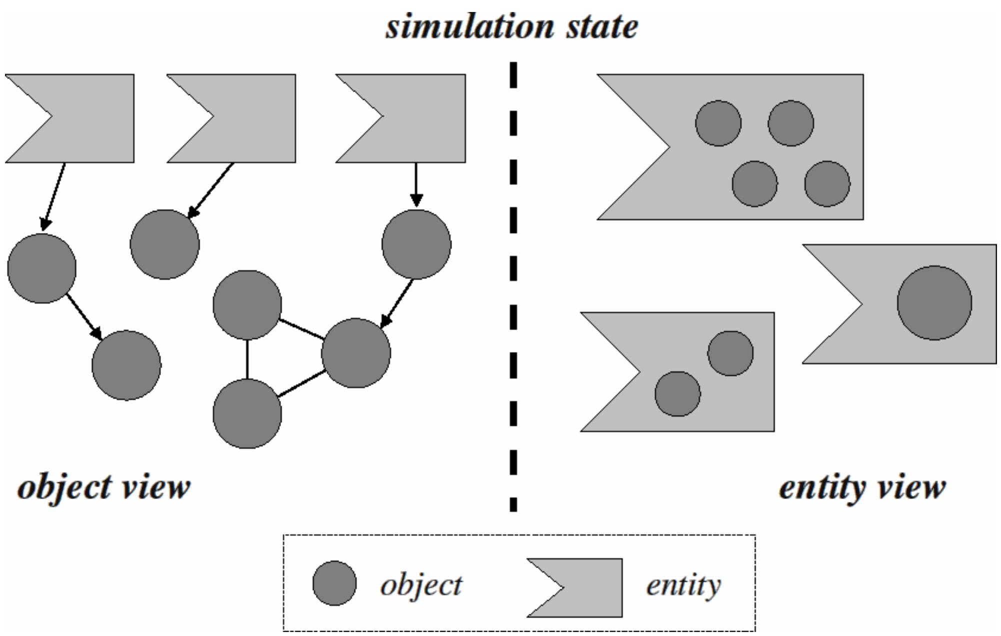
Simulation approaches
- languages (e.g. Parsec, Simula)
- libraries (e.g. Yansl, Compose)
- systems (e.g. TWOS, Warped)

simulation time

- **program time**
  - **progress of program *independent* of time**
- **real time**
  - **progress of program is *dependent* on time**
- **simulation time**
  - **progress of time is *dependent* on program progress**
    - **instructions take zero (simulation) time**
    - **time explicitly advanced by the program, `sleep`**
  - **simulation event loop embedded in virtual machine**

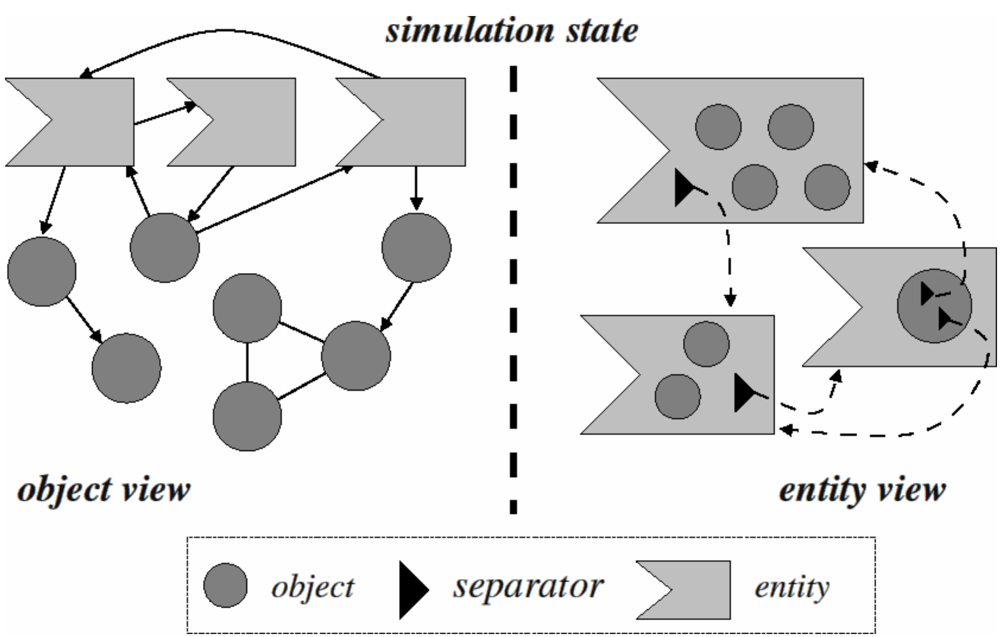
extended object model

- program state contained in objects
- objects contained in entities
  - each entity runs at its own simulation *time*
  - as with objects, entities do not share state
  - think of an entity as a simulation component



extended execution semantics

- entity references replaced with separators
  - event channels; act as state-time boundary
- entity methods are an event interface
  - simulation time invocation
  - non-blocking; invoked at caller entity time; no continuation



benefits of the jist approach

- more than just scalability
- application-oriented benefits
  - type safety   source-target statically checked
  - event types   not required (implicit)
  - event structures   not required (implicit)
  - debugging   dispatch location and state available
- language-oriented benefits
  - garbage collection memory savings, cleaner code
  - reflection   script-based configuration of simulations
  - safety   fine granularity of isolation
  - Java   standard language, compiler, runtime
- system-oriented benefits
  - IPC   no context switch; no serialization
  - Java kernel   cross-layer optimization
  - robustness   no memory leaks, no crashes
  - rewriting   no source-code access required
  - concurrency   supports parallel and speculative execution
  - distribution   provides a single system image abstraction
- hardware-oriented benefits
  - cost   COTS hardware, clusters (NOW)
  - portability   runs on everything