Simulating Supply-Chain Contract Execution: A Multi-Agent Approach (Extended Abstract)

Long Tran Tran Cao Son Dylan Flynn Marcello Balduccini
New Mexico State University Saint Joseph's University
longtran@nmsu.edu stran@nmsu.edu df752850@sju.edu mbalducc@sju.edu

Supply chains exhibit complex dynamics and intricate dependencies among their components, whose understanding is crucial for addressing the challenges highlighted by recent global disruptions. This paper presents a novel multi-agent system designed to simulate supply chains, linking reasoning about dynamic domains and multi-agent systems to reasoning about the high-level primitives of the NIST CPS Framework. Our approach synthesizes existing research on supply chain formalization and integrates these insights with multi-agent techniques, employing a declarative approach to model interactions and dependencies. The simulation framework models a set of autonomous agents within a partially observable environment, and whose interactions are dictated by contracts. The system dynamically reconciles agents' actions, assessing their feasibility and consequences. Based on the state of the domain, the simulation framework also draws conclusions about the high-level notions of requirements and concerns of the NIST CPS Framework, which provide a uniform and domain-agnostic vocabulary for the understanding of such complex systems as supply chains.

1 Introduction

Building on our previous work on formalizing supply chains and the underlying contracts [2], this paper introduces a simulation of the dynamics of supply that relies on a view of the supply chain as a multi-agent system. Central to our approach is the use of formal contracts, which define the obligations and interactions between agents within the simulation environment. Our simulation integrates an additional knowledge layer based on the National Institute of Standards and Technology's Cyber-Physical Systems (NIST CPS) Framework [1]. The CPS Framework makes it possible to link the low-level view of contracts to the high-level aspects of stakeholders' requirements and concerns about the supply chain as a whole. By integrating a multi-agent simulation with the structured approach of the NIST CPS Framework, we aim to capture the nuanced interplay of contractual obligations, agent behaviors, and their ramifications on the supply chain, providing insights into the potential points of failure and enabling strategies to improve resilience. The agents and the simulation environment are modeled using declarative techniques through Answer Set Programming (ASP) [3]. This approach allows for a clear specification of the logic governing agent behaviors and their contractual interactions. This paper presents not only the theoretical underpinnings of our simulation model but also a practical use case that illustrates its potential. Through these simulations, stakeholders can better understand the critical dependencies within their supply chains, evaluate the robustness of contractual arrangements, and explore strategies to enhance overall resilience.

2 A Distributed Multi-Agent Simulator

Two main components of the system are the environment simulator and the agent program. The code of the system is available on GitHub at `github.com/ltran1612/Research_CPS_SupplyChain`.

Submitted to:
SOS 2007

Environment Simulator. The environment simulator consists of two main parts: the controller and the reasoning engine. The reasoning engine is a logic program that takes a state (of the environment) and a set of actions and determines whether the actions can be executed. In this case, the reasoning engine computes the state of the world. The environment simulator works with the global domain, denoted by D_{env}, which is the union of domains of all agents. The main portion of the code of the reasoning engine is for computing the effects of a set of actions on a state of the environment and is similar to the code for computing a plan in [5]. The controller is responsible for all the communication between the environment and the agents (e.g., receiving actions that agents execute, informing agents about the changes in the local state of agents). Besides disallowing the execution of incompatible actions, the simulator can also disrupt the execution of a supply chain by randomly disallowing the execution of some action. This forces the agents, whose actions are not executed, to replan with the new local state.

Agent Program. Each agent program consists of the following components: the action domain, the set of rules for reasoning about concerns of the agent, the set of rules for reasoning about the agent's actions as well as planning, and a controller. The controller is responsible for the communication of the agent with the environment. The controller's behaviour is described by the pseudocode given in Algorithm 1. The set of rules for reasoning about the agent's actions and planning is similar to the code for planning as described in [5].

Algorithm 1 Overall Behavior of the Agent Program

```
Require: agent ID, action domain, set of contracts
  registers ID with environment and wait for acknowledgement
  sends the action domain to the environment and wait for acknowledgement
  $step = 0$
  generate a plan $p$ for the set of contracts
  while true do
      send $p[step]$ (the $step$-th action of $p$) to environment
      wait for response ($local_s$ – the local state) from the environment
      if $local_s \neq \bot$ then            ▷ all actions were executed successfully
          update the current state with $local_s$
          $step = step + 1$
      else                                 ▷ some action cannot be executed
          generate a new plan $p$ for the agent
          $step = 0$                                      ▷ resetting
      end if
  end while
```

Communications. We used the publish/subscribe architecture (MQTT protocol specifically, see [6] for details) to facilitate the communications between agents and the environment. Each entity sends and receives messages by topics through a publish/subscribe broker component (middleman) [4]. Communication between an agent (with a unique *AgentID*) and the environment are done using two topics: "env/*AgentID*" and "for/*AgentID*". The former is for the agent to send and the latter is for receiving from the environment.

3 Case Study: Supply Chain Simulation

As a case study, we leveraged a dataset developed by the Observatory of Economic Complexity (OEC, `https://oec.world/en`), an organization that collects and analyzes international trade data. The dataset contains data about international imports and exports, from which we extracted part of an automotive supply chain. The scenario involves 8 agents with 7 contracts containing a total of 21 clauses, 8 supply chain requirements, and 5 concerns. An example of a contract clause from the scenario is: `C1: A responsible_for produced(vehicle_parts, 9K) when by_week 4`. Agent `A` is called speedy_auto_part. The private parts of the contract map `C1` to various requirements, addressing concerns of the CPS ontology. An example is: `C1: material-safe-for-production`. In this scenario, each agent can perform 4 types of actions: produce, deliver, pay, and receive. Each also keeps track of relevant fluents, like the total amount of payments made. The corresponding domains are encoded using a template. For example, action *produce* from `A` and related fluents are specified by rules including:

```
produce_item(speedy_auto_parts, vehicle_parts).
action(Agent, produce, (Item, Amount)) :- produce_item(Agent, Item), number(Amount).
causes(Agent, produce, Value, produced, increase, ()):-action(Agent, produce, Value).
```

A sample run of the simulator with the OEC agents progresses as follows. At first, the agents register with the environment and send their information (action domains and initial state) to the environment. At each step i, the agents send to the environment the actions they plan to execute. The environment computes the next state, sending back to the agents their local states, and waits for another set of actions. For example, at step 0, `A` has 0 `vehicle_part` and plans to take the action that produces 9,000 parts. The global state of the environment records at step 1 that `A` now has 9,000 `vehicle_part`, which is reflected in the local state of `A` as well. At the end of each iteration, the system determines which clauses are satisfied. For example, at step 0 no clauses are satisfied; at step 1, clause $C1$ is satisfied. In turn, the system uses this information to infer which requirements and concerns are satisfied.

4 Conclusions

In this paper, we presented a novel multi-agent simulation framework designed to address the complexities and challenges inherent in modern supply chains. We demonstrate that standard ASP programs such as planning or diagnosing modules can be integrated into a system for monitoring contract executions. By integrating the NIST CPS Framework with advanced contract formalization techniques, we have developed a robust system capable of simulating diverse supply chain scenarios and assessing the impact of various types of disruptions. Our evaluation on a realistic case study demonstrates that the framework not only enhances the understanding of supply chain dynamics but also provides actionable insights into improving resilience and reliability. It is understandable that a system utilizing ASP technology will inherit all of its problems (e.g., grounding, scalability). However, the prototype works fine with the use cases and appears acceptable to the owner of the use cases. Identifying the limit of the current system in terms of the limit of the system (e.g., the complexity of the domains or contracts) is an interesting issue and is our immediate future work.

Acknowledgement. Portions of this publication and research effort are made possible through the help and support of NIST via cooperative agreement 70NANB21H167.

References

[1] Edward, Christopher Greer, David A. Wollman & Martin J. Burns (2017): *Framework for cyber-physical systems: volume 1, overview*. doi:10.6028/NIST.SP.1500-201.

[2] Dylan Flynn, Chasity Nadeau, Jeannine Shantz, Marcello Balduccini, Tran Cao Son & Edward Griffor (2023): *Formalizing and Reasoning about Supply Chain Contracts between Agents*. In: *25th PADL*, 13880, doi:10.1007/978-3-031-24841-2_10.

[3] Michael Gelfond & Vladimir Lifschitz (1991): *Classical Negation in Logic Programs and Disjunctive Databases*. *New Generation Computing* 9, pp. 365–385, doi:10.1007/BF03037169.

[4] B. Reselman (2021): *The pros and cons of the Pub-Sub architecture pattern*. Red Hat. Available at `https://www.redhat.com/architect/pub-sub-pros-and-cons`.

[5] Tran Cao Son, Enrico Pontelli, Marcello Balduccini & Torsten Schaub (2022): *Answer Set Planning: A Survey*. *Theory and Practice of Logic Programming*, pp. 1–73, doi:10.1017/S1471068422000072.

[6] M. Yuan (2021): *Getting to know MQTT*. IBM. Available at `https://developer.ibm.com/articles/iot-mqtt-why-good-for-iot/`.