An extensible object-oriented and agent-based framework for modelling and simulating supply chains

Li Tan*

School of Electrical Engineering and Computer Science,
Washington State University,
Richland, WA 99354, USA
E-mail: litan@wsu.edu
*Corresponding author

Shenghan Xu

College of Business and Economics,
University of Idaho,
Moscow, ID 83843, USA
E-mail: shenghan@uidaho.edu

Benjamin Meyer and Brock Erwin

School of Electrical Engineering and Computer Science,
Washington State University,
Richland, WA 99354, USA
E-mail: berwin@wsu.edu

Abstract: We propose an extensible object-oriented agent-based framework for modelling and simulating supply chains. A problem with existing supply-chain analysis tools is that most of them are designed only for specific configurations of supply chains. The primary goal of this work is to provide an open and extensible framework for analysing supply chains with heterogeneous elements and structures. Our framework incorporates the following features:

1. It adopts an agent-based approach to handle interactions among elements of a supply chain, and an analyst can introduce new types of elements for a specific application;
2. To promote the design reusability, we propose an object-oriented type system that supports behaviour inheritance;
3. The framework formally defines a meta-model for elements of a supply chain;
4. The framework includes a discrete-event simulation algorithm, which defines interactions among elements via messages and deliveries.

We also discuss SIMRISK, our Java implementation of the framework.

Keywords: agent-based modelling and simulation; design extensibility; formal semantics; supply chains.

Copyright © 2012 Inderscience Enterprises Ltd.

Reference to this paper should be made as follows: Tan, L., Xu, S., Meyer, B. and Erwin, B. (2012) ‘An extensible object-oriented and agent-based framework for modelling and simulating supply chains’, *Int. J. Information and Decision Sciences*, Vol. 4, Nos. 2/3, pp.251–267.

Biographical notes: Li Tan received his PhD in Computer Science in 2002 from State University of New York at Stony Brook. He is an Assistant Professor in the School of Electrical Engineering and Computer Science, Washington State University, the USA. His research interests include formal verification, software testing, model-based design, and supply-chain modelling and analysis.

Shenghan Xu received her PhD in Operations Management from the University of Massachusetts Amherst, the USA. She is an Assistant Professor in the College of Business and Economics, the University of Idaho, the USA. Her research interests fall into the area of supply-chain management, especially in supply-chain consolidation, risk analysis, and network optimisation.

Benjamin Meyer received his BS in Computer Science from Washington State University Tri-Cities. His research interests include supply-chain modelling and analysis, and computational music science.

Brock Erwin is an undergraduate student in Washington State University Tri-Cities. His research interests include supply-chain analysis and high performance simulation of biological systems.

This is a revised and expanded version of a paper presented at the 2009 IEEE International Conference on Information Reuse and Integration (IRI’09), Las Vegas, Nevada, USA, 10–12 August 2009.

1 Introduction

Supply chains concern the movement of merchandise from suppliers to customers’ hands. With increasing integration of the world’s economy, the size and the complexity of global supply chains are rising rapidly, and so is the reliance of our economy on these supply chains. Understanding and optimising the behaviours of these large-scale global supply chains are essential for a variety of topics in supply-chain management, including supply-chain risk management (Chen and Zhang, 2008), contracting (van Delft and Vial, 2004), and performance evaluation (Wei et al., 2007). To reduce cost and maintain profit margin, many companies engage themselves in global supply-chain expansion involving different elements of supply chains such as suppliers, distributors, retailers, and logistics providers across multiple continents (Ferrer and Karlberg, 2006). For example, Costco operates its 544 warehouse stores in North America, South America, Asia, and Europe. It sources merchandise from all over the world (Costco Wholesale Corporation, 2007). Geographically, the participants of a global supply chain are distributed across continents. Generally, these participants act as autonomous agents with their own business logic and they answer to their own continency. A research question is how to model and analyse a supply chain with a diversified profile of elements.

The primary goal of this work is to provide an open and extensible framework for modelling and analysing heterogeneous supply chains. The structure and behaviour of a supply chain are heavily influenced by its underlying business model and market. For example, Walmart's supply-chain operation follows a more traditional setting. It contains multiple echelons of suppliers, distribution centres, and stores. In contrast, Dell uses a direct channel approach that eliminates intermediate layers. These two supply chains differ not only in structure, but also in functionality and strategy at the element level. For example, in Dell's 'build-to-order' model, supplier sites have manufacturing capabilities that can assemble custom-built computers, whereas for most retail chains, suppliers are simply warehouses of merchandise providers. Because of the differences and complexity in supply-chain operations, most existing supply-chain tools choose to target at only a specific set of supply chains with limited extensibility. These tools have a limited capability in terms of how much an analyst can extend the tools for his/her own application: although many of the existing tools allow an analyst to instantiate new elements from existing types, they provide limited [e.g., GBSE (Wang et al., 2008)] or no option [e.g., EasySC (Liu et al., 2004)] for an analyst to introduce a new type of element or redefine the behaviour of an existing type. This shortcoming severely restricts the development and use of general-purpose supply-chain modelling and analysis tools, since limited built-in element types cannot meet the demand of modelling diversified profiles of supply chains that real-world companies operate. This work is to address this shortcoming by proposing an extensible object-oriented and agent-based framework. Since supply chains are heavily influenced by their underlying business models and markets, one can never anticipate types of elements that may appear in a real-world supply chain. So instead of restricting an analyst to a set of predefined types of elements, we provide his/her a higher degree of flexibility by supporting custom-defined types of elements. Using our object-oriented and agent-based formal framework, an analyst can instantiate an existing element type and/or introduce a new type of element. Furthermore, to improve design reusability, our framework allows an analyst to define a new type of element by inheriting the attributes and behaviours of an existing type.

The proposed framework makes the following contributions with the focus on improving the extensibility and reusability of supply-chain analysis tools: first, the framework uses an agent-based approach that supports custom-defined element (agent) types. The framework models supply-chain elements as autonomous agents, and it provides common functionalities for studying interactions between elements. Analysts can define their own types of elements for underlying supply-chain applications; second, to promote the design reusability and ease the difficulty in introducing a new type of element, the framework incorporates an object-oriented type system that supports behaviour inheritance. An analyst can focus on defining behaviour unique to a new type of element, while inherits common behaviour from existing types; third, the framework defines a meta-model for supply chains and formally defines its semantics. The meta-model provides a behavioural and structural abstraction of various supply-chain models. The formal semantics removes ambiguity in defining, interpreting, and validating a supply-chain model; finally, based on the formal semantics of the meta-model, we propose a discrete-event simulation engine for simulating supply chains.

Our framework uses an agent-based approach for modelling and analysing complex supply-chain systems. With a growing number of businesses considering overseas suppliers as a way to cut cost, a global supply-chain operation often consists of a large number of facility nodes including suppliers, warehouses, and retailers, each of which can be seen as an (semi-)autonomous decision-maker. Agent-based approach has been successfully used to study a wide range of complex systems (Axelrod, 1997). These systems typically consist of groups of autonomous agents. Agent-based modelling can trace its root back to Von Neumann's theory on cellular automaton (Neumann, 1966), but it gets its deserved attention only recently because the advance in computing hardware makes it a reality to simulate interactions between a large collection of agents on even commercial off-the-shelf (COTS) computers, and emerging multi-core desktop computing platforms accelerate such trend.

We adopt an agent-based approach because the challenge of simulating complex supply chains is the exact problem the agent-based modelling technique is prescribed for: although the behaviour of each facility node may be simple to understand, the overall behaviour of a supply chain as the result of interactions among these nodes is not. Our framework models elements of a supply chain as agents, which include facility nodes (e.g., warehouses), transportation links (e.g., routes), and other special-purpose elements. To improve extensibility, the framework provides several layers of abstraction that separate common functionalities from the decision logics and the physical structures unique to each type of element: first, the framework makes a clear distinction between internal behaviours of elements of a supply chain and their interactions. The framework provides common functionalities such as message passing for modelling interactions among agents, whereas it leaves the definition of internal behaviour and structure of an element to the discretion of an analyst; second, the framework provides a meta-model for supply-chain elements. The meta-model abstracts common behaviours and structures of elements. For example, the meta-model defines a structural interface including ports, and it also provides a skeleton for defining common internal behaviours such as internal merchandise transformation. To define a new type of element, an analyst only needs to fill in the decision logic unique to that type of the element.

A distinctive feature of our framework is that it introduces a formal syntactical and semantical definition of the meta-model. A common problem for existing supply-chain tool (cf. Liu et al., 2004) is that they often provide only a narrative and casual description of the semantics of a supply-chain model. Without a rigorous definition of a supply-chain model, it often falls into one's guess work to precisely understand a supply-chain model and unambiguously interpret analysis result. Moreover, a casual definition of the semantics of a model makes it harder to validate analysis result since the expected behaviour of a supply-chain model is not rigidly defined. For the same reason, lack of formal semantics makes it harder to tell bugs from legitimate features in tools implementations. The formalism we introduced has several benefits: it helps validate simulation results and also reduces errors in tool implementations; the formalism also facilitates formal analysis of a supply chain. For example, our formal framework supports a model-checking-based risk analysis approach proposed by Tan and Xu (2008).

To reduce overhead in defining a new type of element and improve the design reusability, we propose an object-oriented type system. In the type system, types of elements are defined as classes and elements are instances of these element classes. The type system supports the design reuse by class inheritance. For example, when defining a new type of element, say, a special type of warehouse, an analyst can inherit common functions and interfaces from a base warehouse class, and override only those of the methods representing the decision logic and the internal structure unique to the new type of warehouse.

To see our proposed framework in action, we developed a discrete-event simulation algorithm and implemented the framework in Java. Simulation remains as an important tool for analysing the behaviour of a supply chain. Compared with other methods such as stochastic analysis, simulation does not heavily tax one's theoretical background and analysis skills. Simulation results can be visualised and explained to executives and other stakeholders with little or no background on supply-chain management. With the introduction of more powerful hardware, especially emerging multi-core architecture, there are renewed interests in recent years in desktop-based simulation tools for supply chains (Wang et al., 2008). The simulation algorithm provides the guideline for implementing a discrete-event simulation engine for the framework. It is also part of our formal framework in which it defines simulation semantics of a supply-chain model.

The rest of the paper is organised as follows: in Section 2 we discuss related works; in Section 3 we introduce a meta-model for formally defining supply-chain elements. The meta-model defines the interface of an element (Section 3.1), its internal and external behaviours (Section 3.2), and constraints (Section 3.3). In Section 4, we introduce our simulation algorithm and define a simulation-based formal semantics for a supply-chain model. In Section 5, we propose an object-oriented type system based the meta-model to promote the design reusability and to reduce the overhead of defining a new type of element. In Section 6, we discuss SIMRISK, a prototype implementation of our proposed framework. Finally, Section 7 concludes the paper and discusses future research directions on this subject.

2 Related works

Supply-chain modelling and simulation have attracted much research interest recently. For instance, Liu et al. (2004) introduced a Java-based supply-chain simulation tool Easy-SC. In Easy-SC modelling environment, facility nodes are instantiated from six predefined enterprise node types, and routes were defined as connecting arcs between facility nodes. Wang et al. (2008) discussed a general business simulation environment (GBSE) developed in IBM China research lab. GBSE is a Java-based event-driven simulation tool built on top of the Eclipse platform. GBSE defines three types of facility nodes and one type of link. These tools provide limited (e.g., GBSE) or no option (e.g., EasySC) for an analyst to define a new type of element. In contrast, our approach improves the tools extensibility and the design reusability by allowing an analyst to define his/her own types of elements and/or to redefine existing ones.

Agent-based approaches have been explored by other researchers for simulating supply chains. Swaminathan et al. (1998) proposed a multi-agent approach for modelling supply chains. They believed a multi-agent approach was '*a natural choice*' for modelling supply chains because '*supply-chain management is fundamentally concerned with coherence among multiple decision makers*' (Swaminathan et al., 1998). They modelled structural elements as agents, which interacted with each other using control elements. Our agent-based research follows the same line but one of our improvements is to introduce a meta-model for supply chains and formally define its semantics.

On the implementation side, Rossetti et al. (2007) proposed an objective-oriented framework for simulating supply chains. The work is based on a generic simulation package JSL developed by Rossetti (2008). Although the framework and its implementation provide some flexibility for an analyst to customise decision logic of a node, the internal structure of a node is fixed and the analyst cannot change it. In this work we propose an object-oriented type system that supports the customisation of the decision logic of a node. In addition, the meta-model in Section 3 allows an analyst to customise the internal structure of a node.

3 An agent-based meta-model for supply chains

Agent-based modelling studies interactions among autonomous agents. A typical supply-chain consists of elements such as facility nodes and routes that interact with each other via orders, shipments, and other communication methods. The complexity of a large-scale supply chain arises not only from dynamics of individual elements, but also from interactions among these elements. This makes supply chains an ideal application for agent-based modelling. Our modelling technique is based on the agent-based modelling technique with special attention for extensibility.

In our framework, elements of a supply chain are modelled as autonomous agents with their own decision logics. Types of these elements range from facility nodes (e.g., warehouses, suppliers, and retailers, etc.), to transportation links (e.g., routes), to special-purpose elements (e.g., order processing centre). Agent modelling is the first step of our modelling workflow and it is also at the core of our proposed framework. Agent modelling includes physical modelling and behaviour modelling. Our goal for agent modelling is to define a general and extensible meta-model that can

1 model a variety of elements of a supply chain including facility nodes and routes

2 provide a rigid syntax and semantics definition for an element.

Definition 1 gives the formal definition of an element. Next, we model the interface of an element using ports (Definition 2), then proceed to behaviour modelling including merchandise transformation (Definition 2), message sending (Definition 7), and delivery decision (Definition 8). Finally, we discuss constraints (Definition 9) imposed on an agent.

Definition 1 (Element): An element of a supply chain is defined as a tuple $\langle P, U, f_t, f_m, f_d, C \rangle$, where P is a set of ports, U is a set of modes, f_t is a merchandise transformation function, f_m is a message sending function, f_d is a delivery decision function, and C C is a set of constraints.

To model an element, we need to model its internal and external behaviours. The former defines how merchandise move within the boundary of an element, and the latter defines how elements interact with each other via shipment and message-based communication.

3.1 Ports and deliveries

The interface of an element is defined by ports. Depending on the direction of its merchandise flow, a port is either an in-port or an out-port. A duplex port may be defined as a combination of an in-port and an out-port. Each port may buffer the merchandise flow up to a given inventory size.

An element may consist of both in- and out-ports. These ports are the interface of the element via which the element receives and/or delivers merchandise. Definition 2 gives the formal definition of ports. We use the following notations in the rest of this paper: we denote $a.p$ for port p of element a, and $a.p.inv$ for inventory at port p. We denote $a.P^+$ and $a.P^-$ for all a's in-ports and out-ports, respectively. We write $A.P^+$ (or $A.P^-$) in lieu of $\bigcup_{a\in A} a.P^+ (or \bigcup_{a\in A} a.P^-)$. We denote Z for the set of all the integers, and R for the set of all the real numbers. Z^+ and R^+ represent the non-negative subsets of Z and R, respectively.

Definition 2 (Ports): A port may contain its own buffer. It is either an in- or out-port. Formally, the state of a port is defined as a tuple $p = \langle inv, dir \rangle$, where $inv \in Z^+$ is the amount of merchandise stacked at p and $dir \in \{+,-\}$ is the direction of the port. p is an input port if $p.dir = +$, or an output port if otherwise.

Definition 3 (Deliveries): Let p^- be an out-port of an element that is connected to input ports $p_1^+,...,p_l^+$ of some elements, a delivery from p^- is a vector $\hat{\delta}_{p-} = \langle \delta_{p-p_1^+},...,\delta_{p-p_l^+} \rangle$, where $\delta_j \in Z^+$ is the amount of merchandise delivered to p_j^+ from p^-.

By Definition 3, a delivery at an out port p^- is characterised by the amounts of merchandise delivered to receiving in-ports. The total delivery received by an element a in a supply chain $\mathcal{S}$ is defined as,

$$\sum_{p^- \in \mathcal{S}.P^-} \sum_{p^+ \in a.P^+} \delta_{p^-p^+}$$

The total delivery sent by a is defined as,

$$\sum_{p^+ \in \mathcal{S}.P^+} \sum_{p^- \in a.P^-} \delta_{p^-p^+}$$

3.2 Behaviour modelling

Definition 4 (Modes and states): A state of an agent a is defined as a tuple $\langle p_1.inv,...,p_l.inv,u \rangle \in (Z^+)^l \times U$, where $p_i.inv$ is the inventory of ith port of a and U is the set of a's modes.

A state of the element is defined by its mode and inventory at its ports. We use the following notations in the rest of the paper: given a state $s = \langle p_1.inv,...,p_l.inv,u \rangle$, $s[p_1.inv \leftarrow p_i.inv']$ is a new state s' obtained by replacing p_i's inventory $p_i.inv$ by $p_i.inv'$, i.e., $s' = \langle p_1.inv,...,p_l.inv',...,u \rangle$. Similarly, $s[u \leftarrow u']$ is a new state obtained by replacing mode u in s by u'.

In our agent-based framework, the behaviour of an element is defined by its internal merchandise transformation, message sending and processing, and delivery decisions.

Internal merchandise transformation: Internal merchandise transformation depicts activities that produce and/or transport merchandise inside an element. In case of a route, merchandise is moved from its receiving in-port to its out-port. In case of a manufacturing site, raw materials received at in-ports are transformed into finished products at out-ports.

Definition 5 (Merchandise transformation functions): A merchandise transformation function of an element a has form of $f_t : S \to S$, where S is the set of a's states. In addition, $f(\langle a.p_1.inv, a.p_2.inv,..., a.p_l.inv, u \rangle) = \langle a.p_1.inv', a.p_2.inv',..., a.p_l.inv', u' \rangle$ satisfies the following constraints, 1 $a.p^k.inv' - a.p^k.inv \leq 0$ if p^k is an out-port. $a.p^k.inv' - a.p^k.inv$ represents the amount of merchandise consumed at port p^k.

2 $a.p^k.inv' - a.p^k.inv \geq 0$ if p^k is an in-port. $a.p^k.inv' - a.p^k.inv$ represents the amount of merchandise produced at port p^k.

Definition 5 is general enough to describe a variety of internal merchandise transformation activities. For a manufacturing site a, $a.p^k.inv' - a.p^k.inv$ represents the amount of raw material consumed if $a.p^k$ is an in-port, and the amount of finished products produced if $a.p^k$ is an out-port. It should be noted that a merchandise transformation function does not restrict types of raw materials and finished products. Raw materials and finished products can be different types of merchandise, and in reality, they often are.

For a route, $a.p^k.inv' - a.p^k.inv$ represents the amount of merchandise being transported. The transportation of merchandise does not have to occur instantaneously. Transportation delay can be modelled in the meta-model. This can be done by, for example, further decomposing modes and introducing finite queues. In such a case, the amount consumed at an in-port is not instantly transferred to out-ports; instead, it is pushed to a finite queue, and the amount transferred to out-ports is dequeue from the same queue. Since the size of an internal queue and its content are finite, the status of the queue can be encoded as part of modes.

Message sending and processing. In our agent-based framework, elements communicate with each other through messages. Each message contains a receiving element (receiver) and an action. An action triggers a transition of mode at its receiver. A receiver processes a message and changes the element's mode as the result.

Definition 6 (Actions and messages): An action α of an agent a is defined as a function $\alpha : U \rightarrow U$, where U is the set of a's modes. A message for agent a is a tuple $\langle a, \alpha \rangle$, where α is an action of a.

Definition 7 (message sending function): Let $\mathcal{S}$ be a supply chain, a message sending function of an agent a in $\mathcal{S}$ is a function $f_m : S \rightarrow 2^M$ where S is the mode of a, and M is the set of messages generated in $\mathcal{S}$.

Message sending function in Definition 7 describes how messages are generated by elements. Based on its mode, an element may send a set of messages and enter the next mode. Once generated, a message is routed to its destination. The set of messages sent by an element may also be empty, meaning that it does not send out any message at the current mode. Such a mode is called a *silent* mode.

Delivery decision: Delivery decisions made by an agent are defined by a delivery decision function. A delivery decision function decides how much merchandise an out-port shall deliver and how it shall be distributed to the in-ports connected to the out-port. A delivery decision function makes its decision based on the current state. Definition 8 gives the formal definition of Delivery Decision Function. In other words, a delivery decision function has form of $f_d(s) = (\langle \delta_{p_1^- p_{11}^+}, ..., \delta_{p_1^- p_{1q1}^+} \rangle, ..., \langle \delta_{p_m^- p_{m1}^+}, ..., \delta_{p_0^- p_{mqm}^+} \rangle)$, where $p_{i1}^-, ..., p_{iqi}^+$ are in-ports connected to agent a's ith out-port p_i^-, and $\langle \delta_{p_i^- p_{i1}^+}, ..., \delta_{p_0^- p_{iqi}^+} \rangle$ is an outgoing delivery at p_i^-.

Definition 8 (Delivery Decision Function): Let $\mathcal{S}$ be a supply chain and a be an element of $\mathcal{S}$ with m out-ports. A delivery decision function of agent a is a function

$f_d : \mathcal{S} \rightarrow (\hat{\delta}_{p_1^-} \times \ldots \times \hat{\delta}_{p_m^-})$, where S is the set of states of a, and $\hat{\delta}_{p1}^-$ is an outgoing delivery at out-port p_i^-.

3.3 Constraints

In our framework, an element may be imposed with a set of constraints. Constraints can be used to model physical or logical limitations imposed on an agent. For example, constraint can be used to model a facility a with limited inventory space V, in which case a constraint for a can be defined as $\sum_{p \in a.P^+ \bigcup a.P^-} p.inv \leq V$, where $a.P^+ \bigcup a.P^-$ is the set of all the a's ports.

Definition 9 (Constraints): Let $\mathcal{S}$ be a supply chain. A constraint for element a is a predicate c over states of a, that is, $c : S \rightarrow$ {true, false}, where S is the set of a's states. An execution ρ of $\mathcal{S}$ is invalid if an agent a can reach a state s such that $c(s)$ is false, where c is one of a's constraints.

3.4 Examples and special cases

Our agent-based framework is general enough to define the structures and behaviours of a variety of elements. For example, let us consider two very different categories of elements: facility nodes and routes. A route is a special kind of element that has precisely 1 in-port and 1 out-port. Its merchandise transformation function transfers merchandise from the in-port of a route to its out-port, often with delay. The delay may be the result of multiple factors such as route scheduling and transportation delay.

Facility nodes may be further classified to several categories. For example, a retailer has only in-ports. Its internal merchandise function models the consumption of merchandise at its demand rate. A warehouse has both in- and out-ports. Its merchandise transformation function and delivery function shall satisfy the flow balance equation. That is, its inventory before an update plus incoming deliveries shall be the same as its inventory after the update minus outgoing deliveries.

An advantage of our framework is that an analyst can define his/her own type of elements for target supply-chain applications. For example, in a traditional retail setting, a supplier has only out-ports, but for companies like Dell, a supplier's site also has manufacturing capability. In such a case, a supplier node has both in- and out-ports. Its merchandise function models how raw materials are consumed at in-ports and finish products are produced at out-ports.

4 Supply-chain semantics and simulation

We formally define the meaning of the meta-model using its simulation semantics. That is, the semantics of a supply-chain model in our framework is defined by its simulation traces. Algorithm 1 defines our simulation algorithm.

Semantically, a supply-chain model is a synchronous system extended with messages. Each iteration in Algorithm 1 simulates a clock update. A clock update is a basic time unit in supply-chain planning. Depending on planning horizon of underlying supply-chain operations, an update may represent a hour, a day, or a month, etc. Each iteration starts with internal merchandise transformation: lines 2–5 call the merchandise transformation function of each agent. As a result, an element enters its next state and sends out messages as defined in its message sending function. In general, whenever an element changes its state, its message sending function is called to check if the element needs to inform others of the change of its state via messages.

Algorithm 1 Simulate ($\mathcal{S}$)

```
Require: A supply chain with $\mathcal{S}$ with a set of *agents* A, where each agent $a \in A$ has a start state $s_0^a$
1   **while** true **do**
2       **for all** $a \in A$ **do**
3           $s^a = f_t(s^a)$;
4           $M = M \cup f_m^a(s^a)$
5       **end for**
6       processMsg( )
7       **for all** $a \in A$ **do**
8           *//Suppose* $a$ *has* $k$ *out-ports* $p_1^- ... p_k^-$
9           $(\hat{\delta}_{p_1^-}, ..., \hat{\delta}_{p_k^-}) = f_d(s^a)$;
10          **for all** $p^- \in \{p_1^-, ..., p_k^-\}$ **do**
11              *//* $p^-$ *is connected to* $p_1^+ ... p_q^+$
12              $\langle \delta_{p^- p_1^+}, ..., \delta_{p^- p_q^+} \rangle = \hat{\delta}_{p^-}$
13              **for all** $p^+ \in \{p_1^+, ..., p_q^+\}$ **do**
14                  // $p^+$ belongs to agent $d$.
15                  $s^d = s^d[d.p^+.inv \leftarrow (d.p^+.inv + \delta p^- p_q^1)]$
16                  $M = M \cup f_m^d(s^d)$
17              **end for**
18              $s^a = s^a[a.p^-.inv \leftarrow (a.p^-.inv - \delta_{p^- p_1^+} .. - .\delta_{p^- p_q^+})]$
19              $M = M \cup f_m^a(s^a)$
20          **end for**
21      **end for**
22      processMsg( )
23  **end while**
```

Algorithm 2 processes messages generated by elements. During every iteration, it takes a message $\langle b, \beta \rangle$ from a message pool M and applies action β on element b. β may change the mode of b. Element b may generate messages at the new mode, or its new mode can be a silent mode with no message being sent. Algorithm 2 uses a run-to-completion semantics. That is, it exits only after the message pool is empty.

Algorithm 2 ProcessMsg( )

```
1 while $M \neq \emptyset$ do
2     $M = M - \{m\}$ // Take a message $m$ from $M$;
3     $\langle b, \beta \rangle = m$;
4     $s^b = s^b[u^b \leftarrow \beta(u^b)]$;
5     $M = M \cup f_m^b(s^b)$;
6 end while
```

Next, Algorithm 1 calls every element's delivery decision function to compute deliveries. To realise a delivery, it subtracts inventory at an out-port and adds it to the connected in-ports. A delivery can also change the states of sending and receiving elements by changing their inventories at ports. The change of states may cause these agents to send out messages. Algorithm 2 is called afterward to process these messages.

5 An object-oriented type system for defining supply-chain elements

To ease difficulty in defining new types of elements and promote the design reusability, we introduce an object-oriented type system for defining custom types of elements. The UML class diagram in Figure 1 shows the outline of the type system. The design of the type system follows the meta-model introduced in Section 3. At the top of the hierarchy is a built-in abstract type *Agent*, which serves as the base class for all the types of elements. *Agent* summarises the attributes and methods common to all the supply-chain elements. As defined in the meta-model, the interface of an element has in-ports and out-ports. The attributes shared by in-ports and out-ports such as inventory are encoded in the built-in abstract type *Port*. An association between *Agents* is implemented via in-ports and out-ports. Element types subclassing *Agent* may choose to restrict such association. For instance, a retailer may choose to restrict the association to in-ports only.

The design of *Agent* follows the strategy design pattern (Gamma et al., 1994). A strategy is a collection of functions that define the decision logics central to the behaviour of an element. The syntax and semantics of these functions are introduced in Section 3.2 as part of our meta-model for supply-chain elements. The merchandise transformation function (mtf) defines an element's internal merchandise flow, the message sending function (msf) defines communications among elements, and the delivery decision function (ddf) defines distributions of merchandise at out-ports. The hierarchy of strategy classes assembles the hierarchy of agent classes. Figure 1 shows a generic node strategy ($NodeStrat$) and a generic route strategy ($RouteStrat$). The generic route strategy defines the behaviour common to all the routes. Since a route typically does not send a message, $RouteStrat$ implements a null msf, and since a route usually deliveries all the merchandise at its out-port to its destination, ddf of $RouteStrat$ simply moves merchandise at the out-port of a route to the in-port of its attached node. Nevertheless, mtf is more complicated and its logic depends on factors such as route scheduling and transportation delay. mtf is declared as an abstract method in $RouteStrat$ and its actual implementation is deferred to actual route classes. The strategy design pattern decouples decision logics and their hosts. Such separation adds additional flexibility to agent modelling. For instance, nodes with the same structure may be associated with different decision logics. An add-on benefit of adopting the strategy design pattern is that an agent can change its strategy on-the-fly. This feature is especially useful when one models an adaptive supply chain, which changes its policy based on its environment.

To define a new type of element, an analyst starts with subclassing *Agent* or one of its descendants. Each subclass may define new behaviour and/or add constraints. For example,

Node class may restrict the type of its inherited attribute *strategy* to $NodeStrat$, so only a strategy of type $NodeStrat$ or its descendants can be used by a *Node* object. As another example, a *Node* class may choose to introduce a new function for checking its inventory. Class inheritance reduces the overhead of defining a new type of element. Note that a type on a lower layer of the hierarchy automatically inherits the behaviours of its ancestors. To define a new type, an analyst may choose to subclass the lowest type in the hierarchy that is a generalisation of the new type. This allows the analyst to focus on behaviours and constraints unique to the new type, and at the same time inherit the generalised logic from the ancestors of the new type. For example, in Figure 1, the type definition of a retailer with just-in-time restocking policy ($JITStore$) may start as a subclass of a more general type Retailer.

6 Implementation

We implemented an initial prototype of the proposed agent-based formal framework in SIMRISK. SIMRISK is an integrated tool for supply-chain modelling, simulation, and risk analysis. It implements a visual integrated development environment (IDE). Its IDE provides three different views of a supply chain: a hierarchical presentation of elements of the supply chain (tree view), a geographic view of elements (network view), and a property page for displaying attributes of a selected element (property view). SIMRISK IDE can visualise the on-the-fly status of a simulation. For example, it shows the animation of shipments during a simulation. Besides a normal simulation mode in which a user can start, pause, and stop a simulation, SIMRISK also provides a batch mode for numerical experiments. During the batch mode, all the non-essential status displays are disabled to reduce computational overhead. In the spirit of the tools extensibility, the design of SIMRISK'S graphic user interface also supports the display of information specific to custom-defined element types: SIMRISK provides a graphic handler to a custom-defined type package, and the package can use the handler to display attributes of an element of custom-defined type. Figure 2 shows a snapshot of SIMRISK'S visual IDE. SIMRISK is written in Java.

Figure 1 The object-oriented type system developed by SIMRISK (see online version for colours)

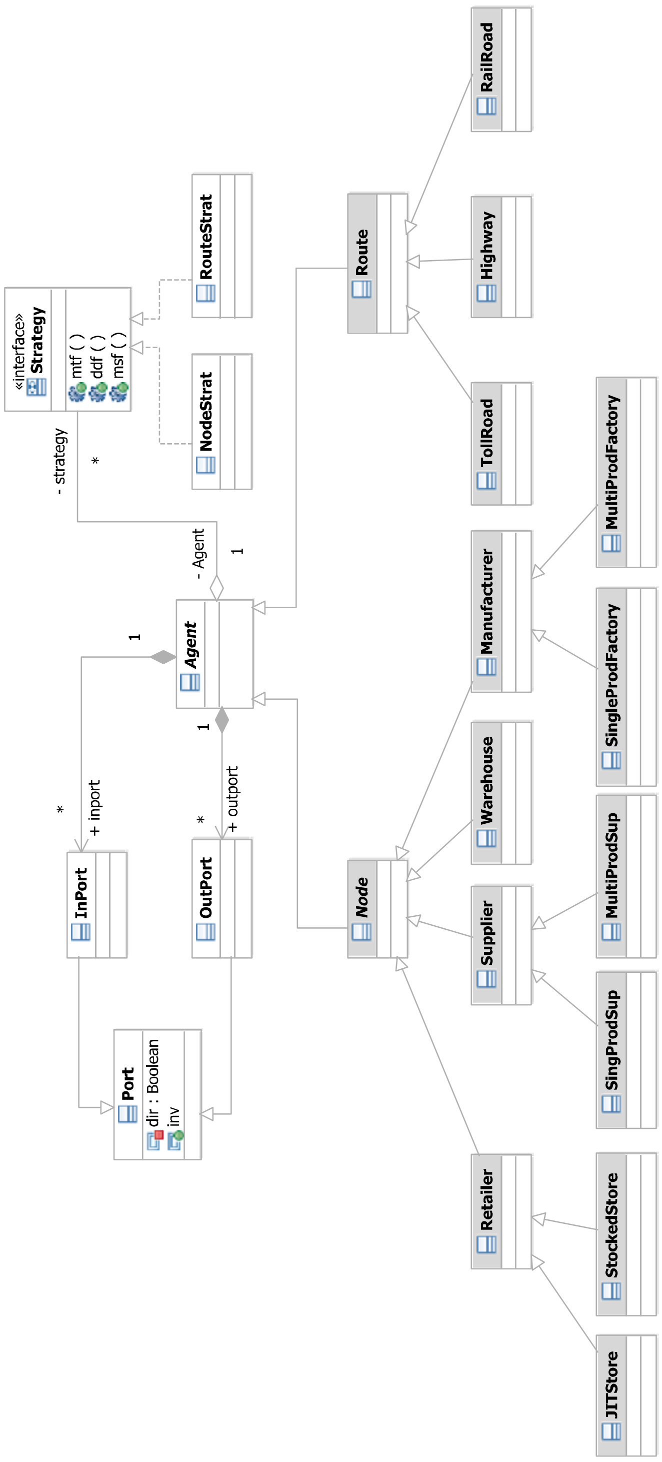
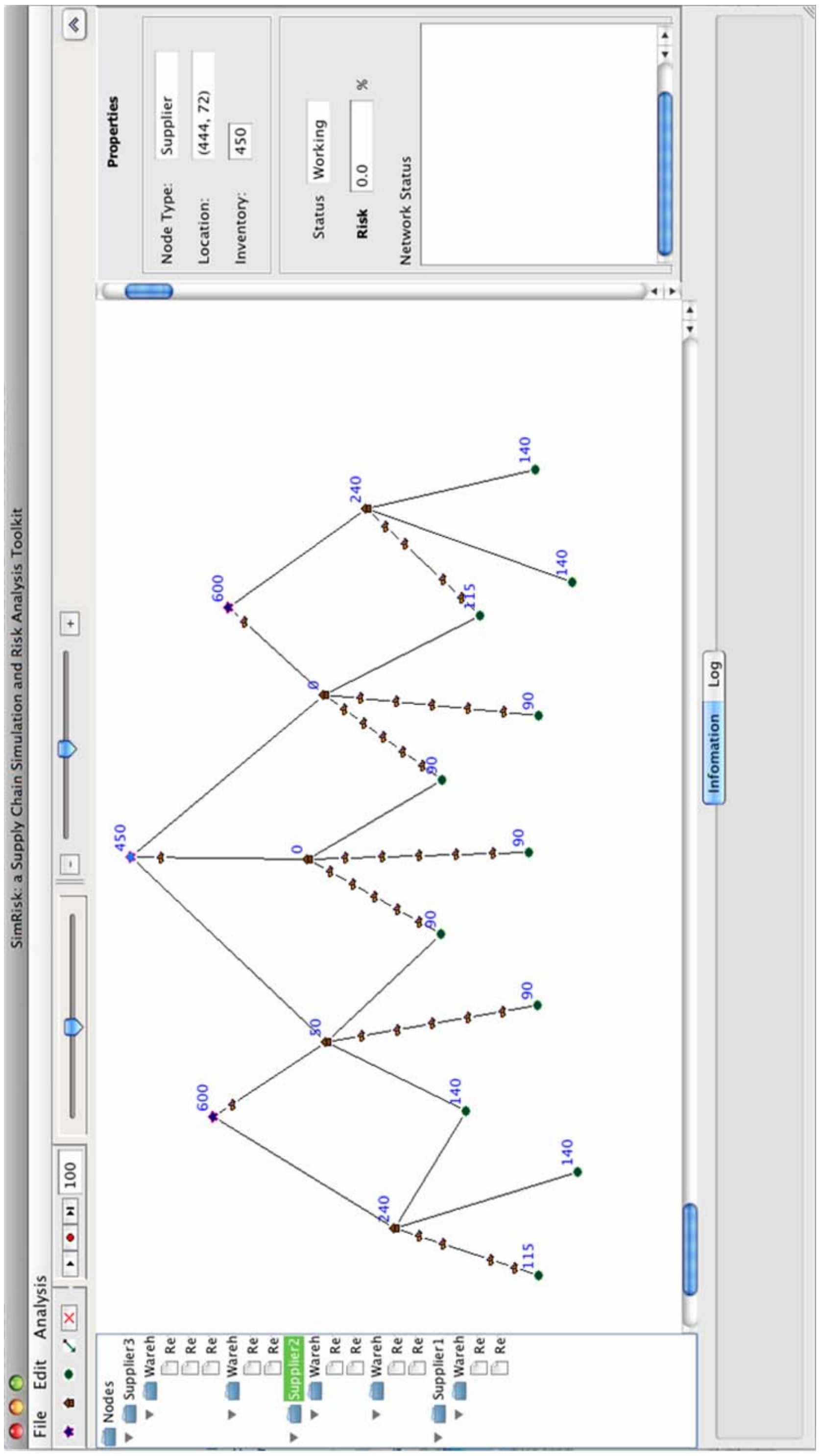


Figure 2 The IDE of SIMRISK (see online version for colours)



Properties

Node Type: Supplier

Location: (444, 72)

Inventory: 450

Status Working

Risk 0.0 %

Network Status

Figure 3 The architecture design of SIMRISK (see online version for colours)

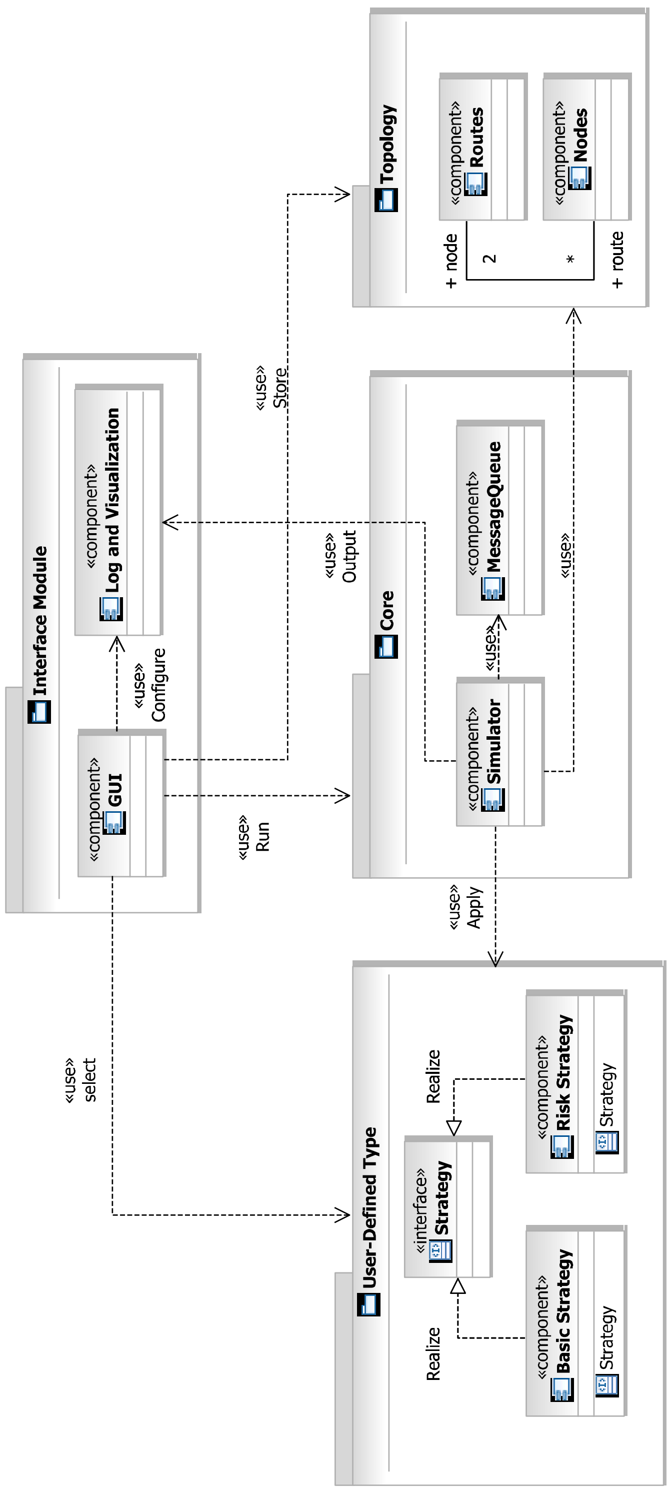


Figure 3 shows the architecture design of SIMRISK, presented as a UML component diagram. To achieve a higher extensibility, a key feature of SIMRISK is to separate the operational semantics of a supply chain from its topology. The requirement for defining the operational semantics of a supply chain is designated by an interface for strategy. An analyst can define his/her own types of elements, as long as she/he supplies the necessary details as required by the strategy interface. The topology package stores the physical structure of a supply chain. It defines, for example, the locations of nodes and how they are connected by routes. In other words, the topology package defines geographic locations of elements and the custom-defined type package specifies their semantics as defined in Section 3. To test different supply-chain policies on the same network structure, an analyst can switch between different strategies on-the-fly. SIMRISK implements an event-driven simulation engine as outlined in Algorithm 1.

7 Conclusions and future works

We proposed an extensible object-oriented and agent-based formal framework for modelling and simulating supply chains. This research work made the following contributions to supply-chain modelling and analysis: first, we developed an agent-based approach that supports custom-defined element types. Both the behaviour and internal structure of a node are customisable; second, to promote the design reusability, the proposed framework incorporates an object-orient type system to simplify the work of defining a new type of element. The type system allows an analyst to inherit common behaviour and structure form existing elements, and to focus on the features unique to the new type of element; third, we introduced a meta-model for supply chains and formally define its semantics. The formalism we introduced defines the interface of an element (Section 3.1), its internal and external behaviours (Section 3.2), and its constraints (Section 3.3); finally, we proposed a discrete-event simulation algorithm. The algorithm also defined the simulation semantics of the meta-model.

There are several directions to extend this research. For instance, multi-core hardware could be an excellent platform for our agent-based framework, with elements running on different cores and communications expedited by shared memory. We plan to study the parallel simulation of the agent-based framework on multi-core hardware. Specifically, we will develop a generative simulation technique that is capable of generating simulation code optimised for a specific multi-core architecture. Another possibility is to provide a tighter integration between simulation-based approach and formal analysis approach. Currently SIMRISK can translate a supply-chain model with elements of built-in types to an extended Markov decision process for formal analysis. In the future, we want to extend formal analysis to supply-chain models with custom-defined element types.

References

Axelrod, R. (1997) *The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration*, Princeton University Press.

Chen, X. and Zhang, J. (2008) 'Supply chain risks analysis by using jump-diffusion model', in *WSC '08: Proceedings of the 40th Conference on Winter Simulation*, pp.638–646, Winter Simulation Conference.

Costco Wholesale Corporation (2007) 'Costco wholesale annual report 2007', Year ended 2 September 2007.

Ferrer, J. and Karlberg, J. (2006) 'Achieving high performance through effective global operations', Technical report, Accenture.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1994) *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley.

Liu, J., Wang, W., Chai, Y. and Liu, Y. (2004) 'Easy-SC: a supply chain simulation tool', in *Proceedings of the 2004 Winter Simulation Conference*, Vol. 2, pp.1373–1378.

Neumann, J.V. (1966) *Theory of Self-Reproducing Automata*, University of Illinois Press, Champaign, IL, USA.

Rossetti, M.D. (2008) 'JSL: an open-source object-oriented framework for discrete-event simulation in Java', *International Journal of Simulation and Process Modeling*, Vol. 4, No. 1, pp.69–87.

Rossetti, M.D., Miman, M. and Varghese, V. (2007) 'An object-oriented framework for simulating supply systems', *Journal of Simulation*, Vol. 2, pp.103–116.

Swaminathan, J., Smith, S. and Sadeh, N. (1998) 'Modeling supply chain dynamics: a multiagent approach', *Decision Sciences*, Vol. 29, No. 3, pp.607–632.

Tan, L. and Xu, S. (2008) 'Model check stochastic supply chains', in *Proceedings of the IEEE 2008 International Conference on Information Reuse and Integration*, IEEE, pp.416–421.

van Delft, C. and Vial, J.P. (2004) 'A practical implementation of stochastic programming: an application to the evaluation of option contracts in supply chains', *Automatica*, Vol. 40, No. 5, pp.743–756.

Wang, W., Dong, J., Ding, H., Ren, C., Qiu, M., Lee, Y. and Cheng, F. (2008) 'An introduction to IBM general business simulation environment', in *Proceedings of the 2008 Winter Simulation Conference*, pp.2700–2707.

Wei, L., Chai, Y., Ren, C. and Dong, J. (2007) 'A research review on dynamic performance analysis of supply chain system', in *Proceedings of the 2006 Asia Simulation Conference on Systems Modeling and Simulation Theory and Applications*, Springer, Japan, pp.163–167.