Future Directions in Simulation Modeling

C. Dennis Pegden

Outline

- A half century of progress.
- Where do we need to go from here?
- How do we get there?

Simulation: A Compelling Technology

- See the future
- Visualize dynamic processes
- Understand the impact of change
- Experiment without risk
- Make mistakes early – and in the model
- Improve performance

The Application Gap

- Simulation is widely accepted as a valuable tool for predicting the performance of complex systems.
- Simulation is applied in a small fraction of the cases where it can bring significant value.

Challenges

> "I know we should be using simulation however we don't have the time and resources to allocate to the project."

- Models are time consuming and expensive to build.
- A simulation project requires significant skills in model building, experimentation, and analysis.
- If we want to close the application gap we need to make significant improvements in the model building process to support the fast-paced decision making environment of the future.

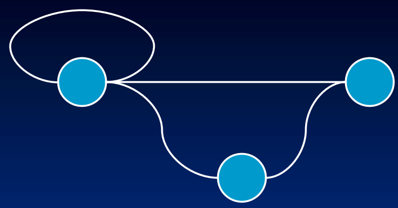
Application Trends

- The world is going flat – competition from everywhere – rapid pace of change – need answers quickly.
- A revolution in computing and communication is driving rapid changes in system design.
- Large integrated systems from suppliers to manufacturing to customers.

Model Building

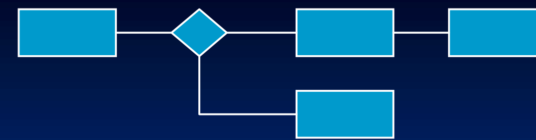
- The process of mapping the real world to a model that executes and changes state over time.
- This mapping from real world to model is based on a world view:
  - Event
  - Process
  - Object (Agent)
  - Continuous (System Dynamics)
- The world view provides a framework for defining the components of the system in sufficient detail to allow the model to execute and simulate the system.
- The framework includes the system state, and mechanisms for changing that state.
- We seek a simple and natural model view that is also flexible and efficient.

Event View



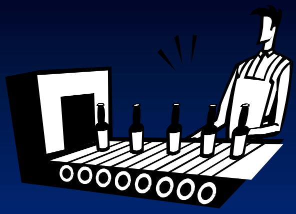
- The most elemental view of a discrete system.
- Models the points in time (events) when the state of the system may change.
- Event logic defines the state changes that occur at each event.
- Time advances from event to event.
- Efficient and flexible – highly abstract.
- Used extensively during the 60's/70's.

Process View



- Models the flow of entities (transactions) through a series of process steps.
- Discrete state changes happen automatically as steps are executed (steps trigger event sequences).
- Process steps can take place over time.
- As flexible and efficient as event modeling – less abstract.
- Used extensively for current day models.

Object (Facility) View



- Models the physical objects in the system.
- Objects combine data and functionality into self-contained units.
- Objects serve as a model of an abstract "actor" that can perform work, report on and change its state, and "communicate" with other objects.
- Object constructs form the basis of modern programming languages.
- Objects provide a natural method for describing a system.

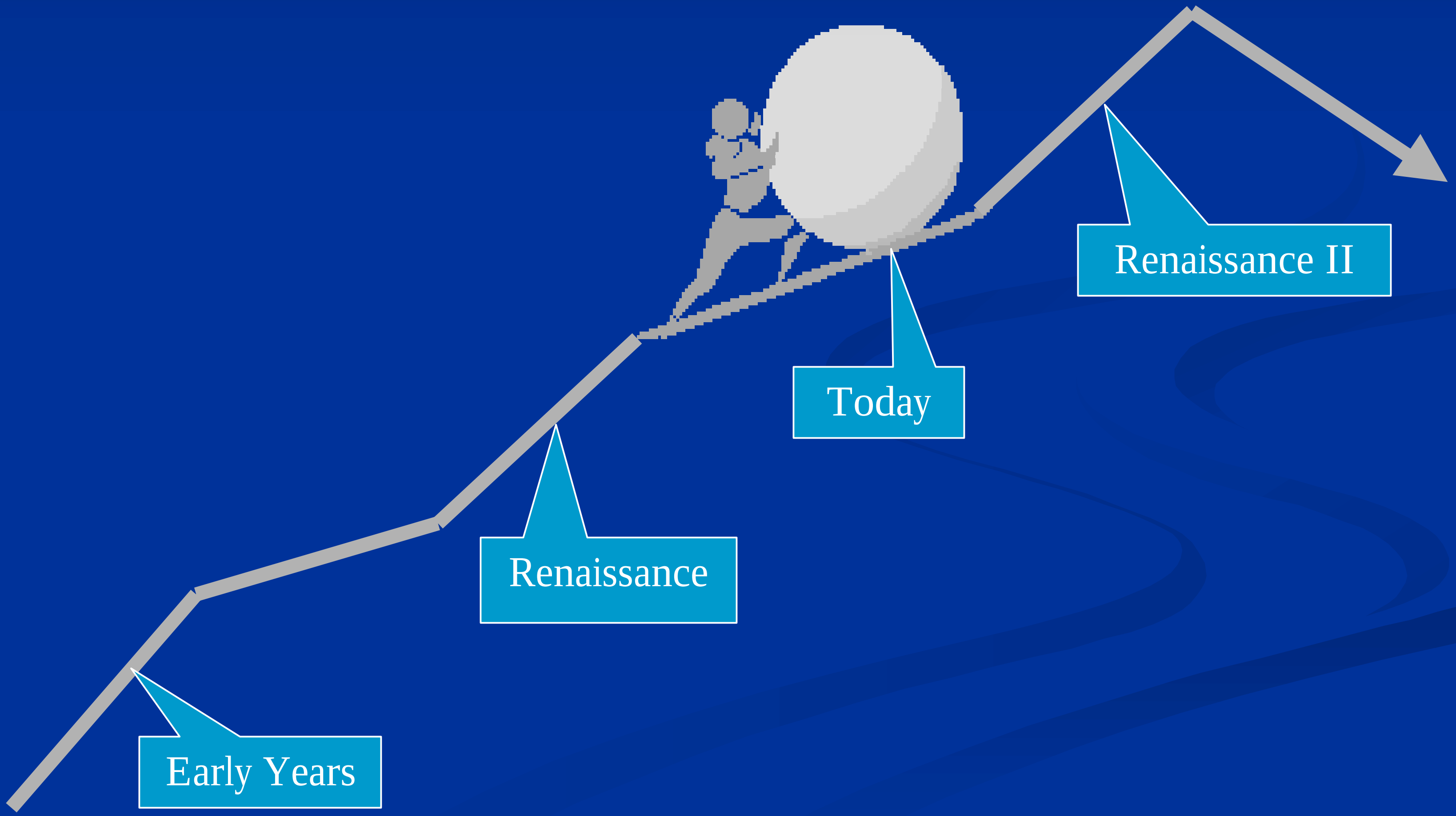
Agent-Based View

- A special case of Object View.
- Macro system behavior emerges as a result of the interaction of a large number of active objects called Agents.
- Agents are typically autonomous, may interact with each other, and are goal directed.
- An alternative approach to aggregated System Dynamics models.

Continuous View

- State of the system changes continuously over time (not just at events).
- Can be used to model continuous systems (e.g. entity movement, fluid flows) or aggregated models of discrete systems (e.g. markets, supply chains, populations).
- Systems of differential equations – key modeling components are feedback loops.

The Path of Progress



The Early Years (60's)

- Birth of modeling concepts
  - Event Modeling (Simscript)
  - Process Modeling (GPSS)
  - Object Modeling (Simula)
  - Systems Dynamics (Dynamo)
- Low application success rate
  - Event programming / Inefficient process modeling
  - No debug tools
  - Tabular outputs / no analysis tools
  - Slow batch computers

The Renaissance (80's)

- An explosion of advances in modeling, animation, and analysis.
- PC based simulation tools.
- Shift from event to flexible and efficient process modeling.
- Graphical model building – advanced GUI.
- 2D (3D) Animation.
- Hierarchical modeling.

The Key to Progress

- The paradigm shift from event to process
  - Efficient next event processing logic
  - Flexible process modeling constructs
  - Graphical model building - improved GUIs
- 2D (3D) Animation
  - Brings the model to life
  - Verification/validation
  - Communication from shop floor to top floor

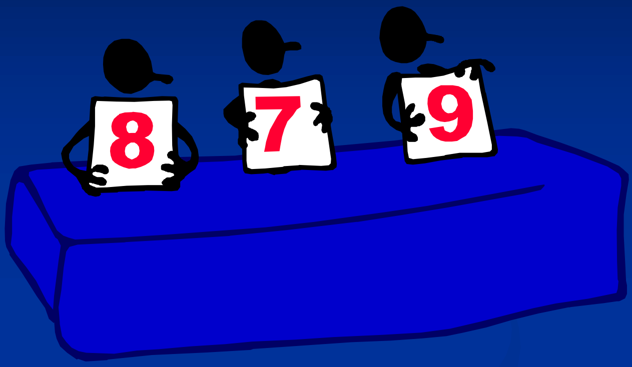
Post Renaissance (90's, 00's)

- Wider acceptance of simulation.
- Tools have become feature rich – but the fundamental modeling paradigm has not changed.
- The application growth created by the first Renaissance has stagnated.
- The important applications are becoming bigger and more complicated.

Looking Ahead: Renaissance II

- The goal is a fundamental shift in ease of use that will expand the application of simulation.
- In Renaissance I the shift was created by moving from an event view to a process view and adding 2D animation.
- In Renaissance II the shift will come by moving from a process view to a 3D animated facility (object) view.
- Success will require innovative ideas in next generation tools.
- What are the challenges and solutions?

Measuring Success



- Practitioners are the judges
- Can a new/existing user
  - Quickly learn the tool?
  - Model the system of interest?
  - Get meaningful results in a timely fashion?
  - Make better decisions with the tool?
- We succeed if we
  - Increase the number of practitioners
  - Increase the number and size of applications
  - Improve the success rate

Benefits of the Facility (Object) View

- The Facility View is a very natural way to describe a system.
- Objects correspond directly to the facility – they support a one-step model build and 3D animation.
- A single object definition can be instantiated (not copied) multiple times – all sharing a common definition.
- Objects can be multifaceted.

The Next Generation Tool

- Unified framework with object, process, event, and continuous modeling.
- 3D (2D) animated objects/models.
- Graphical model/object build.
- Domain neutral framework – application focused objects.
- Lightweight objects – fast execution.
- Distributed application using Web services.

Rethinking the O-O Paradigm

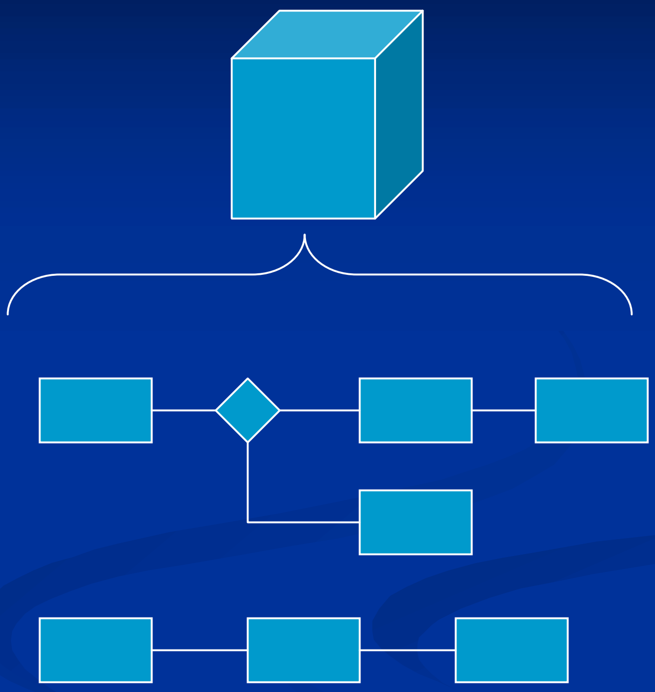
- The O-O paradigm was invented in the simulation world (Simula) and then adopted by the programming world.
- Most modern languages (C++, Java, C#, ….) are based on the O-O paradigm.
  - Abstraction: focus on the essential.
  - Encapsulation: only the object can change its state.
  - Polymorphism: messages trigger object-specific actions.
  - Inheritance (is-a): specialized objects derived from existing objects.
  - Composition (has-a): new objects built by combining existing objects.
- We can code simulation objects in an O-O programming language – however this does not achieve our objective of making simulation dramatically easier to use.
- A better alternative is to build a graphical simulation modeling system around the O-O concepts.
- How do we do this?

A Model is an Object

- Make the terms model and object interchangeable. Model builders are object builders.
- A model can be instantiated into other models. A model can be a single machine or an entire factory or supply chain. A model can have multiple instances.
- A model has a 3D(2D) state-driven animated view.
- A model instance has properties that specify input parameters for the model.
- A model can be built from processes, sub-classed from another model, or created by combining existing models.

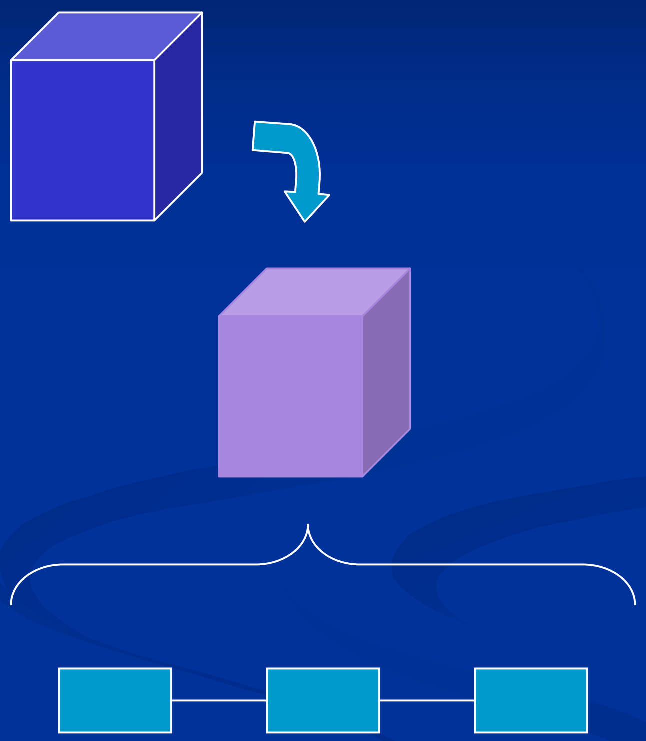
Base Objects

- Built from processes.
- Processes are analogous to methods in O-O programming – but span across time.
- Events trigger processes that are executed by tokens. These processes change the state of the parent object.
- Events include time, change, threshold, and logic events.



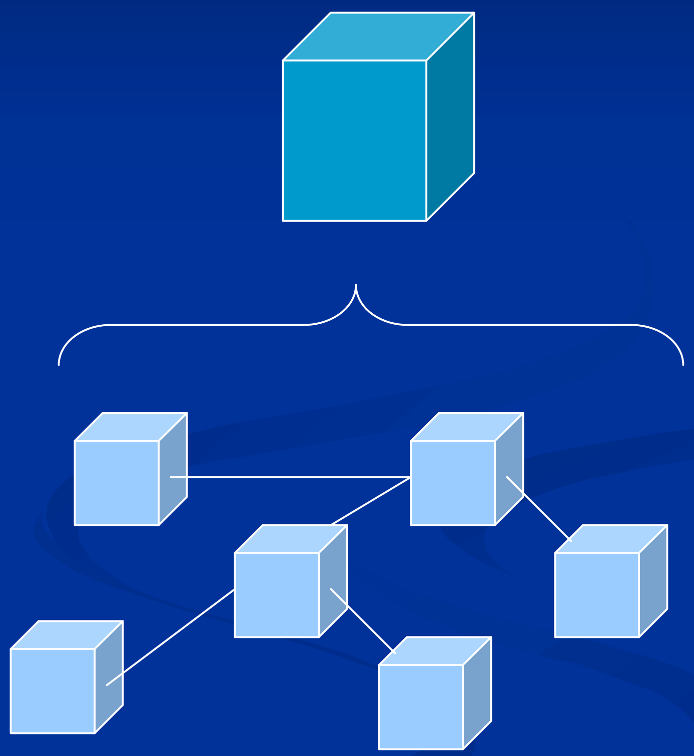
Derived Objects

- Built by inheriting and modifying/extending the behavior of a parent object.
- Parent processes may be overridden.
- New processes may be added.



Hierarchical Objects

- Built by combining instances of existing objects into a facility model (object hierarchy).
- Entity arrivals to an object spawn new entities that move through a facility model of the object.



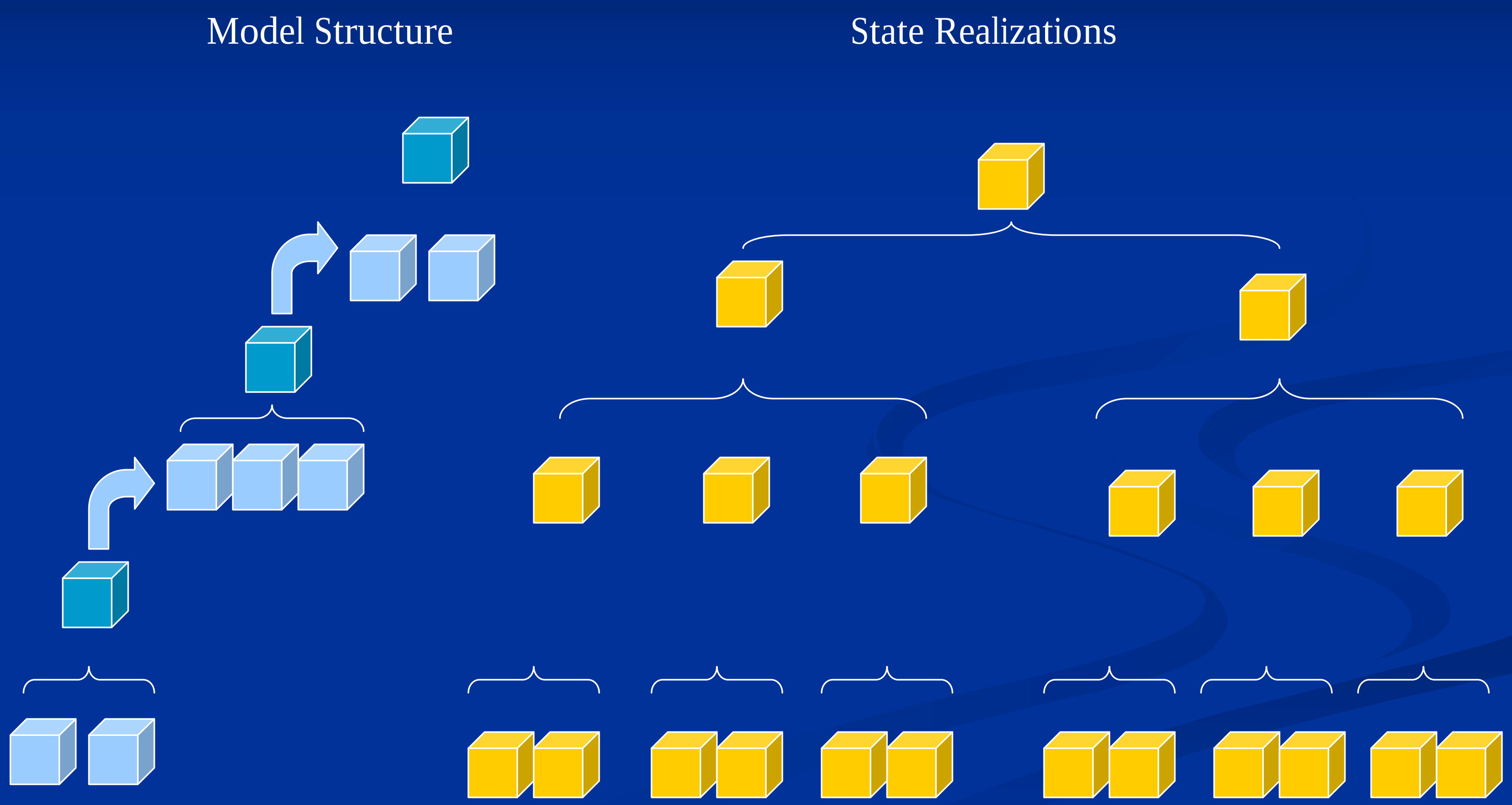
Some Key Design Challenges

- Making models objects
- Lightweight objects
- Complex movements
- Flexible object interactions
- Shareable objects
- Fast execution

3-Tier Object Structure

- 3-Tiered Objects:
  - Definition:
  - Instance:
  - Realization:
- Instances hold properties, realizations hold states.
- A definition may have multiple instances, each instance has properties that specify parameters of the object.
- An instance may have multiple realizations, each realization holds the state of the object.
- Object realizations are only created and used during execution.
- This 3-tier structure facilitates light weight objects, “Change and Go” execution, and parallel execution of replications.

Hierarchical Example



Object Movements: Object Classes

- Objects
  - A fixed location in the model
  - Entities arrive and depart from the object at transfer stations
  - Objects have intelligence defined by processes that are triggered by events
- Entities
  - Objects that move across networks of links from object to object.
  - An entity (object) may have intelligence
- Links
  - Objects that provide a pathway for entity movements
  - Start and end at intersections/stations
  - A link (object) may have intelligence
- Transporters
  - Entities that pick up, carry, and drop off other entities
  - A transporter (entity) may have intelligence and move across links

Object Interaction

- Objects must co-exist and interact with each other.
  - Transferring an entity between objects.
  - Messaging an object to perform an action.
  - Detecting other objects.
- Polymorphic – object specific responses to messages.

Shareable Objects

- Object Fidelity
  - Conventional wisdom – purpose built model – designed to answer specific questions.
  - With reusable objects – the model purpose is not known in advance.
  - Objects must be able to simulate at multiple levels of fidelity.
- Encapsulation
  - Objects do not know details about other objects in the system.
  - Objects do not know the details of entities that arrive to the object.

Execution Speed

- Computers are getting faster – but problems are getting larger.
- Managed code (Java, .NET) executes slower than conventional code.
- Fast execution is necessary to enhance the analysis of results.
- Parallel execution of replications is highly desirable.
- Implementation details are critical.
  - Time and threshold event management
  - Process steps
  - Continuous state variables

Looking Beyond Renaissance II

- Unified Analysis Tool - Multifaceted Objects
  - Layout
  - Kinematics
  - Simulation
  - Emulation
- Vendor Supplied Objects

Summary

- 60's and 80's were periods of great progress – the past decade has been one of refinements.
- Future growth depends on making simulation dramatically easier to use.
- The next renaissance will be built around a unified facility – process – event view.
- Success is in the details of design and implementation.
- Two key insights: model == object, 3-tier object
- Practitioners judge our work.