Framework para simplificar la construcción del modelado y simulación de cadenas de suministros

DIRECTOR: DR. ALEJANDRO FERNÁNDEZ
CODIRECTOR: DR. JORGE HERNÁNDEZ (UNIVERSIDAD DE LIVERPOOL)
ALUMNOS: PANKOW MATIAS NAHUEL Y SANCHEZ ESTEBAN MARTIN

Temas a tratar

- Introducción
- Objetivos de la tesina
- Enfoque general
- Diseño del framework
- Algunos detalles de implementación
- De una idea a un modelo de simulación
- Caso de estudio
- Conclusiones

Introducción

- ¿Qué es una cadena de suministros?
  - Una cadena de suministros es un conjunto de empresas que pasan materiales.

Imagen de los componentes que integran la gestión de cadenas de suministros.

Introducción

- ¿Qué es un modelo?¿Y para qué se utiliza?
  - Un modelo es una simplificación de la realidad y nos permite comprender sistemas complejos en su totalidad.

Imagen de un modelo de automóvil.

Introducción

- ¿Qué es una simulación? ¿Y para qué nos sirven?
  - Puede definirse a la simulación como la experimentación con un modelo que imita ciertos aspectos de la realidad. Esto permite trabajar en condiciones similares a las reales, pero con variables controladas y en un entorno que se asemeja al real pero que está creado o acondicionado artificialmente.

Imagen de un simulador de carreras

Introducción

- ¿Porque es interesante simular modelos de cadenas de suministros?
  - Gracias a estas simulaciones, las organizaciones pueden evaluar en un entorno controlado, similar al mundo real, posibles escenarios de negociaciones, para intentar alcanzar el más favorable para la cadena de suministros.

Imagen de LLamasoft, simulador de cadenas de suministros

Introducción

- ¿Que es un framework?
  - Es un sistema que se puede personalizar, especializar o ampliar para proporcionar capacidades más específicas, más apropiadas o ligeramente diferentes.

Imagen de algunos frameworks web

Introducción

- Como antecedente a esta tesina de grado, contamos con la tesis doctoral “Propuesta de una arquitectura para el soporte de la planificación de la producción colaborativa en cadenas de suministros de tipo árbol” desarrollada por el Dr. Jorge Hernández.

Objetivos de la tesina

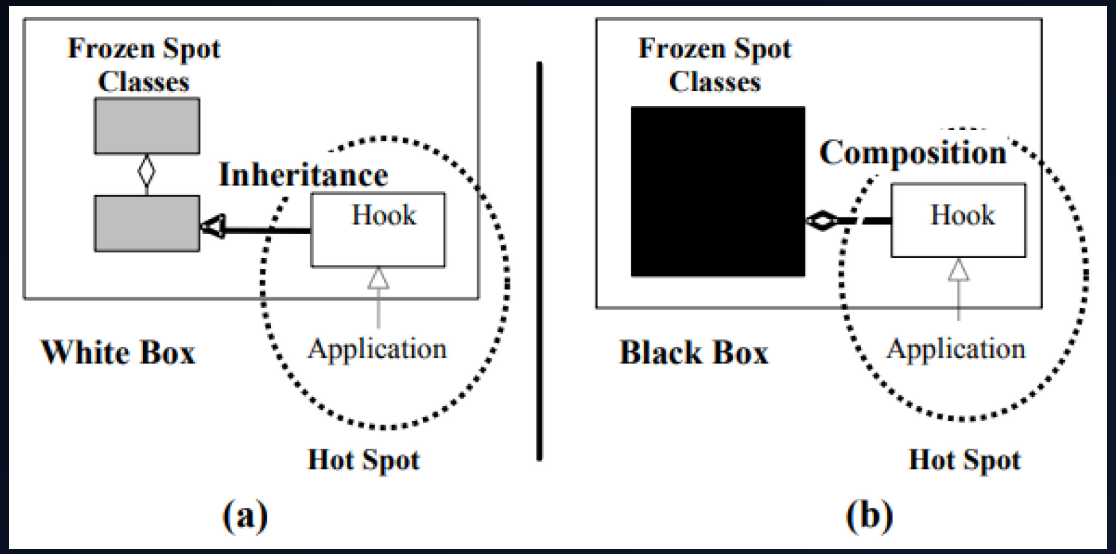
- Obtener un framework que simplifique el armado de modelos de cadenas de suministros.
- Ofrecer mejoras, en relación a estrategias ya existentes, en términos de flexibilidad, implementación, ejecución, despliegue e interacción de los modelos.
- Ofrecer herramientas de monitoreo de ejecución de los modelos.
- Evaluar los aportes en un caso concreto.

Enfoque general

- Un framework para la construcción y simulación de cadenas de suministros.
- Herramienta interactiva para el modelado y ejecución de simulaciones de cadenas de suministros.

Enfoque general

- Un framework para la construcción y simulación de cadenas.
  - Hot spots personalizables para cada nodo de la cadena de suministros.
  - Frozen spots lógica compartida por todos los nodos de la cadena.
  - Hook methods elementos públicos del framework con los cuales el usuario puede construir, métodos que, una vez definidos conformarán la distinción entre varias simulaciones.



Clasificación de los frameworks, (a) de caja blanca y (b) de caja negra.

Enfoque general

- Herramienta interactiva para el uso y ejecución de simulaciones
  - Interfaz en la cual podemos administrar cadenas (C.R.U.D., clonar, vaciar).
  - Interfaz para administrar agentes, enviarles mensajes y observar las conversaciones resultantes.
  - Permite elegir entre el ambiente de ejecución local o AWS.

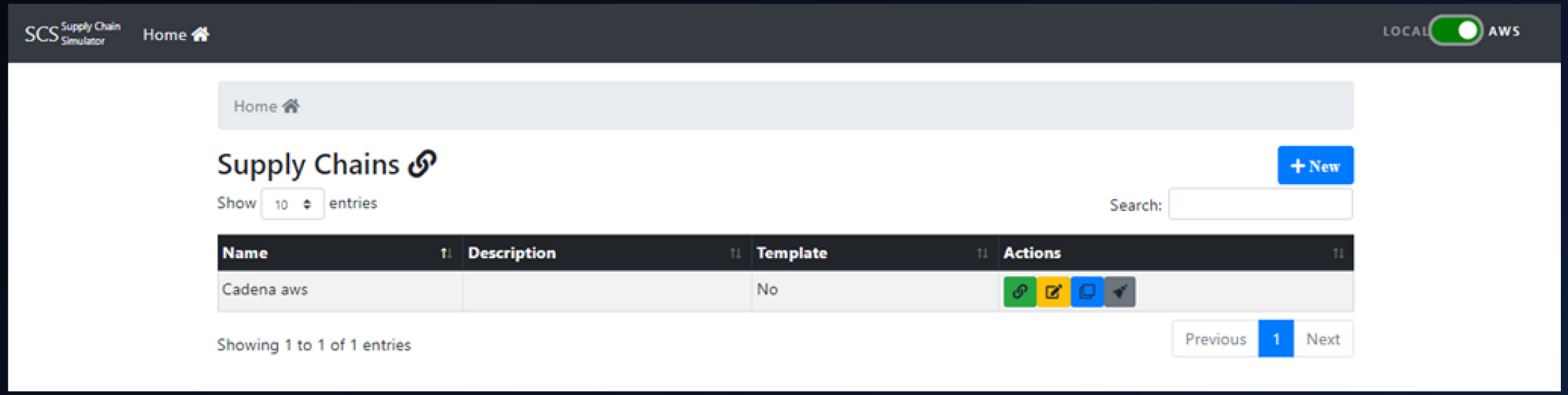


Imagen de la herramienta interactiva

Enfoque general

- Tecnologías utilizadas:
  - Serverless.
  - Base de datos NoSql.
  - FIPA.
  - REST.

{JSON}
JavaScript Object Notation

fipa

serverless

Diseño del framework

- Alcance del framework: Permitir el desarrollo de sistemas para la creación y simulación de cadenas de suministros.
- Además, permitir la interacción con cadenas externas.

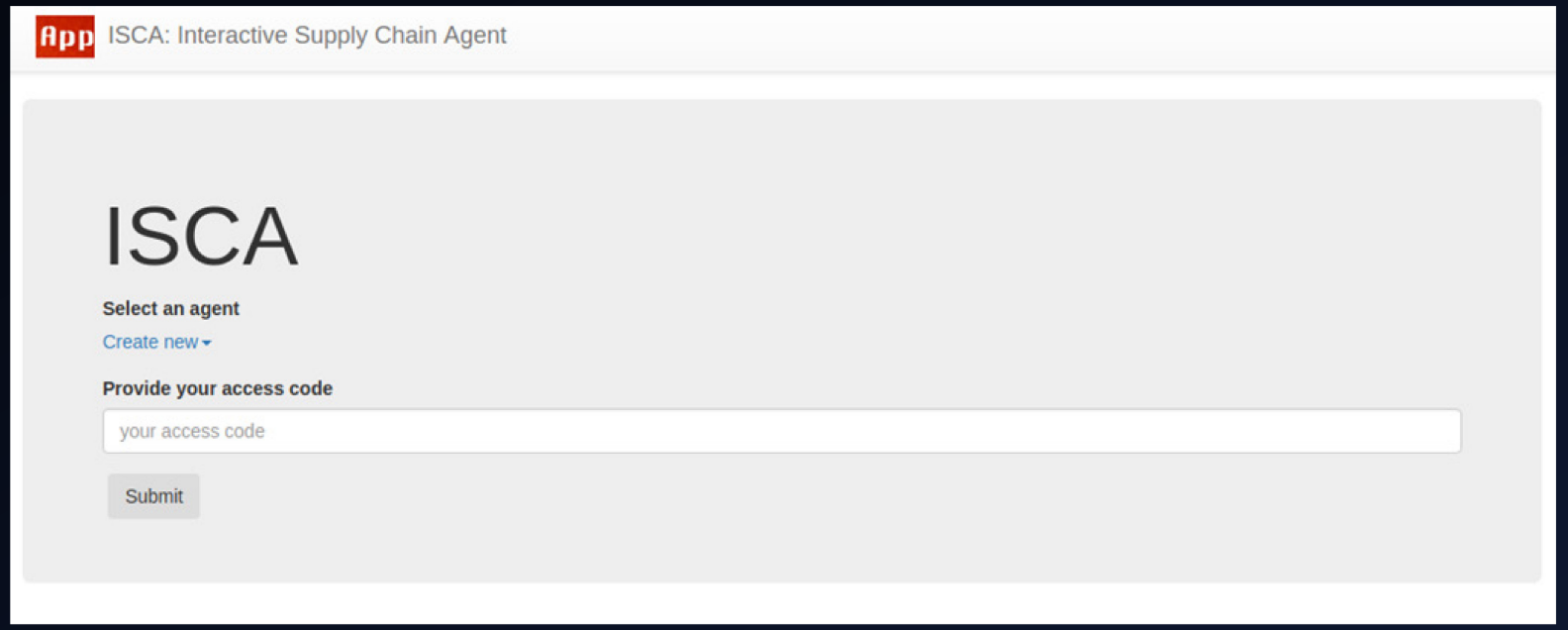


Imagen de cliente interactivo externo

Diseño del framework

- Frozen Spots: Los elementos inmutables de nuestro framework está dado por el almacenamiento de los datos, comunicación entre agentes y la ejecución de la simulación.

Diseño del framework

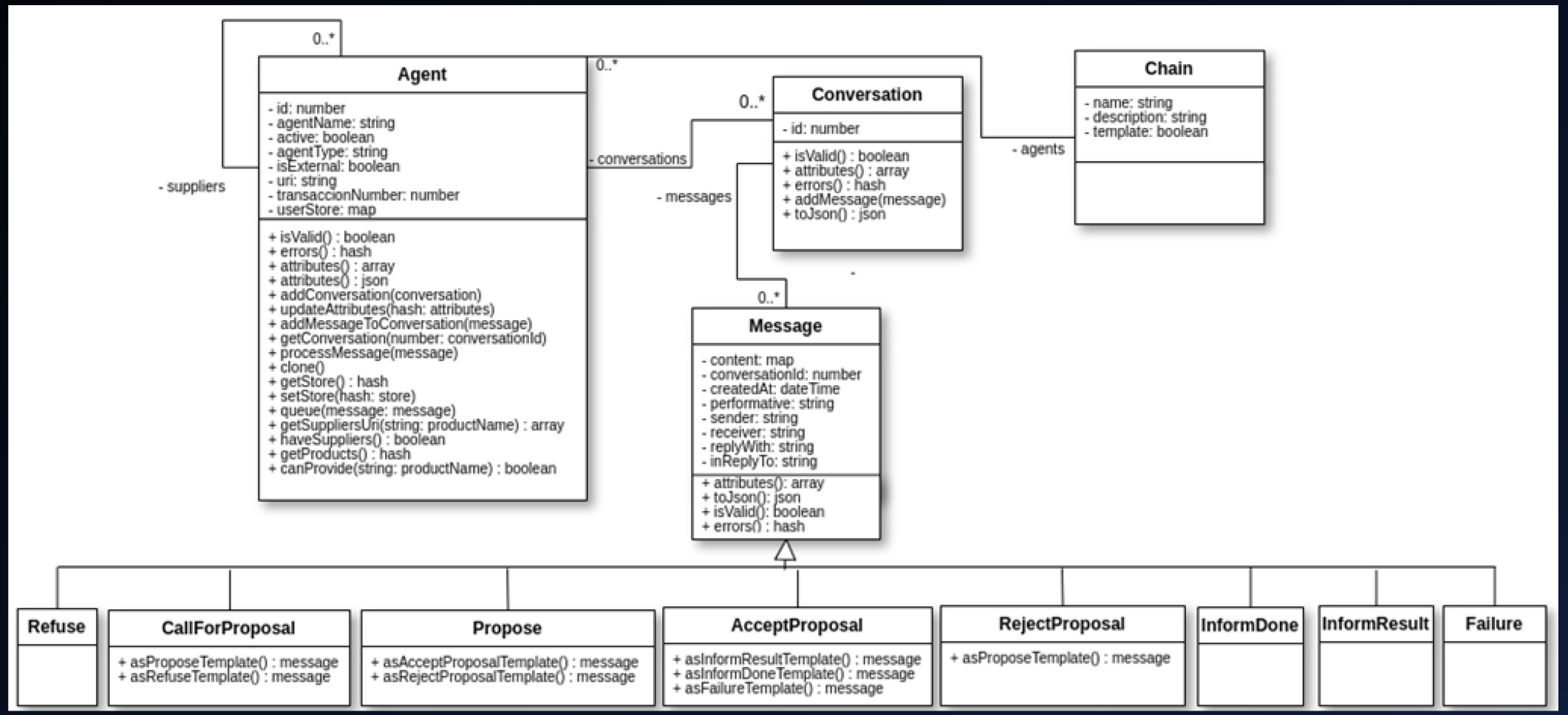
- Hot Spots: Al implementar una nueva clase de agente veremos varios hotspots posibles a implementar. Esto tiene que ver, en su mayoría, en cómo reaccionaría el agente a los distintos mensajes que puede recibir.
  Algunos de ellos son los siguientes:
  - processCallForProposal(message)
  - processPropose(message)
  - processAcceptProposal(message)
  - processRejectProposal(message)
  - defaultStore()

Diseño del framework

- Hook methods:
  - getStore()
  - setStore(store)
  - queue(message)
  - getConversation(conversationId)
  - haveSuppliers()

Diseño del framework

- Diagrama de clases



Algunos detalles de la implementación

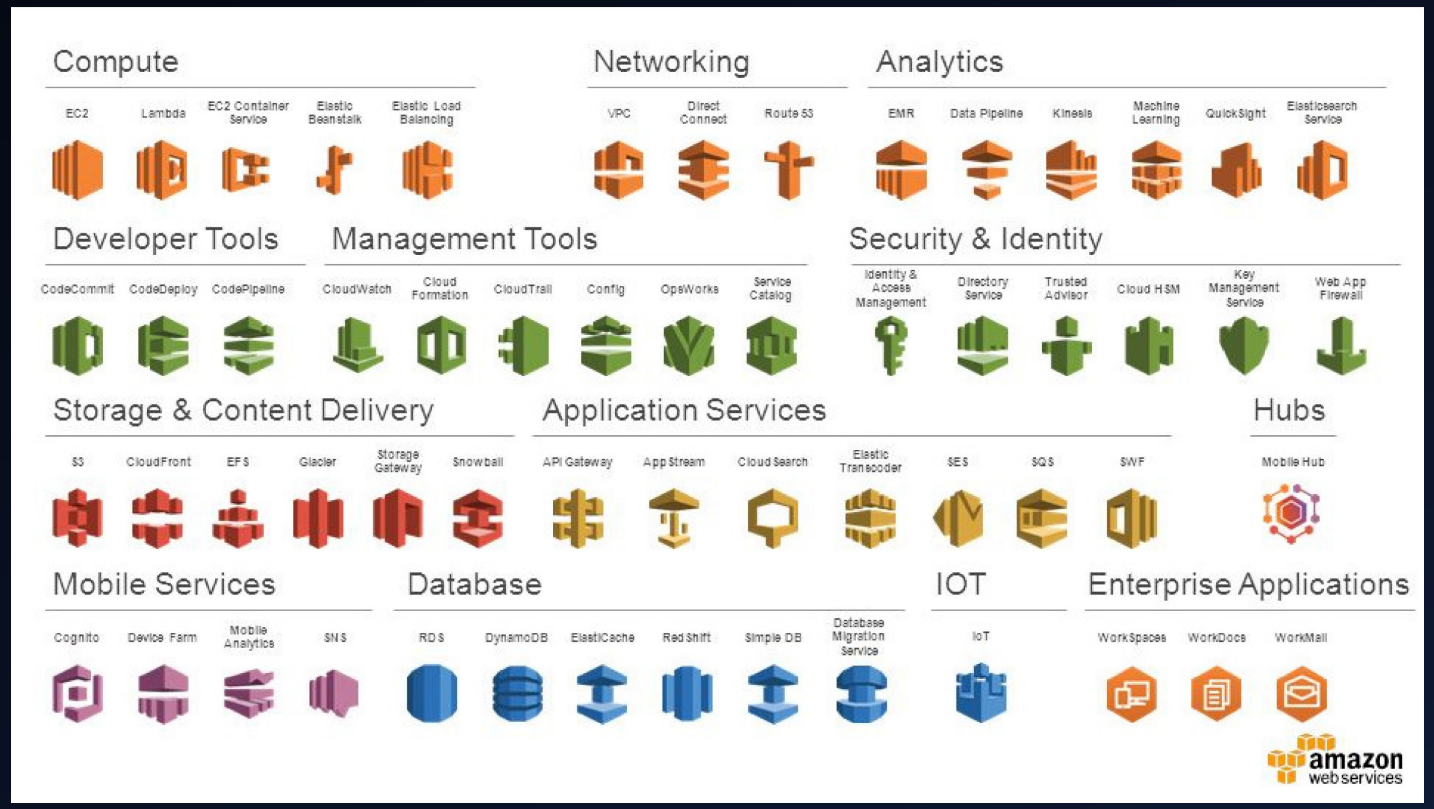
- Ambientes de ejecución posibles:
  - Ambiente AWS.
  - Ambiente Local.

Ambiente AWS

Ambiente Local

Algunos detalles de la implementación - AWS

- Servicios utilizados:
  - Lambda.
  - CloudWatch.
  - IAM.
  - S3.
  - Api Gateway.
  - DynamoDB.



Servicios brindados por AWS

Algunos detalles de la implementación - Local

- Servicios utilizados:
  - Serverless Offline.
  - DynamoDB local.

Manejador de paquetes de NodeJS

Entorno de ejecución para JavaScript

De una idea a un modelo de simulación

- Pasos a seguir:
  - Imaginarlo
    - ¿Quienes conforman la cadena?
    - ¿Qué productos comercializan?
    - ¿Cómo negocian?
  - Desarrollar las clases
    - Utilizar el generador de clases
  - Armar la cadena
    - Herramienta interactiva
    - Api REST(Postman, curl, etc)
  - Simular
    - Herramienta interactiva
    - Api REST(Postman, curl, etc)

```
'use strict';
var Agent = require('../agent');
let Message = require('../message');

class Distribuidor extends Agent {

  constructor(dataAgent={}){
    dataAgent.agentType = 'Distribuidor';
    super(dataAgent);
  }
  // Subclasses should redefine this method
  defaultStore(){
    return {
      "products": {
        "Paquetes de verduras": {"price": 10, "count": 1},
      }
    }
  }
```

Caso de estudio

- Paquetes de verduras procesadas para sopa.
- Cadena de suministros integrada por:
  - (a) Distribuidor.
  - (b) Empacadora.
  - (c) 2 Granjas.
  - (d) Fábrica de envases.

(a)

(b)

(c)

(d)

(c)

Caso de estudio - Distribuidor

- Características
  - Stock limitado de paquetes de verduras
  - Lista de proveedores:
    - Empacadora
  - Toma de decisiones:
    - Con stock disponible
    - Sin stock disponible

```
'use strict';
var Agent = require('../agent');
let Message = require('../message');

class Distribuidor extends Agent {

  constructor(dataAgent={}){
    dataAgent.agentType = 'Distribuidor';
    super(dataAgent);
  }
  // Subclasses should redefine this method
  defaultStore(){
    return {
      "products": {
        "Paquetes de verduras": {"price": 10, "count": 1},
      }
    }
  }
  // Subclasses should redefine this method
  processCallForProposal(message){ ...
  }
  // Subclasses should redefine this method
  processPropose(message){ ...
  }
  // Subclasses should redefine this method
  processAcceptProposal(message){ ...
  }
  // Subclasses should redefine this method
  processInformDone(message){ ...
  }
  // Subclasses should redefine this method
  processRefuse(message){ ...
  }
  // Subclasses should redefine this method
  processFailure(message){
  }
  // Subclasses should redefine this method
  processInformResult(message){
  }

  canSupplyRequest(message){ ...
  }

  prepareMessageToProviders(message){ ...
  }

  prepareRefuse(message){ ...
  }

  prepareAcceptProposal(propose){ ...
  }
```

Caso de estudio - Empacadora

- Características
  - Sin stock
  - Bill of materials(BOM)
  - Lista de proveedores:
    - Granjas
    - Fábrica de envases
  - Toma de decisiones:
    - Búsqueda del menor precio

```
'use strict';
var Agent = require('../agent');
let Message = require('../message');

class Empacadora extends Agent {

  constructor(dataAgent={}){
    dataAgent.agentType = 'Empacadora';
    super(dataAgent);
  }
  // Subclasses should redefine this method
  defaultStore(){
    return {
      "products": {
        "Paquetes de verduras": {"price": 10, "count": 0},
      },
      "bom": {
        "Tomates": 5,
        "Zanahorias": 6,
        "Zapallos": 7,
        "Envases": 1
      }
    }
  }
  // Subclasses should redefine this method
  processCallForProposal(message){ ...
  }
  // Subclasses should redefine this method
  processPropose(message){ ...
  }
  // Subclasses should redefine this method
  processAcceptProposal(message){ ...
  }
  // Subclasses should redefine this method
  processRejectProposal(message){
  }
  // Subclasses should redefine this method
  processRefuse(message){ ...
  }
  // Subclasses should redefine this method
  processFailure(message){
  }
  // Subclasses should redefine this method
  processInformDone(message){ ...
  }
  // Subclasses should redefine this method
  processInformResult(message){
  }

  canSupplyRequest(message){ ...
  }
```

Caso de estudio - Granjas / Envases

- Características
  - Stock ilimitado
  - Sin proveedores
  - Emisores de propuestas

```
'use strict';
var Agent = require('../agent');
let Message = require('../message');

class Granja extends Agent {

  constructor(dataAgent={}){
    dataAgent.agentType = 'Granja';
    super(dataAgent);
  }
  // Subclasses should redefine this method
  defaultStore(){
    return {
      "products": {
        "Tomates": {"price": 10, "count": 200},
        "Zanahorias": {"price": 1, "count": 100},
      }
    }
  }
  // Subclasses should redefine this method
  processCallForProposal(message){ ...
  }
  // Subclasses should redefine this method
  processPropose(message){
  }
  // Subclasses should redefine this method
  processAcceptProposal(message){ ...
  }
  // Subclasses should redefine this method
  processRejectProposal(message){ ...
  }
  // Subclasses should redefine this method
  processRefuse(message){
  }
  // Subclasses should redefine this method
  processFailure(message){
  }
  // Subclasses should redefine this method
  processInformDone(message){
  }
  // Subclasses should redefine this method
  processInformResult(message){
  }

  canSupplyRequest(message){ ...
  }

  getProduct(productName){ ...
  }
}
```

Conclusiones

- Gracias al desarrollo de este framework se logró simplificar el trabajo de creación de sistemas de modelado y simulación de cadenas de suministros, brindando de esta forma que los sistemas creados a partir del mismo requieran menor costo y alcancen una mejor calidad.

Conclusiones

- En relación a estrategias ya existentes, se lograron mejoras en flexibilidad, debido a que gracias a este framework, el usuario puede desarrollar un sistema a su medida, implementación, ejecución y despliegue, gracias a los posibles entornos que posee la herramienta e interacción debido a la utilización de nuevas tecnologías y estándares.

Conclusiones

- El editor de cadenas ofrece al usuario la posibilidad de monitorear de forma amigable las simulaciones.
- Hemos podido experimentar y demostrar todas estas conclusiones en un caso concreto, ejemplificando de forma sencilla una problemática compleja del mundo real.