Entity Based System Dynamics

Larry Yeager, larry@ventanasystems.com

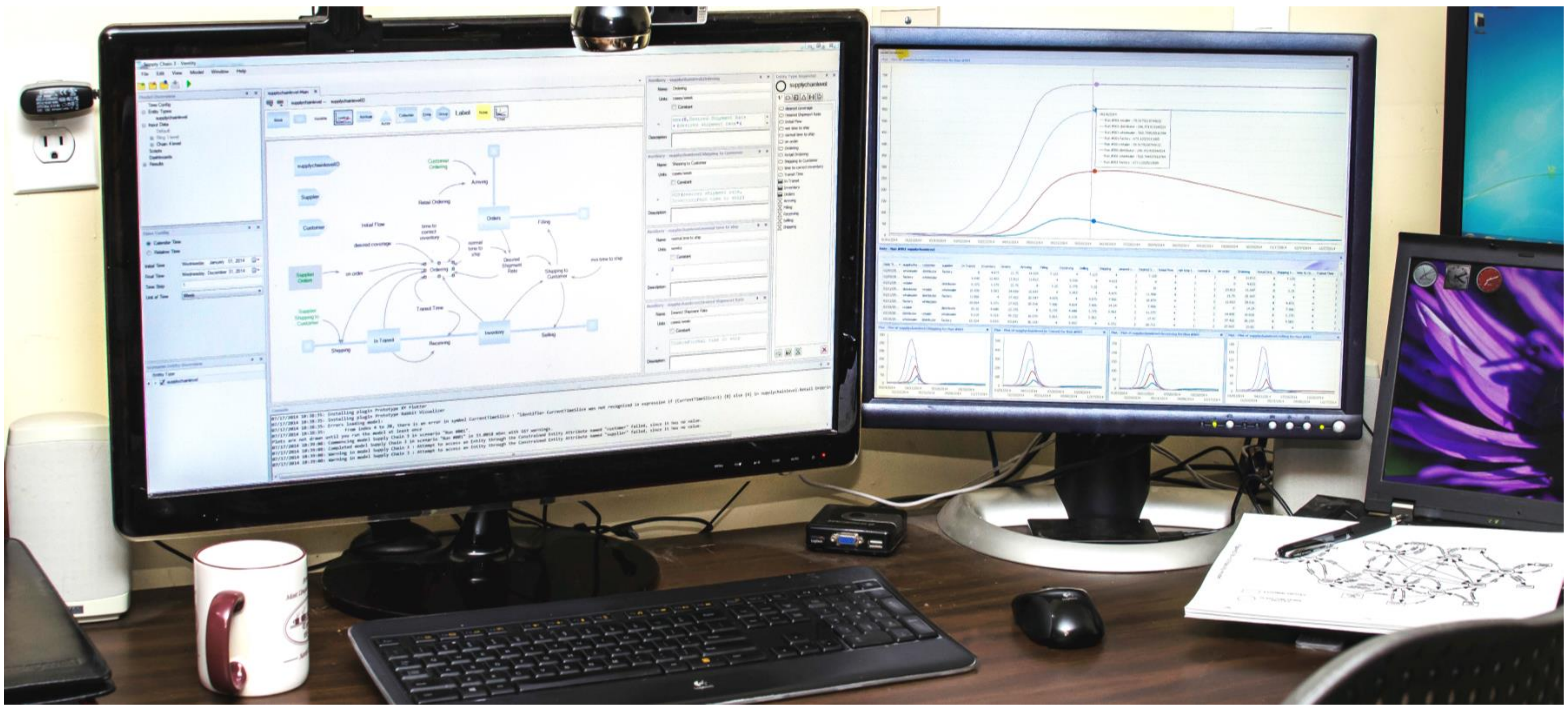
Thomas Fiddaman, tom@ventanasystems.com

David Peterson. david@ventanasystems.com

Ventana Systems, Inc.
60 Jacob Gates Road
Harvard, MA 01451
ventity.biz

Abstract

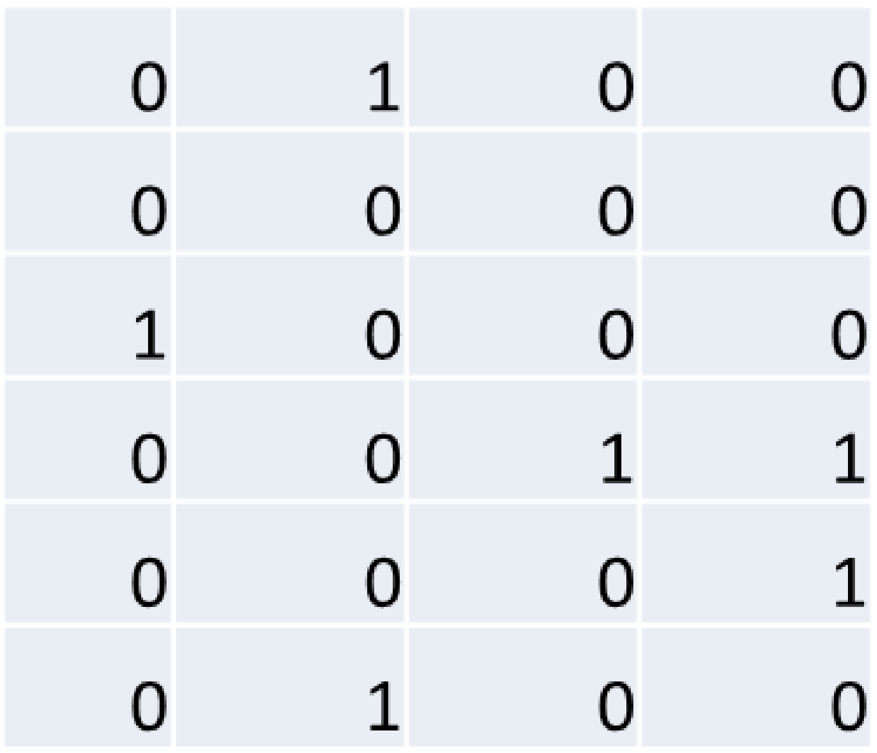
We describe a new platform for system dynamics modeling that supports detailed and object oriented modeling while preserving attractive features of existing tools, including a completely declarative language with a graphical representation. New concepts supporting this platform include collections of entities, attributes, relationships, aggregation and allocation functions, and actions, which are presented with examples. The design facilitates modularity and collaboration, provides a more natural description of detail than arrays, and solves sparse matrix problems. It has application to both traditional system dynamics, with modular sectors, and to agent based modeling.



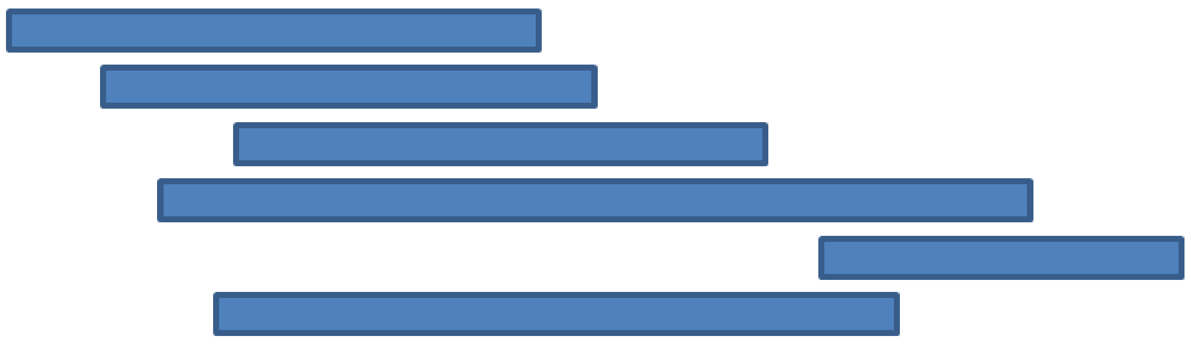
Why?

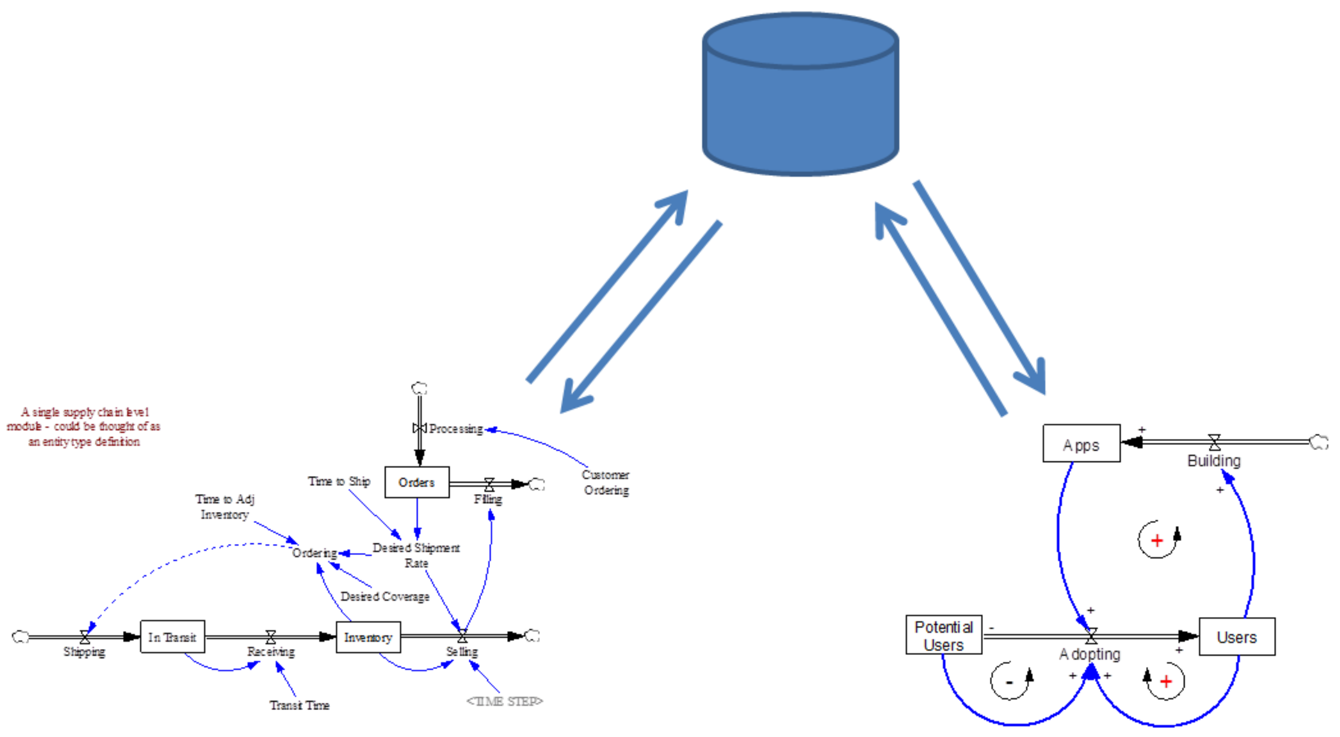
Complex Systems

- Project
  - Task-prerequisite matrix is sparse; arrays are inefficient
  - Project data is granular and relational
- Loan portfolio
  - Use individual statistics
  - Dynamically create & delete loans from data
- Climate Integrated Assessment
  - Delegate atmospheric and economic sectors to different teams
  - Initialize regional emitters flexibly from data
- Social network
  - Hard to determine rules for aggregation a priori
  - Sparse matrix of contacts
- Competitive dynamics
  - Multiple products, lines, customers … not a natural matrix
  - Product lifecycle cohorts
- Big Data
  - Hard to draw the stock-flow network a priori
  - Individual statistics



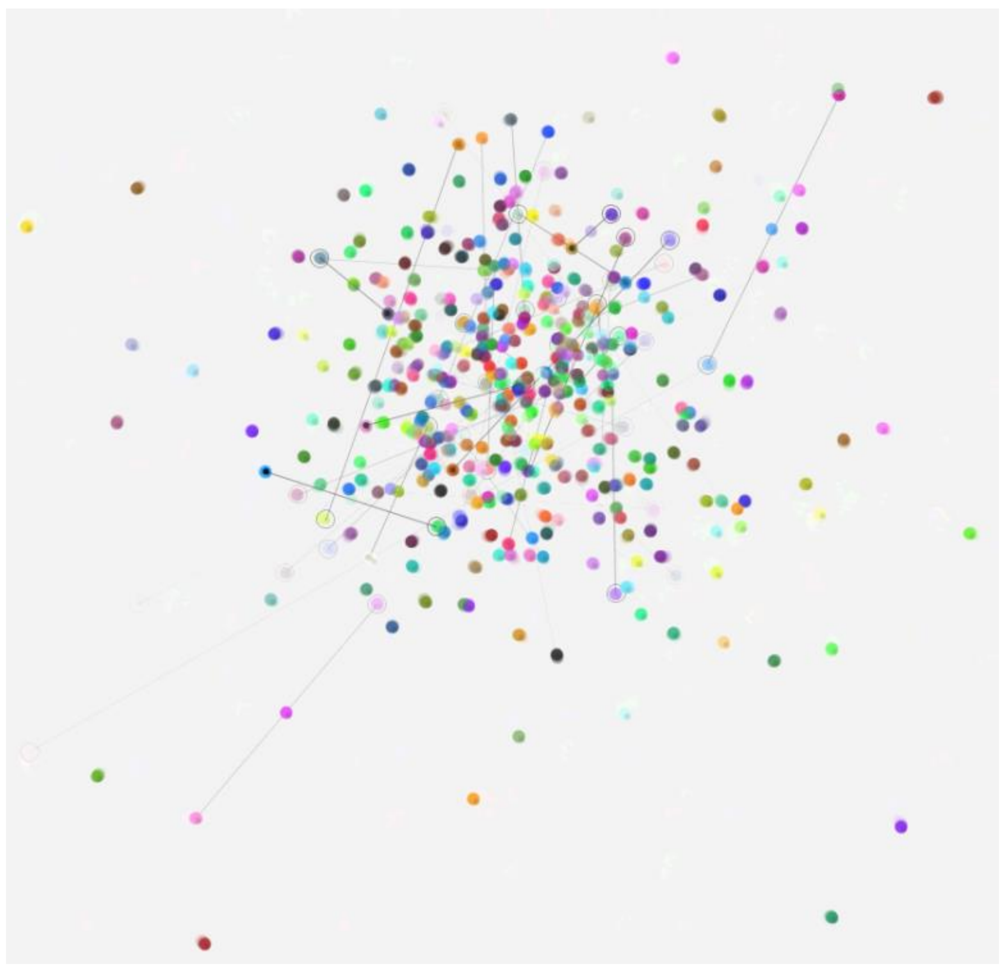
0	1	0	0
0	0	0	0
1	0	0	0
0	0	1	1
0	0	0	1
0	1	0	0





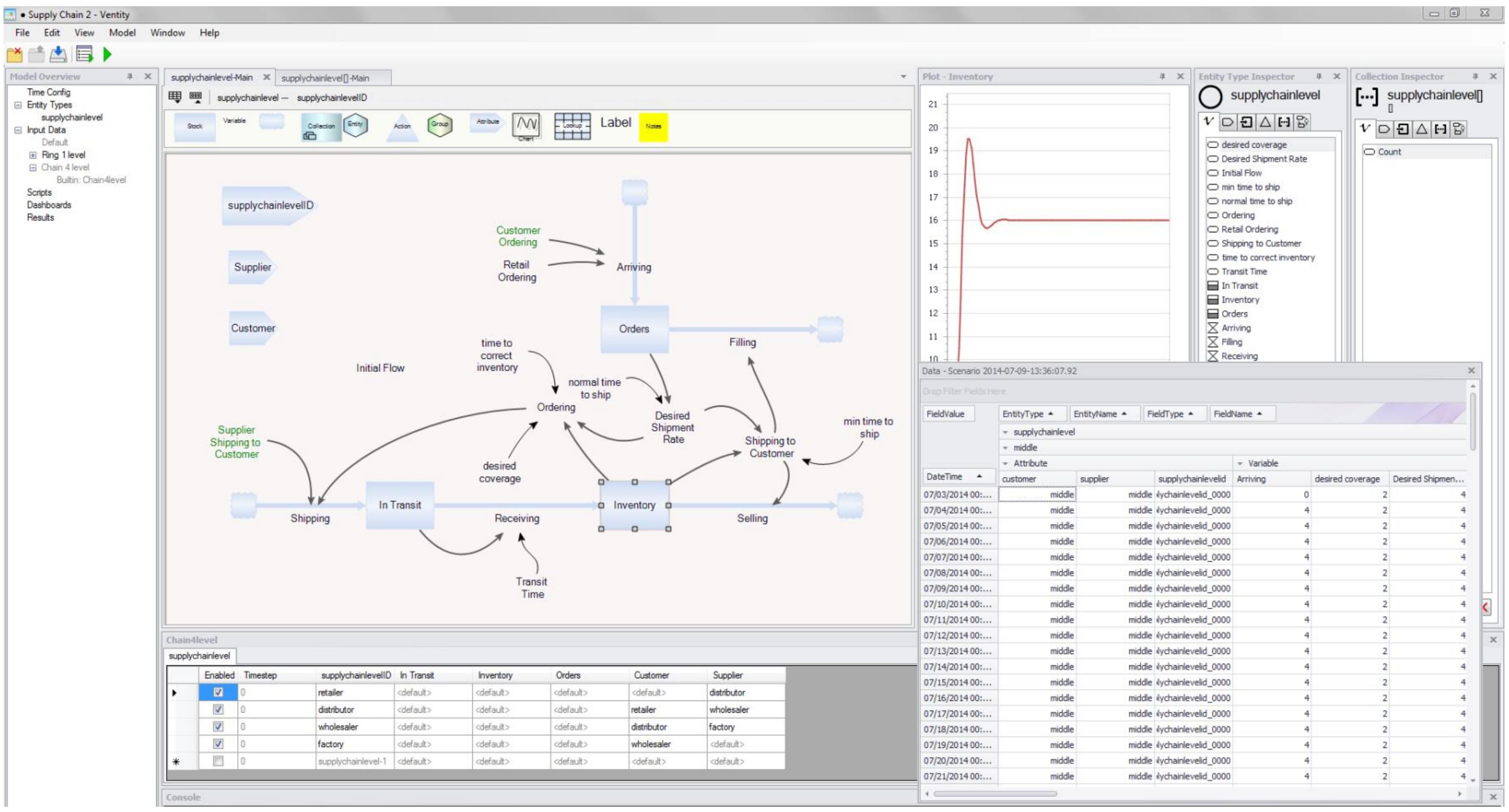
Complex Situations

- Involve non-modelers and spreadsheet users
- Capture knowledge from subject matter experts
- Persuade decision makers



Ingredients for productivity & quality

- Integrated development environment
- Pure declarative language (no code)
- Advanced graphics
  - Informative diagram
  - Interactive charts and visualization
- Rich, efficient representation
- Seamless data connections
- Interaction
  - Synthesim
  - Data exploration & pivots
- Run management
  - Scenario management
  - Advanced algorithms
- Team collaboration, model federation and submodels
- Quality control
  - See everything, all the time—Causal Tracing™
  - Automated Reality Checks™



Chain4level

supplychainlevel

	Enabled	Timestep	supplychainlevelID	In Transit	Inventory	Orders	Customer	Supplier
▸	☑	0	retailer	<default>	<default>	<default>	<default>	distributor
	☑	0	distributor	<default>	<default>	<default>	retailer	wholesaler
	☑	0	wholesaler	<default>	<default>	<default>	distributor	factory
	☑	0	factory	<default>	<default>	<default>	wholesaler	<default>
*	☐	0	supplychainlevel-1	<default>	<default>	<default>	<default>	<default>

Data - Scenario 2014-07-09-13:36:07.92

Drop Filter Fields Here

FieldValue | EntityType | EntityName | FieldType | FieldName

	supplychainlevel					
	middle					
	Attribute			Variable		
DateTime	customer	supplier	supplychainlevelid	Arriving	desired coverage	Desired Shipmen...
07/03/2014 00:...	middle	middle	lychainlevelid_0000	0	2	4
07/04/2014 00:...	middle	middle	lychainlevelid_0000	4	2	4
07/05/2014 00:...	middle	middle	lychainlevelid_0000	4	2	4
07/06/2014 00:...	middle	middle	lychainlevelid_0000	4	2	4
07/07/2014 00:...	middle	middle	lychainlevelid_0000	4	2	4
07/08/2014 00:...	middle	middle	lychainlevelid_0000	4	2	4
07/09/2014 00:...	middle	middle	lychainlevelid_0000	4	2	4
07/10/2014 00:...	middle	middle	lychainlevelid_0000	4	2	4
07/11/2014 00:...	middle	middle	lychainlevelid_0000	4	2	4
07/12/2014 00:...	middle	middle	lychainlevelid_0000	4	2	4
07/13/2014 00:...	middle	middle	lychainlevelid_0000	4	2	4
07/14/2014 00:...	middle	middle	lychainlevelid_0000	4	2	4
07/15/2014 00:...	middle	middle	lychainlevelid_0000	4	2	4
07/16/2014 00:...	middle	middle	lychainlevelid_0000	4	2	4
07/17/2014 00:...	middle	middle	lychainlevelid_0000	4	2	4
07/18/2014 00:...	middle	middle	lychainlevelid_0000	4	2	4
07/19/2014 00:...	middle	middle	lychainlevelid_0000	4	2	4
07/20/2014 00:...	middle	middle	lychainlevelid_0000	4	2	4
07/21/2014 00:...	middle	middle	lychainlevelid_0000	4	2	4

Extensions of the SD paradigm

- Collections, create and delete actions map to stocks, inflows and outflows
- Attribute changes move entities among collections like flows move continuous quantities between stocks
- Replace arrays, which make work by making an object property into equation properties
- Dynamic creation and deletion of model components and relationships creates structure on the fly

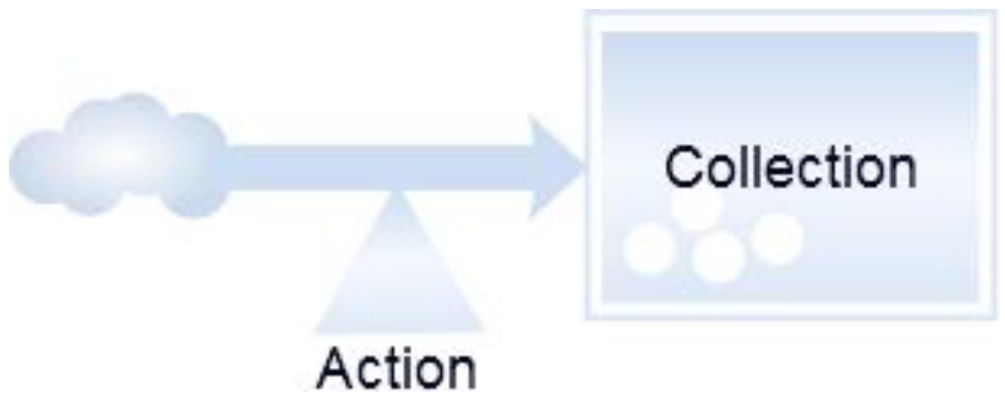
Elements

- The modeler creates, or reuses, a separate **entity type definition** (ETD) for each component in the system. These are intentionally similar to class definitions in an object oriented programming language. Each ETD can be defined using system dynamics stock and flow diagrams.
- Individual **entity instances** are created at runtime, and may be specified by external data sources, making it easier to use a model or its components across many separate applications.

- **Collections** contain lists of entities. They can also be referenced as a variety of subcollections, corresponding to subsets of attribute values. Supercollections can contain entities of different types, e.g., a collection of businesses containing restaurants and factories.
- Each entity contains one or more identifiers, called **attributes**, which identify, categorize, and logically differentiate entities created from the same entity type definition. Attributes serve a similar role to key fields in relational database tables and arrays in traditional SD. They support the modeling of references, relationships, collections, and aggregated reporting.

- **References** permit an entity to access the variables of any other entity via a unique identifier. **Relationship entities** link two or more other entities with references, as in a social network or a task prerequisite matrix.
- **Aggregate** functions summarize the properties of many entities in a collection for use by one entity elsewhere. Corresponding **allocation** functions provide the opposite mapping, as when many moviegoers compete for scarce seats in a theater, or many to many relations, as when multiple buyers seek a product from multiple suppliers.

- **Actions** provide discrete events for dynamic creation of entities, changing attributes, delete, split, and merge. Actions can be used to package transactions or flows that must occur together in order for an internally consistent change of the model state to occur; for example, double entry bookkeeping. in a financial model.


Collection
Action