

# Complexities of Simulating a Hybrid Agent-Landscape Model Using Multi-Formalism Composability

Gary R. Mayer

[Gary.Mayer@asu.edu](mailto:Gary.Mayer@asu.edu)

Hessam S. Sarjoughian

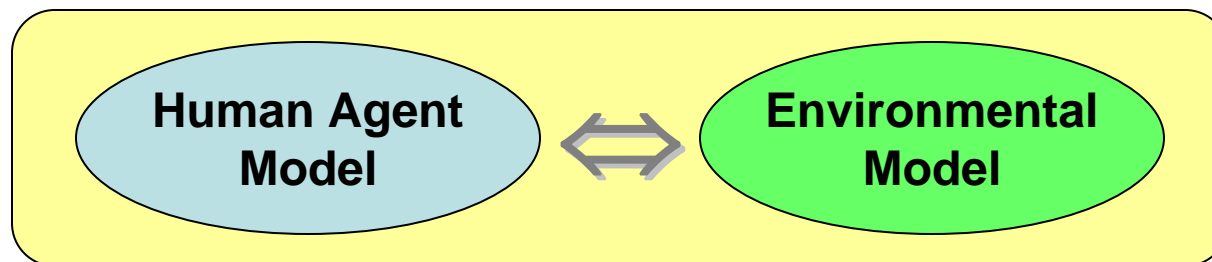
[Sarjougian@asu.edu](mailto:Sarjougian@asu.edu)

Arizona Center for Integrative Modeling & Simulation  
Computer Science & Engineering Department  
School of Computing and Informatics  
Arizona State University, Tempe, Arizona

Presented at the Agent-Directed Simulation (ADS'07) Conference  
A Part of the 2007 Spring Simulation Conference (SpringSim'07)  
Norfolk, VA, USA  
25 – 29 March 2007

# Hybrid Agent-Landscape Models

- “Hybrid”: both the human and environmental sub-system models are well developed
  - Possibly separable and independently executable
- Better understand the effects of humans interacting *with* their environment
  - Bi-directional interaction
- Choose appropriate formalism to describe each sub-system
- Complexity of interactions confounds V&V



# Using Multiple Models

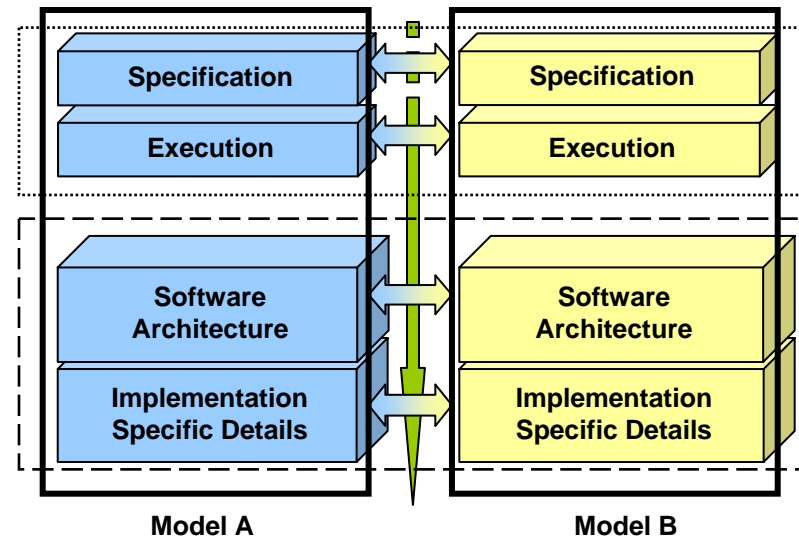
- Independence between models provides
  - benefits for **reorganization and reuse** and
  - opportunities to **study the interaction** between the models
- Multi-Formalism provides
  - **easier translation of requirements** from domain to system model
  - the ability to choose a modeling formalism that is expressive enough to **fully represent the system**
  - model dynamics under simulation that are **more logical to modeler**
  - **easier V&V** as interactions and model-domain connection is more clear
  - a challenge in figuring out **how to put the two models together**
    - Our approach, “*composition*”

# Model Correctness

- *Can a simulator correctly generate model output given a specific state and input?*
- Modeling the composition of heterogeneous components is more difficult to prove correct
  - Correct w.r.t formalism and realization
  - Interactions between models
  - Emergent property of agents makes it necessary to test system behavior at system level

# Composability Problem

- Composition implies more than just a software architecture (data and control) solution
  - Models are composed in both formalism and realization aspects.
  - Compose formalism, then realization
- Each model must be correct and valid
- How are two disparate models composed while ensuring that the entire model remains syntactically and semantically correct?
- All too often, modelers consider ways to make it work, not how it should work.



# Formalism

- A model's *formalism* is a description of a dynamic system that is independent of domain and implementation
  - Specification
    - Mathematical description of the system
  - (Abstract) Execution
    - How the models are executed
  - Generalized for a class of models
    - e.g. discrete-time
- Composition Consideration Examples
  - Timed with timeless models
  - Discrete-event with continuous models

# Realization

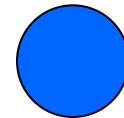
- A model's realization is the implementation of the model formalism into less abstract structures
  - Software Architecture
    - Conceptual code structure and dynamics
  - Implementation Specific Details
    - Data-related implementation issues
  - Incorporates domain-specific considerations
- Composition Consideration Examples
  - Conversion of a string of data into an object message
  - 100m resolution versus 10m resolution

# Multi-Modeling Approaches

## Mono-Formalism

single formalism; multiple models

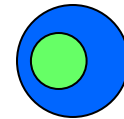
*(everything must be expressed within that one formalism)*



## Super-Formalism

one formalism can describe two or more other formalisms

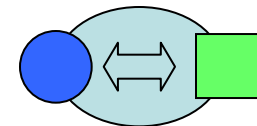
*(same-family restriction; forces uniform syntax and semantics)*



## Meta-Formalism

two different formalisms are mapped to a third formalism

*(restricted expressiveness to ensure proper mapping)*



## Poly-Formalism

two different formalisms interact via a third formalism

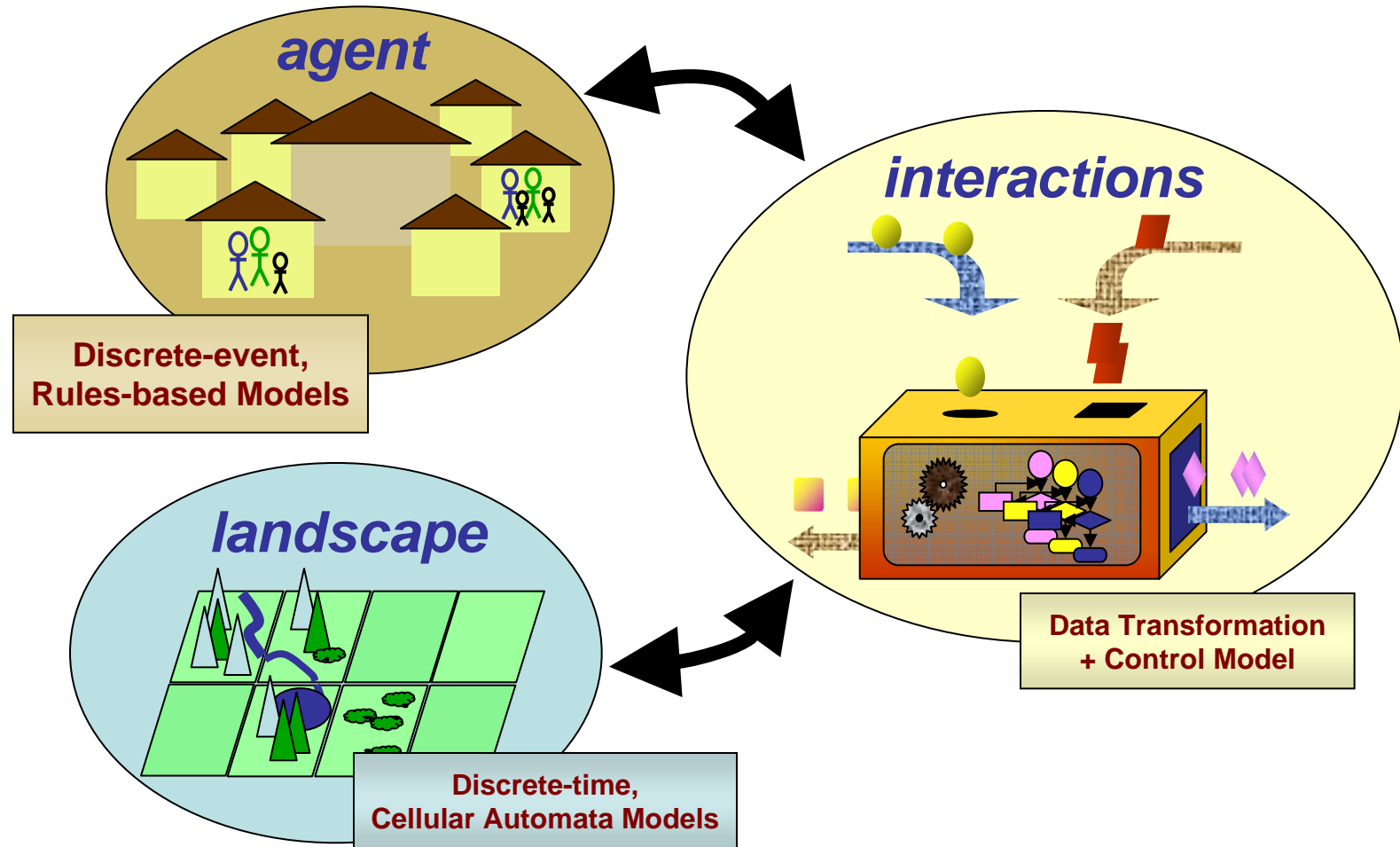
*(unique composition for each set of composed formalisms)*



# Which Approach?

- **IT DEPENDS!**
- One of our project goals is to assist in the development of a laboratory environment for social science research of human and environment interactions.
  - Devise an agent model
  - Develop an interface between the agent and a landscape model.
  - Would like ability to study modeled systems separately, study bi-directional interaction between models, make significant revisions to one with minimal impact to the other, and enable researchers (who may be non-programmers) to use the resulting software framework as a testbed.
- We will focus on implementing the poly-formalism approach to compose our two models.

# Poly-Formalism Agent-Landscape Model Architecture



# Discrete-Event, Rules-Based Agent

$$M_{DERBA} = (X_M, Y_M, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta)$$

(a parallel Discrete Event System Specification formalism) where

$X_M = \{(p, v) \mid p \in IPorts, v \in X_p\}$  is the set of input ports and values

$Y_M = \{(p, v) \mid p \in OPorts, v \in Y_p\}$  is the set of output ports and values

$S$  is the set of sequential states

$\delta_{ext}: Q \times X_M^b \rightarrow S$  is the external state transition function

$\delta_{int}: S \rightarrow S$  is the internal state transition function

$\delta_{con}: Q \times X_M^b \rightarrow S$  is the confluent transition function

$\lambda: S \rightarrow Y^b$  is the output function

$ta: S \rightarrow \mathfrak{R}_{0,\infty}^+$  is the time advance function

$Q = \{(s, e) \mid s \in S, 0 \leq e \leq ta(s)\}$  is the set of total states

\* Note that the superscript 'b' refers to a "bag" or set with possible multiple occurrences of its elements

# Discrete-Time, Cellular Automaton

$$M_{DTCA} = \langle X_N, D, \{M_{ij}\}, h_N \rangle$$

(a multicomponent Discrete Time System Specification structure) where

$X_N$  is an arbitrary set of input values, could be  $\emptyset$

$h_N = 1$ , time base is a set of integers

$D = \{(i, j) | i \in I, j \in I\}$  is the index set, and

for each  $(i, j)$ , the component  $M_{i,j}$  is specified as

$M_{i,j} = \langle Q_{i,j}, Y_{i,j}, I_{i,j}, \delta_{i,j}, \lambda_{i,j} \rangle$ , where

$$Q_{i,j} = \{0,1\}$$

$Y_{i,j}$  is an arbitrary set of outputs of  $(i,j)$  or  $\emptyset$  if no outputs

$I_{i,j} \subseteq D$  is the set of influencers of  $(i,j)$ , defined by its neighbors

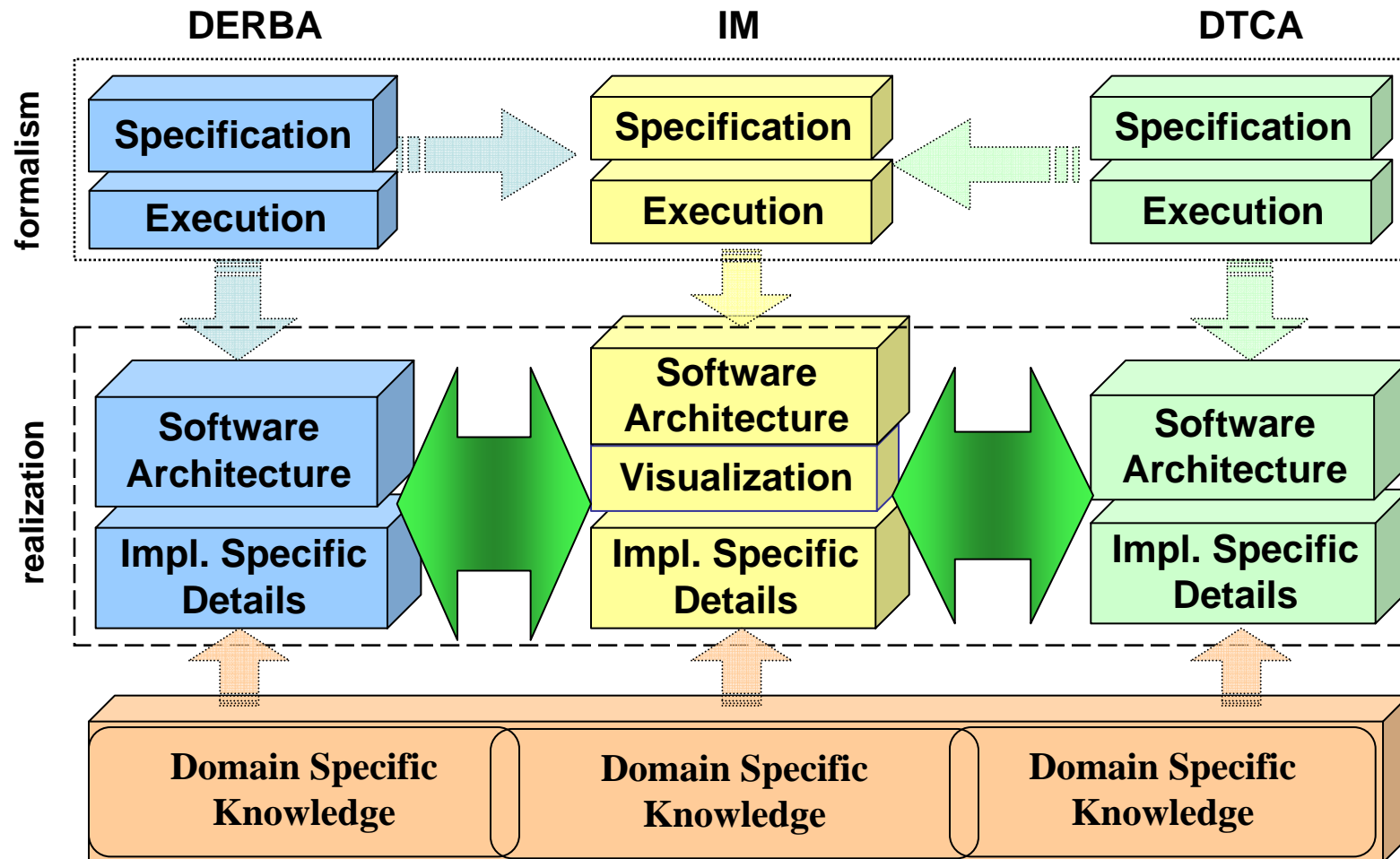
$$\delta_{i,j} : x_{k,l} \in I_{j,k}, Q_{k,l} \rightarrow Q_{i,j}$$

$$\lambda_{i,j} : x_{k,l} \in I_{j,k}, Q_{k,l} \times X_N \rightarrow Y_{i,j}, \emptyset \text{ if } Y_{i,j} = \emptyset$$

# On-Going R&D

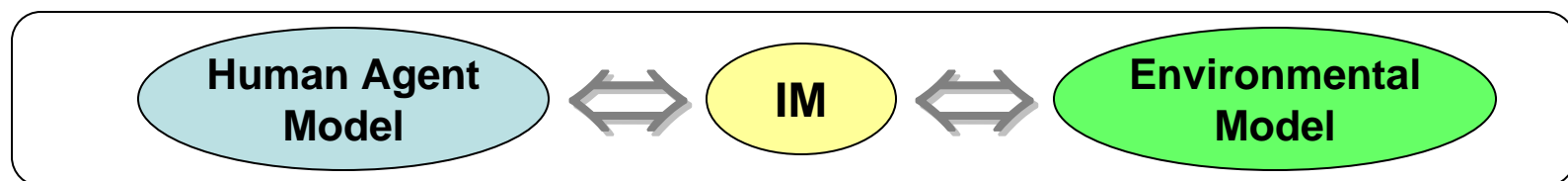
- Humans
  - Discrete-event, rules-based agent (DERBA) models
  - Implemented in DEVJSJAVA
- Landscape
  - Discrete-time cellular automata (DTCA) models
  - Implemented in Geographic Resources Analysis Support System (GRASS) (a Geographic Information System (GIS))
    - Independent set of C modules originally developed for command line interface; no formal interface across multiple modules.
- Interaction Model (IM)
  - Discrete-event model
  - Implemented in DEVJSJAVA

# Proposed Hybrid Model Architecture

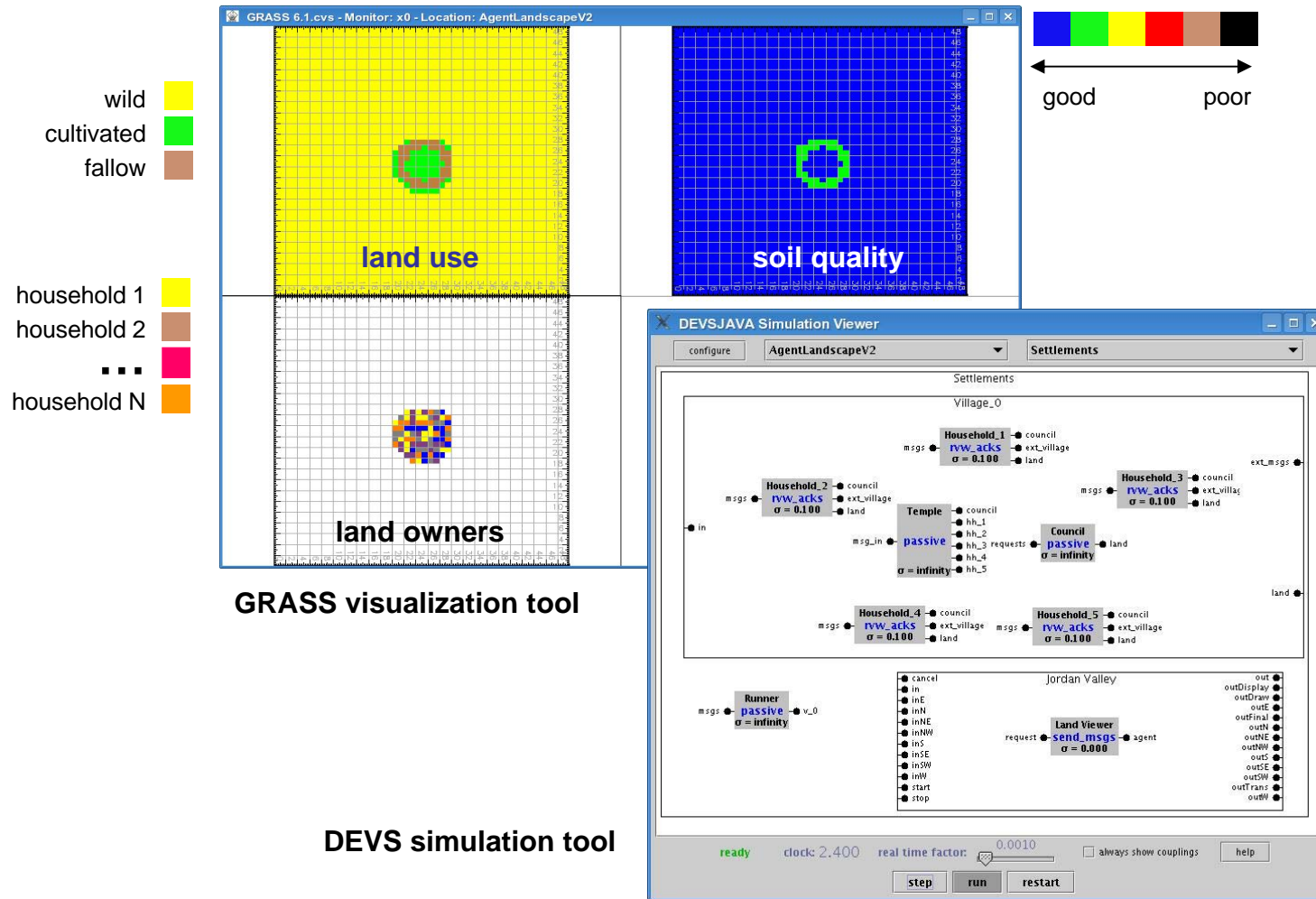


# Interaction Model

- Provides the composition between formalism and realization of the composed models
  - Comprised of translation methods to convert between the two composed models
  - ensure that the models formalism is met and executed correctly (e.g. handle unexpected events in discrete-time models)
  - Manage data and control issues (e.g. converting multi-variable objects into int, float, string components)
  - Synchronous versus asynchronous execution
    - Handle potential deadlock situations (*between* composed models)
  - Scale and resolution disparities
    - May maintain an internal map to assist translation



# Research Progression



# Poly-Formalism Approach

- Moves many of the details of the domain and formalism of both composed models into the interaction model.
  - Removes knowledge of composed models from the other
- Ability to interact lies completely within interaction model.

# Challenge: Specification Interaction

- Mathematically, what does it mean for the output of one model to be injected as input into the other?
- Our Example: Discrete-event interaction model
  - Respond to input from either model at unknown times
- Issue: Discrete-event IM injects input into a discrete-time model that is not synchronized with a regularly scheduled input segment.
- Proposed Solution: Composed discrete-time models may not contain a function that explicitly anticipates the time delta between function executions.

# Challenge: Execution Control

- How closely tied is the model to its execution? Is there a formal engine (simulator)? Do the models explicitly incorporate time?
- Our Example: Model execution coordination.
  - DEVS is timed; GRASS is untimed.
- Issue: Control and execution of all three models.
- Proposed Solution: IM controls model execution synchronization.
  - Centralized execution engine using IM simulator.
  - Use DEVS model to provide timing to GRASS.

# Challenge: Software Architecture

- Must account for disparate software languages and constructs, and hardware resource needs.
- Our Example: We plan to use the Java implementation of the DEVS formalism – DEVSJAVA. GRASS is a set of C modules that are best accessed using scripting languages (Bash, Python, etc.)
- Issue: How to make DEVSJAVA communicate with GRASS modules
- Proposed Solution: We have had success dynamically creating scripts to access GRASS modules and data and then using the Java `Runtime.exec()` command to execute those scripts and capture return data in output and error buffers. Place this functionality within the IM.
  - Parse data from buffers and wrap in DEVS message object

# Challenge: Visualization

- Provide unified, synchronized data visualization with data elements from both models.
  - Can not be assumed that only data researchers will want to see is data being transmitted through IM.
- Our Example: Data visualization of large landscape dataset combined with a few hundred mobile agents.
- Issue: How to architect system to allow full access to each models data without destroying modularity and independence of each model.
- Proposed Solution: Implement a visualization element within the IM architecture.
  - Model-View-Controller design pattern
  - Centralized execution control also allows IM to ensure models are synchronized before data extraction and display.
  - Data formatting, mapping, and aggregation
  - Allows central initialization of all models

# Challenge:

## Implementation Specific Details

- Scalability from an execution (performance) perspective.
- Our Example: A landscape model may employ millions of data elements, each at a 100ha scale. The agent may have only a few hundred and operate at a 10ha scale.
- Issue: Agent may not need all data or may operate on a combination of data. Further, if an agent moves partially within a 100ha space, what does that mean to the models?
- Proposed Solution: The IM handles data formatting, mapping, and aggregation. It may also maintain an internal map representation to handle scale disparities.

# Challenge: Usability

- How to develop a complex framework while still maintaining ease of use for modelers with minimal or no formal coding skills.
- Our Example: DEVSJAVA uses object-oriented constructs and requires coding skills in order to implement models.
- Issue: As a part of the lab environment, it will likely be necessary to modify the IM and agent behaviors. Given the Java environment, how can this be done to improve usability while still maintaining correctness of the models?
- Proposed Solution: Break single agent model into separate models and components. By keeping the behavior of each static while allowing modification of variables and model organization, different agent behaviors may be realized.
  - Initialization scripts define structure and parameters.

# IM Benefits

- Much greater visibility into the interaction between the two models.
- Three levels of generalization
  - Implementation
    - Compose any GRASS discrete-time, cellular automata model with any discrete-event, rules-based agent written in DEVSJAVA
  - Formalism
    - Compose any system containing the same class of models (discrete-event, rules-based agents and discrete-time, cellular automata)
      - Implementation may require interaction model changes
  - Visualization
    - Maintain separation between data model, visualization and control

# Summary

- Composability solutions should be examined from both the formalism and realization aspects.
- Poly-formalism composability provides model independence but may be costly to develop due to creation of third model.
  - Retains rigorous adherence to formalism and formalism realization to facilitate correctness of interaction between models.
  - Each implementation is dependant upon the two models being composed and, while issues will likely surface, all issues to date appear to have reasonable approaches toward a solution.