Detecting Criminal Disruption of Supply Chains Study: Report: General Methodology and Tools

CARLOTTA DOMENICONI, ABHISEKH RANA, RAJ PATEL, JAMES JONES, HAMDI KAVAK, SEAN LUKE, AND VLADIMIR MENKOV

George Mason University

FRED ROBERTS, ALOK BAVEJA, WEIWEI CHEN, DENNIS EGAN, AMAN GOSWAMI, RONG LEI, PETER MARSH, BENJAMIN MELAMED, AND VISWANATH NARAYAN

Rutgers University

ANDREW CROOKS

University at Buffalo

August 21, 2023

1 Introduction

This report details the methodology used to develop, run, optimize, and analyze supply chain models created for this project. It also discusses the software tools either used, or developed and extended, for the development of the models and implementation of the methodolgy. The project modeled supply chains from three different industries, a generic pharmaceutical company (SC-1), a generic company producing two interrelated medical device products (SC-2), and a company producing solar arrays for communication satellites in the United States (SC-3). The example supply chains are shown in Figures 1, 2, and 3.

The system has three major elements: constructing supply chain network maps/models; developing and running discrete event simulation models, and performing dynamic optimization of parameters which guide criminal disruptions in order to find worst-case disruptions to study. To produce the supply chain maps/models, we developed a Focal-core Supply Chain Methodology (FSCM) that models a focal core firm's supply chain in detail and a larger "external" environment in less detail. The FSCM aids in identifying potential disruptions and in identifying potential mitigations for analysis. To develop the simulation models we extended the MASON [1] agent-based modeling toolkit with an entirely new, extensive, discrete event

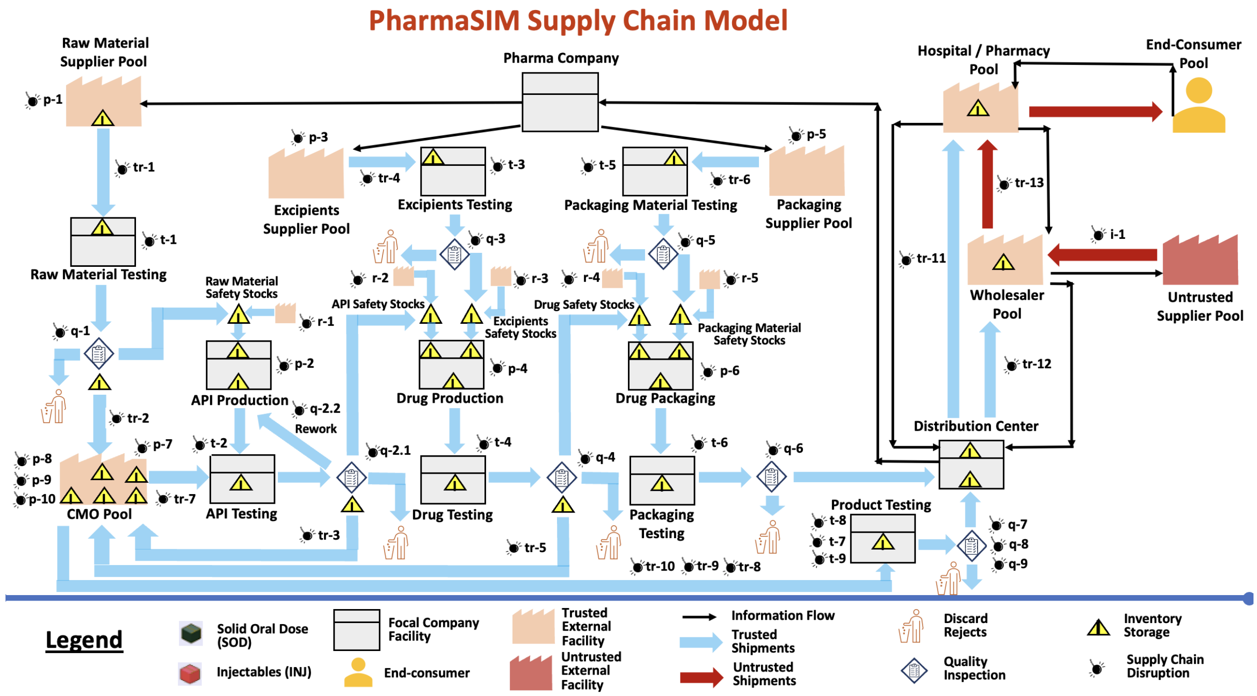


Figure 1: SC-1: A Generic Pharmaceutical Supply Chain Model.

simulation library of our own design. This library is open source and now part of the MASON distribution. To optimize the models based on criminal organization capabilities, we married MASON with the ECJ [2] distributed evolutionary computation and stochastic optimization system, customized to optimize our model parameters of interest.

We used the FSCM-produced models/maps, MASON, ECJ, and our new MASON Discrete Event Simulation (DES) library to develop baseline understanding of each supply chain, and to study the impact of a variety of supply chain disruptions and the efficacy of various mitigations suggested by the FSCM map. In each subsequent case, we learned from the earlier supply chain modeling, identification of disruptions and mitigations, and the simulation experiments. Based on these three case studies, we have developed a general methodology for understanding threats and vulnerabilities of various supply chains, indicators and warnings to alert that a supply chain disruption has taken place, is taking place, or is planned, and efficacious mitigation strategies. The general methodology is described next.

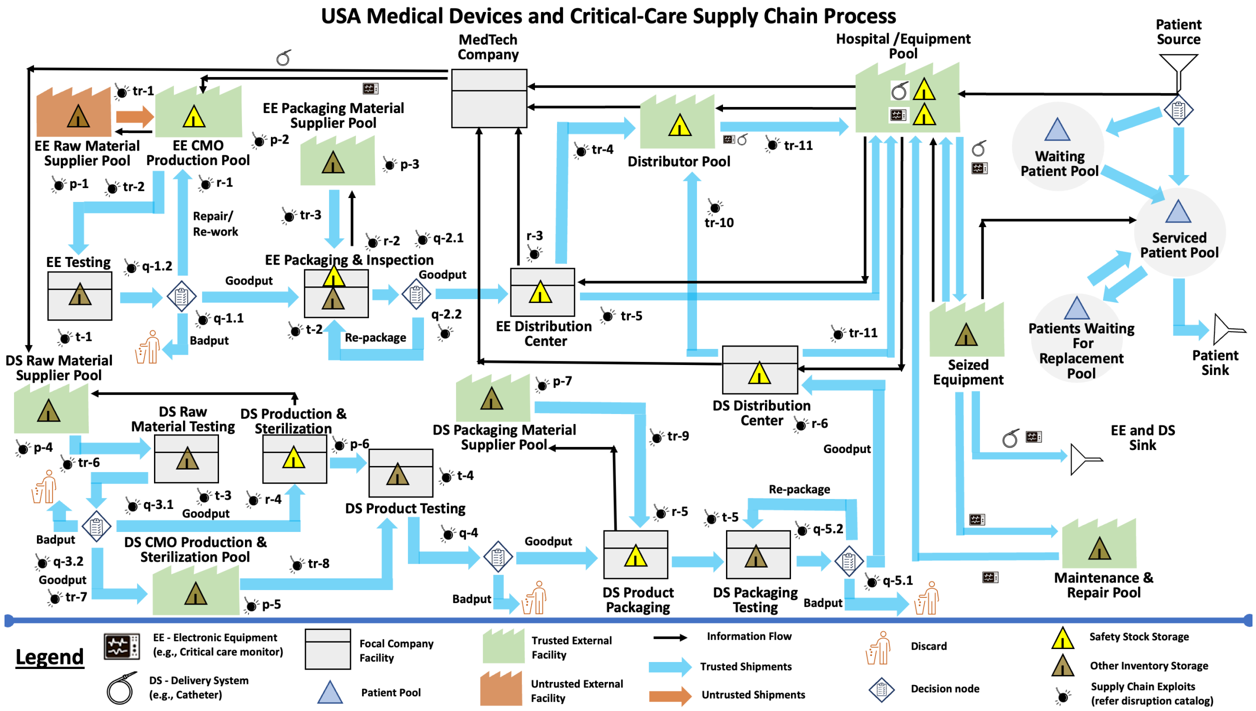


Figure 2: SC-2: Model Topography of MedTech SC.

2 Supply Chain Mapping and Modeling

Modern supply chains are flow networks in which material and information move among facilities, often globally. Many supply chains are complex. Their complexity stems from their size (number of nodes and links in their flow networks), layout (global reach, often spread across continents), and the functional heterogeneity of their nodes (suppliers, customers, facilities, etc.) At best, this complexity makes it challenging to model and analyze, and at worst, makes it impossible when information is unreliable or unavailable.

Recognizing the aforesaid difficulties, we developed a modeling approach for complex supply chains, dubbed the Focal-core Supply Chain Methodology (FSCM), to effectively address the complexity challenge, when modeling supply chains and analyzing the impacts of disruptions. Supply chains SC-1, SC-2, and SC-3 were created using this methodology as it was developed. Broadly speaking, FSCM partitions the supply chain into two components. The first component is a subset of nodes and links of primary interest to decision makers, to be referred to as the focal core. A focal core typically consists of a focal supply chain firm of interest plus a few of its supplier and customer tiers. The second component encompasses supply chain nodes outside the focal core, to be referred to as the focal core's environment (environment, for short). The environment consists of higher-tier suppliers and customers, not included in the focal core.

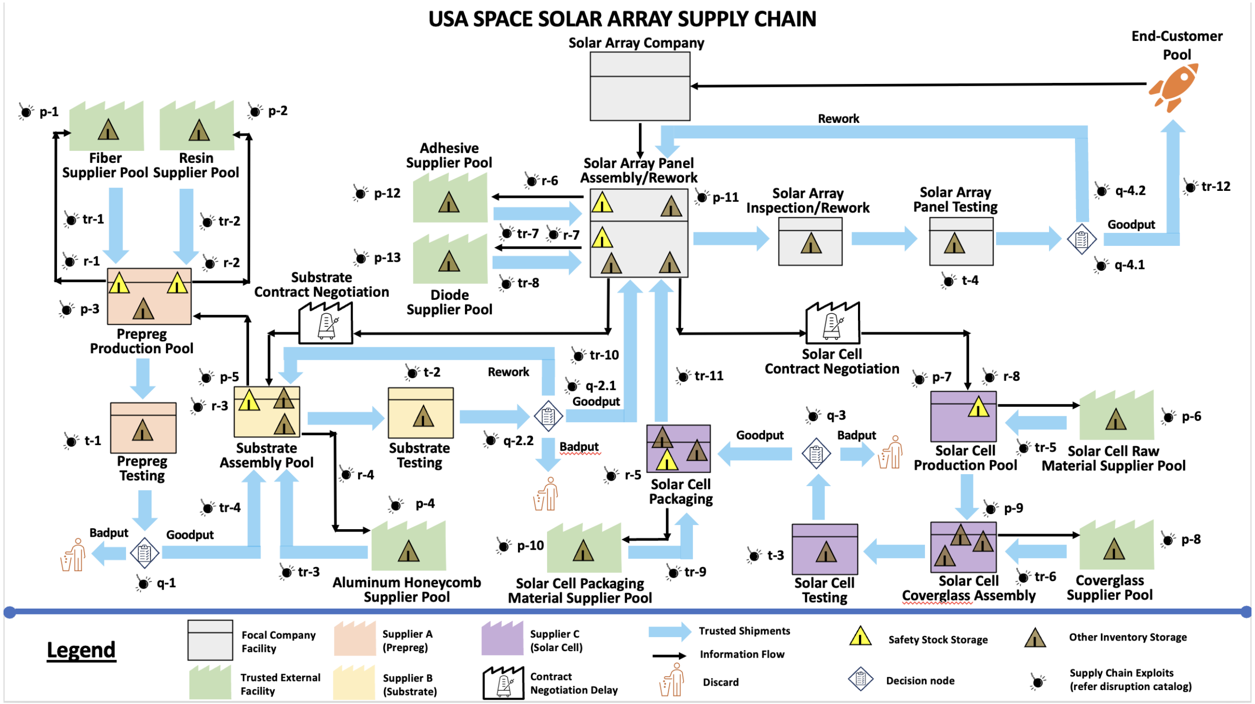


Figure 3: SC-3: USA Space Solar Array Supply Chain .

The FSCM approach is to model the focal core in considerable detail and its environment in much lesser detail. Specifically, the environment is modeled as an extra single tier of nodes, directly linked to the focal core. This extra tier is created by an extensive aggregation of the real-life supply chain in two ways:

1. Aggregation that combines like-nodes into a single pool node. For example, multiple suppliers of the same material are modeled by a single supplier pool. Accordingly, in Figure 2, an EE Packaging Supplier Pool and a DS Packaging Supplier Pool are aggregated nodes.

2. Aggregation by combining the entire supply chain upstream that lies in the environment into a single node. For example, a single supplier node, usually a supplier pool, represents all supplier tiers upstream from it. Downstream distribution tiers to end-customers are handled similarly. Accordingly, in Figure 2, EE and DS Raw Material Supplier Pools are similarly aggregated nodes.

The selection of the focal-core component is a modeling decision driven by two considerations. The first consideration is the availability of information, and the second is the size of the resultant model.

Typically, information about the supply chain under study is obtained from current or past employees of a supply chain company. Such information includes key parameters such as travel times, testing delays, sizes of inventories etc., and these parameters then drive simulations of supply chain performance under normal (baseline) or disrupted conditions. Often employees who aid in supply chain construction and parametrization are only familiar with the structure and data of a subset of the broader supply chain in which their focal firm is embedded, and information about the environment beyond that subset is unavailable. Thus, that subset, minus the farthest upstream and downstream tiers of the focal firm, is a natural choice of the focal core component of the model, with the tiers farthest from it becoming the focal core's environment.

The second consideration regarding size of the resultant model is standard in modeling a system. To gain modeling clarity and actionable insights, the model is run and computes performance metrics, depending on the model's type (e.g., analytical, simulation, statistical analysis, etc.). But to be feasible and timely, such computations, and indeed the model representation itself, must have sufficient availability of memory and time computing resources. Thus, the supply chain model's size must be limited to obtain effective and timely computational results, which is a factor in choosing the model's focal core and its environment.

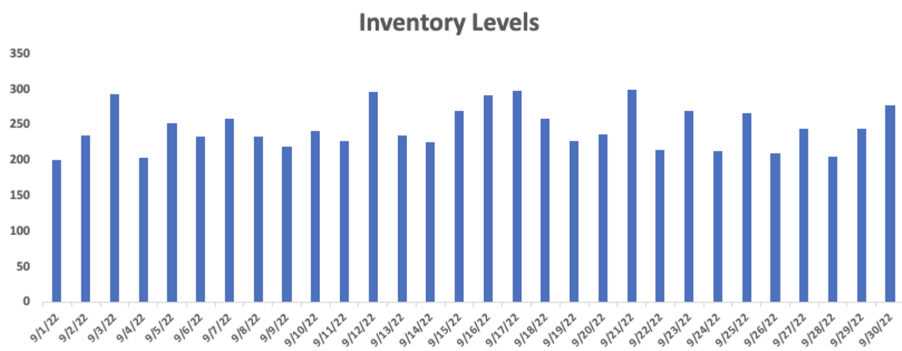
The following is an outline of the FSCM's steps for modeling a supply chain:

1. Assemble an advisory group of experts that can provide the information on the supply chain's model structure and data for model parameters. All subsequent steps are carried out in collaboration with this group.

2. Identify the focal core and environment of the supply chain model and the types of entities flowing through it (e.g., material batches, product assemblies, sub-assemblies, and information.)

3. Create a supply chain map (diagram), showing the supply chain layout, encompassing all nodes and directional links among them. Supply chain map nodes typically represent supply chain fixed assets (e.g., production, testing, and warehousing facilities), while links represent transportation routes and information flows. For each node and link, determine the measurement units of model parameters , for example, in pounds, hours, or as a probability.

4. Based on literature research and consultation with security experts from the advisory group of experts, compile a preliminary list of disruptions to be modeled by location (node or link of possible occurrence) and impact type (e.g., production impairment, unscheduled transportation delays, quality problems affecting rejects, etc.), and identify the model parameter impacted by each disruption (e.g., mean unit production time, mean transportation time, probability of rejects, mean replenishment time of safety stocks, probability of illicit units in inventory and shipments). Note that for a logistics simulation, disruptions are classified by effects, and not by causes. Table 1 lists the modeled disruptions, classified by type.



Mean	246.13
Standard Deviation	30.30
Coefficient of Variation	0.12

Figure 4: Example: Histogram and summary statistics (mean, standard deviation, coefficient of variation) for a random process (inventory levels) .

5. Overlay all possible disruptions on the supply chain map, by placing an icon and a disruption code at each location where the disruption can occur, as shown in SC-1, SC-2, and SC-3. The supply chain map facilitates the editing of the preliminary list of disruptions from the previous step.

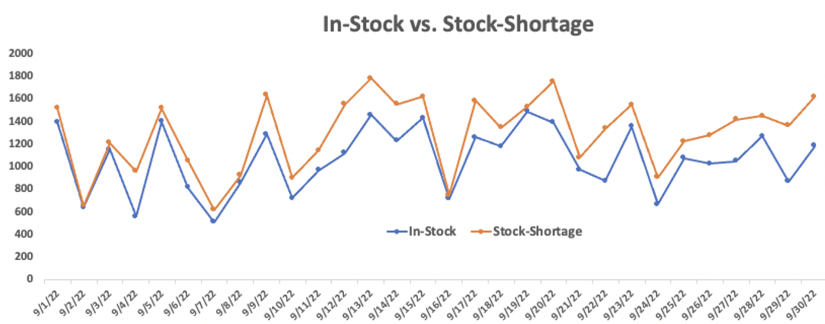
6. Using interview sessions with members of the advisory group of experts, parameterize the model with data that correspond to a baseline supply chain model in approximate equilibrium (a long-term regime, reached in the absence of disruptions). Typical data include delay distributions at supply chain nodes (e.g., unit production delay, shipment travel time along links), probability of rejected units after a testing step, replenishment lead times of stocks, probabilities of routing units to particular links, etc. In practice, a running simulation model is considered to have entered an equilibrium regime when key statistics (e.g., averages and time averages) stabilize over time. Before any simulation experiments are performed, the model should be run for a so-called "warm-up period" to bring it closer to equilibrium (stable baseline operation). Since, in practice, supply chains constantly change, the data and its analyses should be updated regularly.

7. Define model-generated statistics that serve as performance metrics of interest to decision makers, or for gaining insights into supply chain operations. For example, in a simulation, collect data from random processes, such as time series and histograms of inventory levels, lead times of shipments, and inventory stock-out durations, as well as their summary statistics (means, variances, coefficients of variation, etc.) Figures 4, 5 and 6 illustrate some typical performance metrics germane to disruptions.

8. Create the supply chain model and run it in baseline mode, and separately, in disrupted mode by injecting one or more disruptions into the baseline model. Measure performance metrics of interest

Mean	90.40
Standard Deviation	18.57
Coefficient of Variation	0.21

Figure 5: Example: Time Series and summary statistics (mean, standard deviation, coefficient of variation) for a random process (supplier lead times).



Parameter	In-Stock	Stock-Shortage
Mean	1064.40	230.07
Standard Deviation	288.69	140.06
Coefficient of Variation	0.27	0.61

Figure 6: Example: Time Series and summary statistics (mean, standard deviation, coefficient of variation) for two random processes (in-stock and stock-shortage).

identified in step (7), including disruption-related ones, such as the fraction of time during which the inventory stocked out and the time to recover from disruptions. For example, if a disruption lowers an inventory level below a baseline level or depletes it, measure the time it takes the inventory to return to that baseline level for the first time thereafter. Figure 7 illustrates an example of the resilience triangle - a metric of interest which is used to evaluate the performance of a process under duress.

9. Run scenario experiments using simulation runs to evaluate the severity of a variety of disruptions based on their computed disruption-related metrics. Identify the most vulnerable points in the supply chain and devise robust strategies to mitigate such vulnerabilities, based on the simulations and/or the expert opinion of the advisory group, and/or the open literature. Key mitigations can either be pre-disruption or post-disruption and include increasing the levels of safety stocks at production nodes to sustain downstream flows longer, by increasing the reorder points and target levels of Make-

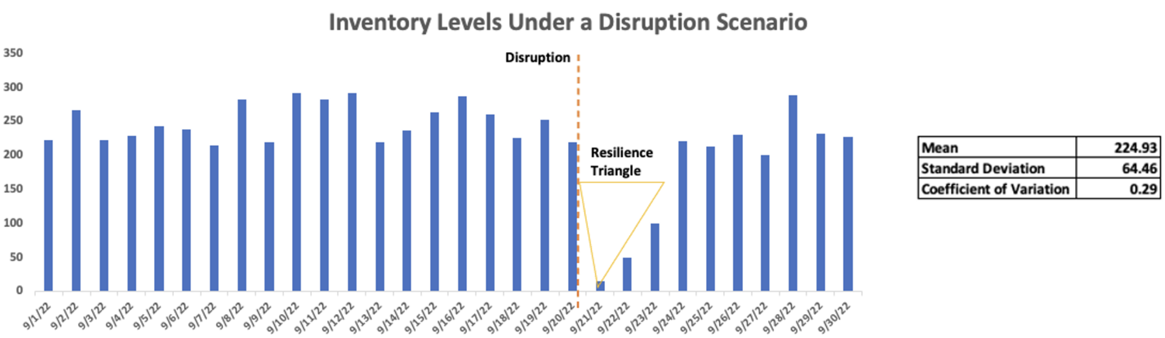


Figure 7: Example: Resilience Triangle indicating time taken to recover after a disruption to a random process (inventory levels). The triangle's depth indicates the severity of disruption, the length indicates recovery time, and the area indicates the resilience of the supply chain – the smaller the triangle's area, the higher the resilience of the supply chain.

to-Stock inventories, and by securing stand-by manufacturing capacity via outsourcing to contact manufacturing organizations (CMO pools).

10. Run scenario experiments using simulation runs on the model with a variety of mitigations, and assess their efficacy.

3 Foundational Tools

Our development of the supply chain models, and their optimization, rested on several foundational software tools developed at George Mason University. These tools are all free open source using the AFL 3.0 or Apache 2.0 academic (non-viral) licenses. The tools include:

- **MASON** An agent-based modeling simulation toolkit designed for efficiency and ease of modification. Written in Java.
- **MASON DES Extension** A software tool we built which extends MASON to provide Discrete Event Simulation facilities. These facilities can interoperate with MASON agent-based models (and indeed are implemented internally using them). Written in Java, this tool was specifically developed for this project.
- **ECJ** A high performance massively distributed optimization toolkit employing a variety of evolutionary algorithms. Like MASON, designed for efficiency and ease of modification. Written in Java. We

Disruption Impact Type	Disruption Description	Disruption Location	Parameter Impacted
Disrupted Production	Disruptions that increase the production time of a product unit, thereby lowering the production capacity	(Grey) Production Nodes	Mean batch production delay
Disrupted Testing	Disruptions that increase the testing time of a product unit, thereby lowering the testing capacity	(Grey) Testing Nodes	Mean batch testing delay
Disrupted Transportation	Disruptions that increase the transportation time of a product unit, thereby lowering the transportation capacity	(Blue) Transportation Arrows	Mean batch transportation delay
Disrupted Replenishment Lead Time	Disruptions that increase the lead time of replenishment of a product unit	(Yellow) Triangle Icons	Mean replenishment lead time
Disrupted Production Quality	Disruptions that decrease the probability of goodput at a testing node, thereby lowering the product throughput	(Diamond) Quality Control Icons	Mean percentage of goodput

Table 1: Disruption List by Types

have also developed tools to connect ECJ and MASON in an easy to understand and easy to develop fashion.

3.1 MASON

MASON [1] is a popular open-source Java agent-based modeling toolkit in active development since 2003. It is designed to be fast, hackable, modifiable, and easily integrated with other tools. MASON has been used very widely over the last twenty years by many modelers and institutions.

MASON carefully delineates between the model and its visualization, and while this is common in the simulation field at large, it is unusual among agent-based modeling toolkits. A MASON model is encapsulated in a singleton object called a *SimState* and knows nothing of visualization or inspection. To visualize, inspect, and graphically manipulate the model, the modeler builds a separate *GUIState* which references the SimState objects and displays their visualization. This allows MASON models to run without any visualization, then have different kinds of visualization attached dynamically. It also permits MASON models to be frozen mid-run, serialized to disk or transferred to another computer or environment, and resumed there. MASON models are entirely self-contained. They can be run in parallel in separate threads or even in the same thread in a process: a MASON model can even contain MASON models running recursively inside it. For example, agents might make decisions in a model by running their internal simulations.

MASON models contain two primary components. First, a model holds a representation of time in the form of a heap-based schedule. Agents are scheduled to be woken up at some real-valued time in the future in order to perform some action of their choosing. Agents can then reschedule themselves on the schedule. Second, the model contains one or more representations of space called *fields*. The modeler may provide his own custom fields, but many common ones are available, including network and graph structures, many kinds of grids, continuous space in bounded, unbounded, or toroidal 2D or 3D space, and GIS environments via the *GeoMASON* extension [3]. For each field provided, MASON also provides an accompanying set of 2D and 3D visualization and inspection facilities. MASON has many additional features, including parameter sweeping, publication-quality charts and graphs, and integration with massively distributed model optimization tools. Finally, for larger models, *Distributed MASON* and *Distributed GeoMASON* can be used to distribute a model over a large number of processors in a cloud computing or server farm environment.

Important Links for MASON

- MASON and Distributed MASON
  https://cs.gmu.edu/∼eclab/projects/mason/
- GeoMASON and Distributed GeoMASON
  https://cs.gmu.edu/∼eclab/projects/mason/extensions/geomason/
- MASON Version 21 Manual (including MASON, GeoMASON, Distributed MASON, and Distributed GeoMASON)
  https://cs.gmu.edu/∼eclab/projects/mason/manual.21.pdf

3.2 MASON DES Facility

The MASON DES extension [4] aspires to provide an alternative to the existing hybrid ABM-DES simulation software by implementing DES facilities on top of MASON's ABM engine and fully integrating them with it. The DES extension is intended to be efficient, easily modifiable, and extensible, offering the same characteristics as MASON itself. The MASON DES extension has been designed to support large-scale simulations and is currently used in a major research and industrial use case.

The DES system essentially consists of a graph structure of *processes*, any of which may, in response to some *event*, communicate with one or more other processes primarily by offering *resources* to them or requesting the same. Resources are used both to transfer information or ownership from one process to another and also serve as the elements seized and released from common resource pools to implement semaphores. Resources are typed, and the modeler may define additional types, but they are grouped into three major categories. First, there are *countable* resources, such as money or bricks. Second, there are *uncountable* resources, such as gasoline or happiness. Third, there are basic tokens called *entities*, which can also contain collections of resources and a manifest, and so might additionally be used to model composite things like cargo containers of various resources.

The DES system defines a non-blocking push-style flow graph with optional pull. That is, during an event, a process may offer to transfer one or more resources to a downstream process. The downstream process must immediately accept or refuse some or all of them. A downstream process can optionally request that an upstream process offer it some range of resources, and the upstream process may similarly refuse.

Processes may take several forms. *Sources* only offer resources, *sinks* only receive resources, and *middlemen* both offer and receive resources. Many provided middlemen, called *filters*, are designed to transform or act on a received resource and immediately offer it or some transformed version of it to other processes. Most processes can only receive a single resource type and offer a single type, but so-called *multi* processes are provided to receive and offer many types. For example, a factory might be implemented as a multi-process that accepts steel, electricity, and tires and produces bicycles and waste. Modelers can choose to use provided process classes or to subclass and customize any of them as they prefer.

Every process is implemented as a MASON agent: it can be placed on MASON's schedule to be woken up at a specific time (its event) to do work. However, only certain processes, such as *delays* or resource generators (*sources*), need to be placed on the schedule. Instead, many processes are designed to act only when offered resources. For example, when a *queue* — essentially a warehouse — receives an offer of resources, it stores them in the queue if it has enough capacity. Then it immediately offers resources from the queue to select downstream processes. While a queue can also be scheduled to offer to downstream processes at specific times, like every three hours, by default it only offers them when it has received new resources.

MASON also offers *macros*: process-like objects that encapsulate subgraphs of other processes. For example, the graph defining a university computer network may be encapsulated into a macro, then placed in a higher-level graph of abstracted institutional networks. Macros are fully recursive.

A MASON DES extension model is built exactly like a MASON ABM model: it uses the same SimState and separation between model and visualization. The DES extension offers rudimentary debugging, inspection and probing, and visualization, which we intend to improve over time. Moreover, it inherits MASON's checkpointing and other capabilities, including integration with external tools such as ECJ [2]. The primary MASON feature not available to the DES system is the ability to be distributed over multiple processors in Distributed MASON. However, a Distributed MASON model can have one or more DES graphs on each processor or contained within agents migrating through the model.

Important Links for the MASON DES Facility

- DES Facility (presently on MASON's development branch, but will soon move to its primary branch)
  https://github.com/eclab/mason/tree/dev/contrib/des

- DES Facility Manual
  https://github.com/eclab/mason/blob/dev/contrib/des/docs/manual.pdf

- DES Supply Chain Modelling
  https://github.com/vmenkov/DES-Supply-Chain-demo

3.3 ECJ

In order to perform optimization of criminal behavior (aimed at identifying "worst case" threats), we employ a variety of techniques drawn from the general family of *evolutionary algorithms*, which are members of a field known as *evolutionary computation*. An evolutionary algorithm is a kind of stochastic or black-box optimization algorithm which works as follows. We begin with an initial set of randomly-generated candidate solutions to test. We then iteravely test them, assessing their objective quality, which we call their *fitness*. We then resample a new set of candidate solutions by selecting from the previous solutions biased randomly by fitness, copying them to form *children*, and modifying the children in slightly random ways. We then test the new candidate solutions and repeat the iteration.

ECJ is a free open source evolutionary computation library written in pure Java. It has support for a very large number of evolutionary algorithms in both single-machine and a variety of massively parallel configurations. Like MASON, ECJ was designed to be highly efficient, and easily modified and extended. ECJ is organized as a set of modules which are connected, parameterized, and specified via an experimenter-provided collection of parameter text files.

In order to optimize candidate solutions, ECJ must test each candidate solution in some environment to assess its fitness. To do this it requires that the experimenter develop a platform for testing solutions, which it will then call, providing the candidate solution or solutions, and then receiving the results.

ECJ-MASON Optimization ECJ is agnostic as to the nature of its assessment platform. However MASON was specifically designed to make it relatively easy to interoperate with ECJ. To assess one or more candidate solutions, ECJ will build a MASON model, provide the candidate solutions to MASON to embed them in the model as appropriate, then run the model. When the model has terminated, ECJ will then query MASON as to the fitness of the solutions tested in the model, then take down the model.

MASON has a variety of hooks to ease development of the glue code to connect MASON and ECJ: but it is still a complex task as it requires the software developer have intimate understanding of how MASON and ECJ both work. To simplify this task further, we have developed a set of classes which automatically connect ECJ and MASON for the most common user cases. This facility is MASON's Optimization Facility.

Using this facility, the software developer only needs to develop the MASON (and DES) model — which presumably he has already done — then specify the MASON model parameters to be optimized, the evolutionary algorithm to use, and a few ECJ parameters. He must also provide a simple function at the end which assesses the fitness of the resulting model (and thus the candidate solution's model parameters). ECJ automatically sets up a distributed evolutionary architecture with multiple machines, runs the algorithm, executes the MASON models with the modified model parameters, and extracts the fitness.

Important Links for ECJ

- ECJ
  https://cs.gmu.edu/∼eclab/projects/ecj/
- The ECJ Owner's Manual
  https://cs.gmu.edu/∼eclab/projects/ecj/manual.pdf
- ECJ MASON Optimization Facility
  https://github.com/eclab/mason/tree/master/contrib/optimize

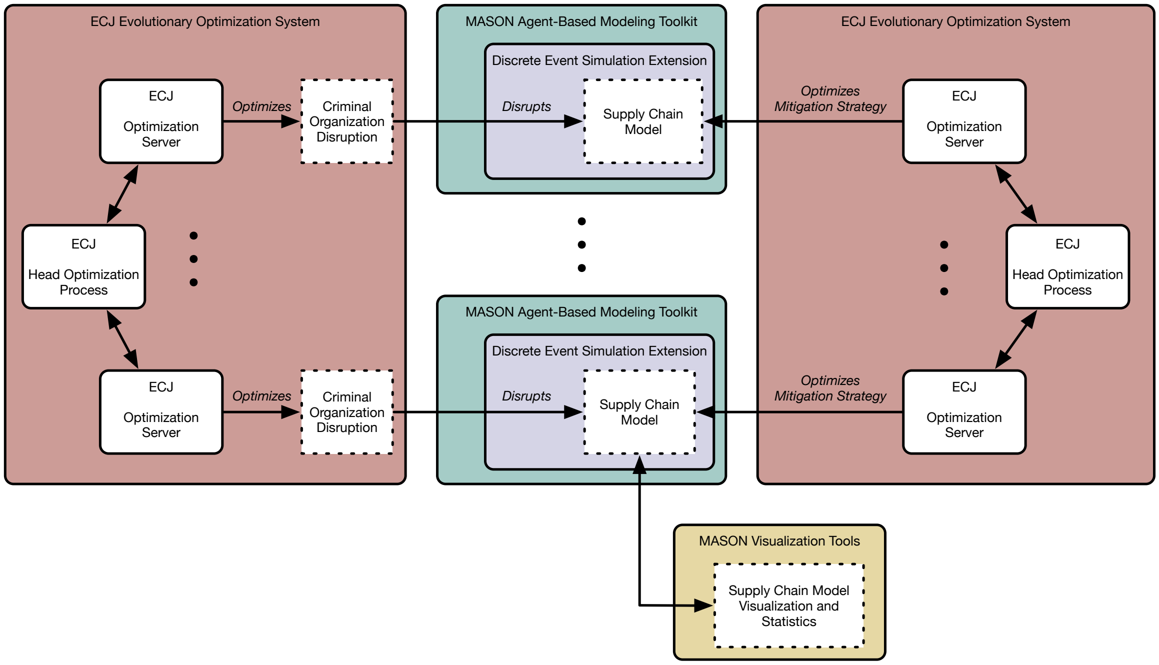


Figure 8: Overview of the Interaction between ECJ, MASON, and the Model. ECJ is used both as a potentially massively distributed optimization framework (left) to search for potential disruptions to the model using parallel supply chain model instances running on separate MASON processes (center), and is also used to optimize possible mitigation strategies (right). Models run in a custom Discrete Event Simulation (DES) environment inside MASON, and can be visualized using specialized visualization tools.

- ECJ MASON Optimization Facility Manual

  Section 13.2 of the MASON 21 Manual, at https://cs.gmu.edu/∼eclab/projects/mason/manual.21.pdf

4 Building a Supply Chain Model with the MASON DES Extension

MASON is an agent-based modeling toolkit at heart: but we have extended it to provide discrete event simulation (DES) instead, as discussed in Section 3.2, then combined it with ECJ to optimize discrete event simulation parameters relating to criminal network operations. This toolkit allows us not only to easily build DES models in MASON, but to combine them with future agent-based models in the same simulation. Figure 8 gives an overview of our system.

In a DES model, various processes provide resources to other processes in real time: for example, in SC-2, the EE Distribution Center may provide EE to the Hospital/Equipment Pool. In an agent-based model, the processes (now called *agents*) have much more flexibility to move about and interact with one another: for example, various gangs may organize themselves dynamically in order to disrupt a supply chain in real-time.

Because we implement disruptions directly and optimize them with ECJ, we have not needed to take full advantage of MASON's agent-based modeling features, and have focused on the DES extension. However these features are available for future extensions to the model which use agent-based modeling more fully.

The supply chain map/model constructed using the FSCM methodology is implemented using a number of Java classes from the MASON DES simulation toolkit as its building blocks. Notably, we have used the `Queue` class to model locations where products are stored (pools and intermediate storage location between processing stages), and the `Delay` and `SimpleDelay` classes for modeling production, transportation, and testing (QA) stages of the supply chain. The model implementation is fully configuration controlled, with a configuration file or a corresponding Java object providing the information about the business rules controlling the system.

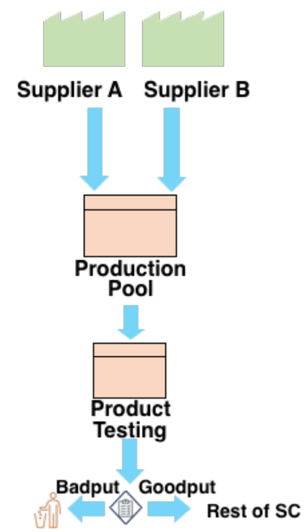


Figure 9: Toy Supply Chain

To illustrate usage of DES for supply chains, we provide a toy example here. Assuming we want to build a product that has two components, *A* and *B*. The supply chain starts two suppliers, implemented using the `Source` class. These are then sent to a production node, implemented using the `Receiver` (handling input of the resources) and `Provider` classes (handling output of the assembled product). These need to be tested, which is accomplished using a testing node, built using the `Delay` class to capture the time it takes to test the recieved product. An illustratioin of the toy supply chain is shown in Figure 9.

4.1 Exploits and Disruptions

A simulation in our MASON-based application can be run jointly with a *disruption scenario* object, which specifies the list of disruptions that can occur during the simulation. Each disruption is characterized by its type, location, starting time, and magnitude. The type, at present, can be one of the following: `Halt` (stopping the operation of a certain production nodes or information flow between nodes for a certain amount of time), `ShipmentLoss` (loss of the shipments sent between certain supply chain nodes during a certain period of time), `Adulteration` (an increase in the probability that the product batches produced on a certain day at a certain production node will be faulty), or `Depletion` (the destruction of certain amount of the product stored at a specified location). The location specifies the supply chain node affected by the disruption, e.g. the EE Raw Material Supplier Pool or the DS Production and Sterilizaton Pool in SC-2. The meaning of the magnitude is type-specific: for halts and shipment losses it represents the number of days the disruption lasts; for adulterations, an increase in failure rate; and for depletions, the amount of the product destroyed.

Once the disruption scenario has been loaded into the system, it affects the behavior of the targeted supply chain nodes accordingly, e.g. in SC-2, the DS Production and Sterilization node will shut down for a while, or will produce more bad product batches.

4.2 Mitigations

At present, the main mitigation tools available to our supply chain model involve the maintenance of adequate levels of extra inventory (safety stocks) and access to backup suppliers and CMOs in case of procurement or production disruptions. Other mitigations (e.g., adding more sensors), are also studied through a variety of changes such as modification of parameters.

Ordering stock via backup suppliers In the normal (baseline) simulation in SC-2, for example, the focal company receives orders for the EE and DS from the Hospital Pool. It then orders the amount of materials required to fulfill these orders from its regular suppliers. In the case of a disruption, or for any other reason, where the materials delivered by the regular supplier cannot meet the required demand, the focal company has the option to order from a backup supplier. This backup supplier is considered to be local, and thus more expensive, but able to fulfill the shortfall in required materials at a faster rate. The way this mechanism works is that at each node in the production chain the company has a target level for materials required to meet the production demand. If for any reason the target level is not met and thus there is insufficient inventory to meet demand, it orders the shortfall in materials from the backup supplier. We carry out experiments with and without these backup suppliers in our model in order to measure their efficacy at ameliorating the damage from disruptions to the supply chain.

Outsourcing production to backup CMOs Similar to the backup suppliers the focal company also has access to backup CMOs to which it can outsource production in case of a disruption to its own production facilities. The backup CMOs are also considered to be local, however they have a lower production capacity than the focal company's own production facilities. A key difference is that these backup CMOs can only be utilized if the focal company is able to detect that a disruption is in effect at one of its production facilities. Thus, a larger detection delay will lead to a delay in the outsourcing of production and will give rise to a correspondingly larger disruption. We carry out different experiments with these backup CMOs in place and not in place, and we also experiment with different detection delay times, which model the impact of inaccurate or missing sensors at these production facilities.

Safety Stock Inventories The focal company also maintains extra inventory (safety stocks) of the finished products. In SC-2, for example, these are EE and DS products at the EE DC (Distribution Center), DS DC, EE DP (Distributor Pool), and DS DP nodes. The Hospital Pool also maintains its own inventory of safety stocks. These safety stocks allow the supply chain to protect itself from disruptions and to also meet sudden temporary surges in demand. These levels are maintained by having a target level for products at each of these nodes that are larger than the actual demand. We carry out experiments with different target levels at these nodes. We optimize the target levels to identify the minimum values necessary to mitigate worst-case disruption scenarios. This process leads to a robust cost-effective solution to protect the supply-chain against performance degradation.

4.3 Optimization of Criminal Agent Disruptions and Mitigations

In our approach, a criminal agent has some set of *resources,* appropriate to the type of agent, with which it may disrupt the supply chain. The question for the criminal agent is how to maximally disrupt the supply chain with the resources available to it. This is an optimization question, and to solve it we apply a stochastic optimization method. We start with some (N) randomly-generated disruption approaches and test each one for efficacy in disrupting the supply chain model. Once we have assessed them all, we then use the optimization algorithm to produce a new generation of (N) more disruption approaches (the "children"). This is done by sampling from the space in the region of the original approaches (the "parents"), tending to sample more in the vicinity of the more successful ones. We then test the children for efficacy, and repeat. With time, each next generation improves until it reaches a global or local maximum of disruption ability.

There are many stochastic optimization algorithms we might apply: we have chosen to start with the Covariance Matrix Adaptation Evolutionary Strategy (or CMA-ES), a well regarded and established technique. The optimization software we use is ECJ, a popular stochastic optimization library for which MASON was designed to dovetail: indeed, as described in Subsection 3.3 we have developed special open source tools which make it easy to marry ECJ and MASON. ECJ has facilities for CMA-ES and many other stochastic optimization algorithms, and like MASON it is easy to extend and modify.

To test each disruption approach we must run a separate supply chain model, which may take a few seconds: but, as needed for further efficiency, these can be run in parallel on many background MASON simulations using distributed evaluation in ECJ. Specifically, a master ECJ process builds the candidate disruption techniques ("disruption scenarios"), then ships them off to remote ECJ server processes for assessment. Each ECJ server process generates a supply chain model on its own MASON simulation, applies the disruption, runs the model, and assesses its performance, then reports back to the master ECJ process. This can scale to very large (many thousands) of machines in parallel as needed.

Once we have determined an optimal disruption approach, we then ask what techniques can be used to mitigate this disruption. The experimenter may set up mitigation methods manually, and may also benefit from applying an optimization algorithm to find the settings of a mitigation method which lead to optimally recover from disruption. To do this, we can again apply a stochastic optimization algorithm, and ECJ, to search for these settings. ECJ would repeatedly set up each MASON supply-chain model as before, but now it will configure each with the mitigation settings to test rather than the disruption as before: the disruption now will be hard-set to the previously discovered optimal disruption.

Competitive Coevolution An alternative to the two-part optimization approach described here is to use so-called *competitive coevolution* to optimize the disruptions and mitigation strategies simultaneously. The idea is to use two separate evolutionary processes, one optimizing disruptions and one optimizing the mitigations. We test a child from one process by iteratively selecting another child from the other process, then pitting them against one another in a simulation. We then assess the performance of the child as the average, minimum, or maximum performance over simulation tests against M opposing children. The opposing optimization process assesses its children in exactly the same way. Thus as the mitigation strategies are improving, so to are the disruption approaches to challenge them.

5 Concluding Remarks

In sum, building a supply chain map/model, based on information from domain experts, enables one to create and accurately simulate a supply chain via discrete event simulation. The map/model and simulations can be used to identify potential disruptions and promising mitigations, and perform experiments which quantify their effect on the supply chain's operations. Such experiments aid in identifying the most concerning disruptions and in particular related vulnerable nodes and links in the supply chain. The experiments also allow for analysis of the efficacy of the mitigations. An optimization model identifies worst-case disruption scenarios based on criminal organization capabilities, and experiments allow one to test mitigations against these worst-case disruptions. Put together, the methodology enables one to identify past, existing, or planned disruptions to supply chains. We have found that the general methodology applies to three very different supply chains and should be widely applicable to others.

References

[1] Sean Luke, Robert Simon, Andrew Crooks, Haoliang Wang, Ermo Wei, David Freelan, Carmine Spagnuolo, Vittorio Scarano, Gennaro Cordasco, and Claudio Cioffi-Revilla. The MASON simulation toolkit: Past, present, and future. In *International Workshop on Multi-Agent-Based Simulation (MABS)*, 2018.

[2] Sean Luke. Ecj then and now. In *Proceedings of the genetic and evolutionary computation conference companion*, pages 1223–1230, 2017.

[3] Keith Sullivan, Mark Coletti, and Sean Luke. Geomason: Geospatial support for mason. Technical report, Department of Computer Science, George Mason University, 2010.

[4] Giuseppe D'Ambrosio and Sean Luke. Hybrid agent-based and discrete event simulation in mason. In *In Society for Simulation and Modeling Annual Modeling and Simulation Conference (ANNSIM)*, 2023.

.