PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

Sistema multiagente para la resolución del Yard Allocation problem

Felipe Rodrigo Gallegos Montero

INFORME FINAL DEL PROYECTO
PARA OPTAR AL TÍTULO PROFESIONAL DE
INGENIERO CIVIL EN INFORMÁTICA

Julio del 2016

PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

Sistema multiagente para la resolución del Yard Allocation problem

Felipe Rodrigo Gallegos Montero

Profesor Guía: **Claudio Cubillos Figueroa**

Profesor Co-referente: **Rodolfo Villarroel**

Julio del 2016

Dedicatoria

Presento esta dedicatoria mi familia que me ha soportado durante todos estos años y por siempre confiar en que saldría adelante. Gracias por ayudarme por todos estos años, y los por venir.

Felipe Rodrigo Gallegos Montero

Resumen

El Yard Allocation Problem se refiere al problema de almacenar, apilar y retirar contenedores de un puerto marítimo de la manera más rápida y eficiente posible, para así no entorpecer el proceso de importación y exportación de estos. Con este propósito se presenta el estado del arte en la resolución del YAP en general y en particular el sistema de agentes inteligentes como herramienta informática para su resolución.

En particular, se presenta un método existente sobre la resolución del YAP usando sistemas de agentes inteligentes y posibles mejoras de éste para obtener un sistema más simple y eficiente.

Finalmente, y dado el sistema mejorado se presentan las consideraciones del sistema mejorado y la función objetivo de esta.

Abstract

The Yard Allocation Problem refers to the problem of storing, stacking and retrieving containers from a maritime port in the fastest and most efficient way possible, as to not slow down the container import and export process. With this purpose it is shown the state of the art in the solving of the YAP in general, and in particular the system of intelligent agents as an informatics tool for it's resolution.

In particular, an existing method for solving the YAP using systems of intelligent agents, and possible improvements to it to make a simpler and more efficient system.

Finally, the considerations and the target function of the enhanced system are shown.

Índice de contenido

Índice de figuras

1 Presentación del tema

1.1 Introducción

Para toda persona que viva, trabaje o de otra manera tenga cerca a un importante puerto marítimo es muy evidente el enorme espacio que se debe destinar al almacenaje de contenedores entrantes o salientes. Los espacios que requieren dichos procesos se han hecho cada vez más estrechos, lo que crea un problema siempre presente en el comercio marítimo, obligando, en más de una oportunidad, a buscar sitios en lugares muchas veces alejados de los puertos mismos, encareciendo, por tal causa, el comercio.

El negocio marítimo, por lo mismo, puede no resultar suficientemente redituable.

Considerando la cantidad y tamaño de estos contenedores, y la importancia de la industria del transporte, es fácil dimensionar la importancia del problema logístico que presenta mover, ubicar y trasladar estos contenedores en tiempos razonables, lo que conlleva a una gran cantidad de trabajos, papers y estudios al respecto. A este problema se le reconoce como Yard Allocation Problem (YAP).

Para abarcar esto, se procede entonces a investigar las tecnologías y métodos aplicados actualmente para la resolución del YAP, específicamente tecnologías relacionadas con agentes inteligentes y sistemas multiagente, y con esto elaborar el sistema para encontrar una posible solución al problema abarcado.

1.2 Objetivo general

Queda como objetivo general la búsqueda y construcción de una solución al problema del YAP a través de agentes inteligentes más eficiente que los existentes. Por eficiente, se entiende un sistema más simple tras la búsqueda y el encuentro de mejores soluciones.

1.3 Objetivos específicos

- Investigación del estado del arte en YAP
- Revisión de la tecnología de agentes inteligentes
- Aplicación del uso de agentes inteligentes aplicados a la resolución del YAP
- Identificación e implementación de estructuras de datos y agentes inteligentes del sistema a elaborar
- Identificación de las relaciones y acciones de las partes del sistema
- Validación de métricas del sistema en funcionamiento

1.4 Metodología de trabajo

Para la realización de este proyecto se han dispuesto diez horas de trabajo semanales para este proyecto, separadas en dos bloques de cinco horas los martes y jueves. La revisión de papers existentes en este ámbito ha sido crucial para la búsqueda de una mejor solución. Además se ocupan como herramientas de trabajo Eclipse combinado con el plugin JADE que

permiten la construcción de la plataforma informática sobre la cual se va a construir el proyecto hasta conseguir su modelado. Por último, se ahondó en la búsqueda e investigación de código fuente ya existente con respecto a este proyecto o proyectos similares que le dieran mayor base de sustentación al propio estudio en cuestión.

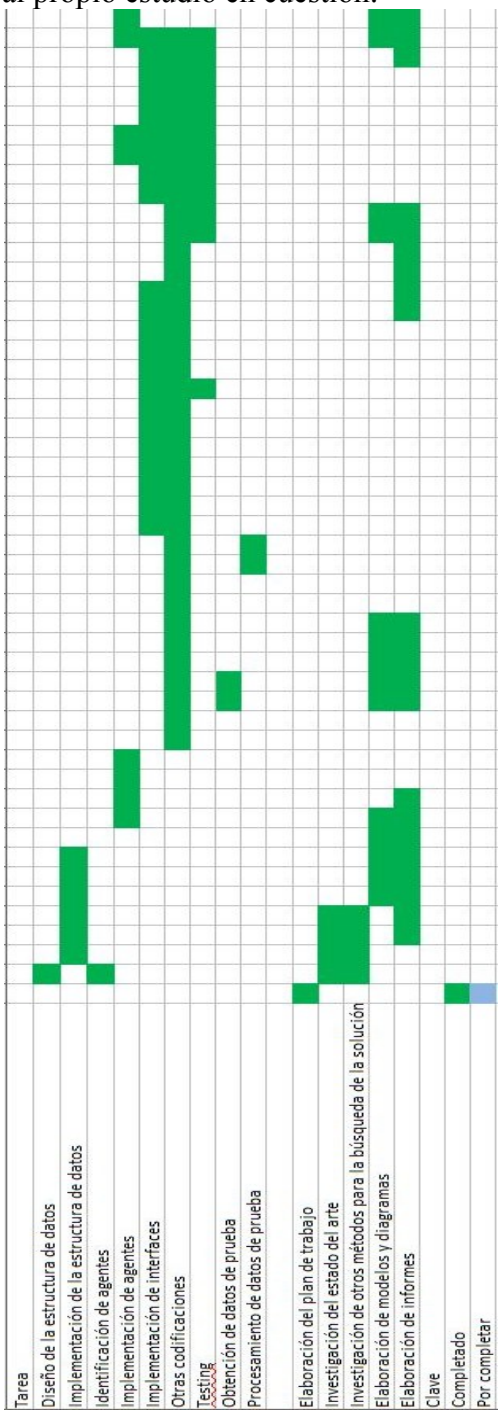

Figura 1.1 Carta Gantt del proyecto al 19-12-2015

2 Sistemas de agentes inteligentes

2.1 Descripción de agente inteligente

Un agente puede ser descrito como una entidad informática autónoma capaz de comunicación con otros agentes y capaces de explorar y modificar su entorno [Mas, 2005]. En general, estos agentes deben de ser capaz de reactividad, capacidad social y proactividad principalmente, y autonomía, movilidad, veracidad, benevolencia, racionalidad y adaptabilidad secundariamente.

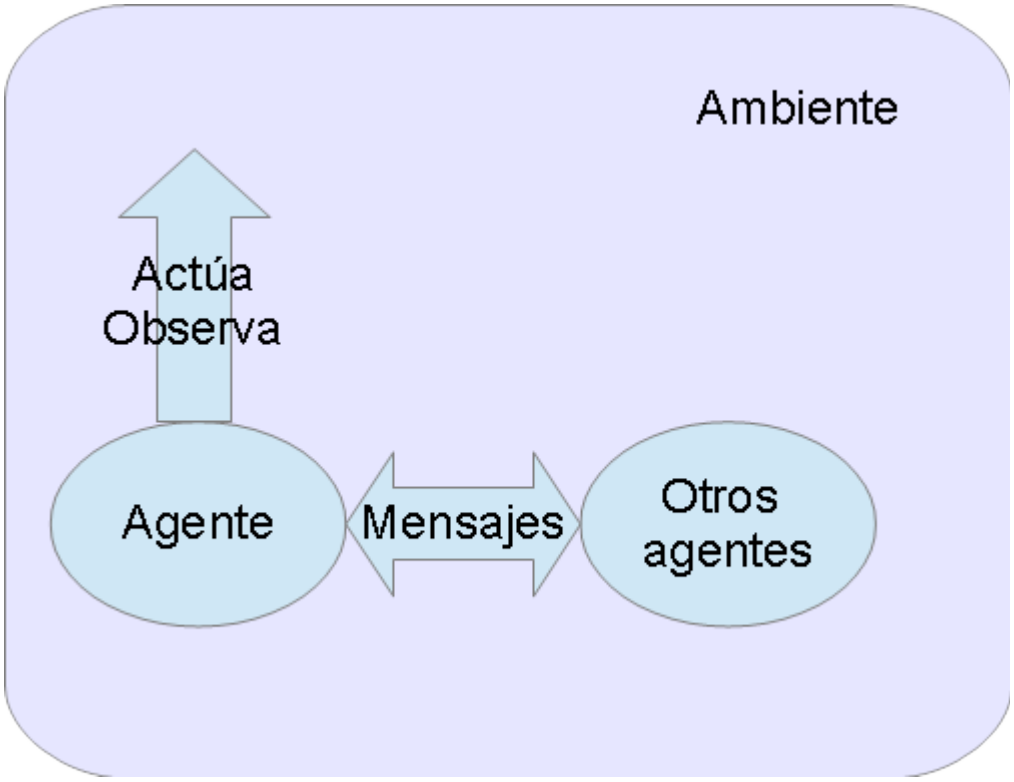

Figura 2.1.1 Interacción agente entorno [elaboración propria]

2.2 Sistemas multiagente

Pero un solo agente no es de mucha ayuda, por lo que en la práctica se usan sistemas de múltiples agentes inteligentes, denominados sistemas multiagente.

Es necesario e imprescindible que estos diferentes agentes tengan una infraestructura en común necesaria para que puedan relacionarse entre sí. La composición de esta infraestructura está integrada por tres componentes importantes: la ontología, el lenguaje de comunicación de agentes (ACL) y el protocolo de interacción de agentes (AIP) [Mas, 2005].

La ontología puede considerarse como un modelo que define con un nivel de formalidad el entorno de los agentes, pudiendo así describir como los agentes pueden interactuar. Ontologías usualmente están compuestas de:

- Clases: referidas a los elementos en general.
- Propiedades: preocupadas las cualidades de las clases.
- Relaciones que describen como las clases se conectan o relacionan entre sí.

Adicionalmente, estas ontologías están estructuradas en lenguajes, que proporcionan el empleo de una ontología de manera clara, formal y no ambigua, permitiendo la comparación e interoperabilidad entre las mismas ontologías.

Ejemplos de ontologías:

- XOL: Ontology eXchange language, US bioinformatics community [SRI, 1999].
- SHOE: Simple HTML. Ontology extension, university of Maryland. [UMD, 2001]
- OML: Ontology markup language, university of Washington.
- DAML: DARPA Agent Markup Language, DARPA project.
- OIL: Ontology Interchange Language, OntoKnowledge project.
- DAML + OIL: W3C [W3C 1, 2001].
- OWL: Web Ontology Language, W3C [W3C 2, 2012].

El ACL, o lenguaje de comunicación de agentes, es el componente que permite la comunicación entre agentes, tanto en el envío como en la recepción de mensajes entre estos, y está modelada en la comunicación hablada entre personas. Este esta integrado por de tres componentes fundamentales, a saber:

- Sintaxis: estructura los símbolos de la comunicación.
- Semántica: que significa cada símbolo.
- Pragmática: interpretación de cada símbolo.

Al estar basado en la comunicación hablada entre personas, el ACL esta igualmente basado en la teoría del acto del habla, por lo que las acciones del ACL pueden dividirse en:

- Locución: el acto de enviar el mensaje
- Ilocución: la intención del mensaje
- Perlocución: la acción final resultado del acto de comunicación

Además, existen los llamados *performatives*, necesarios para eliminar ambigüedad de la intención del mensaje permitiendo acciones específicas de parte del recipiente del mensaje. Finalmente, cabe destacar que existen dos principales ACL en la industria, KQML [UMBC, 1994] y FIPA ACL [FIPA 1, 2002], los que estructuran, en forma de lista, los mensajes a intercambiar entre los agentes y difieren principalmente en los *performatives* posibles.

El AIP, o protocolo de interacción de agentes, describe como estos agentes pueden transmitirse mensajes entre sí, coordinando el comportamiento de estos mismos. El AIP puede considerarse como un *framework* o infraestructura de alto nivel para la implementación de sistemas multiagente. Para efectos de este proyecto, se centrará y usará el protocolo FIPA.

2.3 Protocolos de comunicación de agentes

Ya en poder de un sistema de múltiples agentes inteligentes es necesario obtener una coordinación más allá de la arquitectura elegida. Para esto, existen diferentes protocolos de comunicación que determinan el cómo se comunican estos agentes.

2.3.1 Sistemas de pizarra

Este protocolo descansa en tener toda la información y colaboración entre agentes en la misma área, la llamada pizarra o *blackboard*, y con los agentes representando expertos en el tema, o *knowledge sources*. Esta área es preliminarmente llenada con los datos iniciales del sistema en el cual los agentes observan lo existente y contribuyen, hasta llegar a una solución.

2.3.2 Multiagente Belief Maintenance

Se emplea posee un enfoque basado en inteligencia artificial, para lo cual usa un sistema de “truth-maintenance” (TMS). El protocolo se proporciona en entregar una base común de conocimiento e información transmitida a un grupo de agentes, datos que pueden ser aceptados o no por el grupo de agentes. Además el estado de cada dato no puede ser modificado por un agente en solitario.

2.3.3 Protocolos de negociación

Este tipo de protocolo se funda en grupos de dos o mas agentes tratando de llegar a un consenso, cada uno con sus proprios fines y objetivos. El protocolo se inicia con la comunicación conjunta de sus posiciones, posiblemente conflictivas, a través de los cuales se trata de llegar al concenso a través de concesiones y métodos alternativos.

2.3.4 Protocolo Contract Net

Inicialmente creado y desarrollado para la resolución de problemas de cooperación (ámbito del *task allocation*) en un sistema descentralizado, éste fue extendido para su uso en sistemas multiagente. Éste protocolo provee una resolución al problema de conexión, logrando que se le asigne al agente apropiado una tarea determinada.

El protocolo se basa en un agente *manager* que posee una o más tareas que requieren ser resueltas, siendo los agentes que respondan y resuelvan las tareas los *contractors*. Para esto, el *manager* ejecuta el siguiente procedimiento:

1. Anuncio de la tarea
2. Recepción y evaluación de las ofertas
3. Concesión del contrato a un *contractor* conveniente
4. Recibir y sintetizar los resultados

A su vez, los *contractors* tienen sus propios procedimientos para la recepción de tareas:

1. Recibir anuncios de tareas
2. Evaluar la propria capacidad de respuesta
3. Responder a la oferta, tanto sea aceptando o rechazando dependiendo de su capacidad
4. Si la oferta es aceptada, realizar la tarea
5. Reportar resultados

Además, las tareas pueden ser organizadas en tareas más pequeñas, con los agentes desempeñando las funciones tanto de ser tanto *manager* como *contractor* para una tarea. Además, se puede incluir un modelo formal para ofertar y conceder decisiones, basado en cálculo del costo marginal y similar al *hill-climbing* en el espacio de búsqueda.

Cabe también destacar que en la actualidad el protocolo Contract Net y variantes de este son los mas usados en sistemas multiagente.

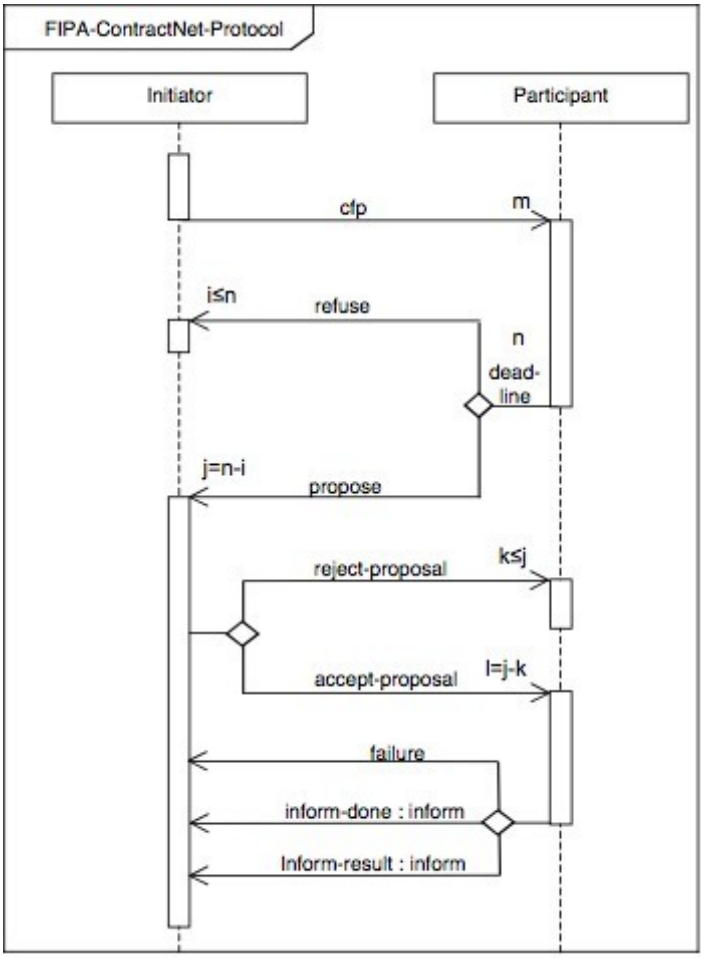

Figura 2.3.4.1 Esquema de FIPA Contract Net [FIPA 2, 2002]

2.3.5 Protocolos basados en subastas

Estos protocolos pueden ser visto como un proceso similar a las subastas reales, pues hay, como en ellas, un agentes subastador (*auctioneer*) queriendo el cumplimiento de una tarea al precio más bajo posible y agentes oferentes dispuestos a realizar la tarea al pago más alto posible, esto resultando en un contrato entre un par subastador-oferente.

Los protocolos basados en subastas poseen múltiples variantes que cambian las reglas de la subasta, y por lo tanto, la interacción entre subastador y oferente.
Las siguientes siendo las mas habituales:

- Inglés (*first price open-cry*): Los oferentes van subiendo sus ofertas, y concluye cuando ningún oferente desee continuar alzando.
- *First price sealed-bid*: Cada oferente entrega una oferta sin conocer las ofertas de los otros oferentes, siendo la oferta mas alta la ganadora.
- Holandés (*descending*): el subastador baja continuamente el precio hasta que un oferente lo tome a ese precio.
- Vickrey (*second price sealed-bid*): Cada oferente entrega una oferta sin conocer las los otros oferentes, resulta ganadora la oferta más alta, pero se paga el precio de la segunda oferta más alta.

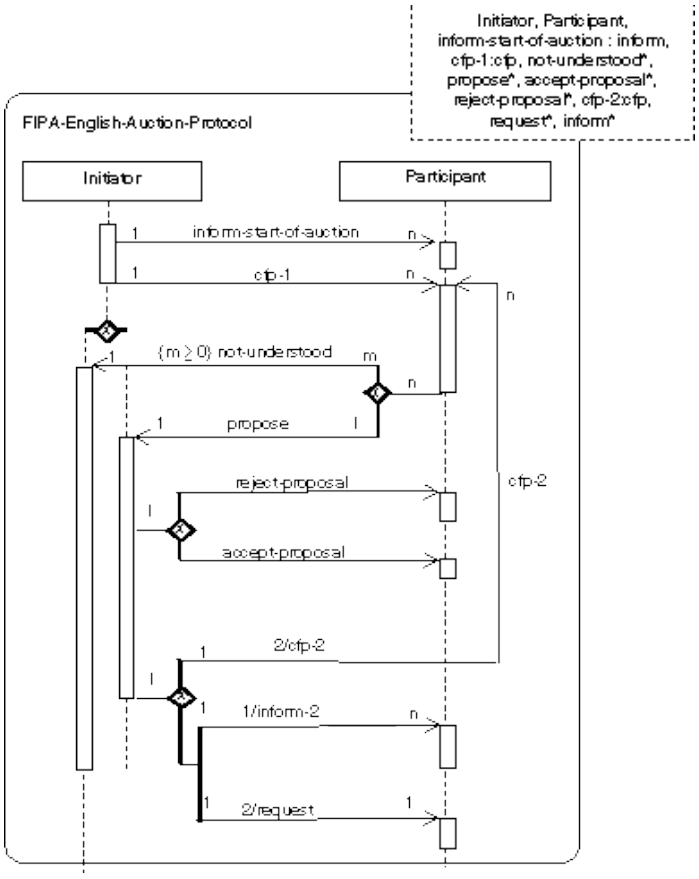

Figura 2.3.5.1 Esquema de FIPA English Auction [FIPA 3, 2001]

2.4 Ambientes de sistemas de agentes inteligentes

Para el uso masivo de sistemas de agentes inteligentes es necesario formalizar y estandarizar estos sistemas con el objetivo de apoyar la interoperabilidad entre sistemas de agentes.

2.4.1 Mobile Agent System Interoperability Facilities (MASIF)

Modelo desarrollado por el *Object Management Group*, fue propuesto como guía para el desarrollo de tecnologías de agentes [OMG, 1997]. Se especifican:

- Agentes:
  - Capacidades: inferencia, planificación, etc
  - Interacciones: sincrónicas, asincrónicas
  - Movilidad: estático, móvil con/sin estado
- Lugares:
  - Ejecución concurrente de agentes
  - Seguridad
  - Movilidad de agentes

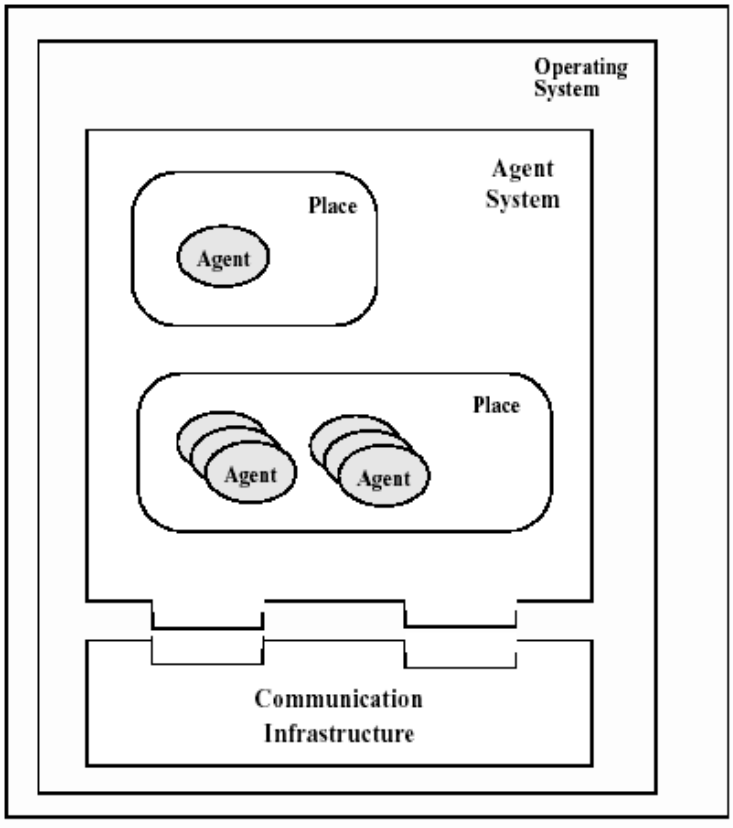

Figura 2.4.1.1 Modelo MASIF [Mas, 2005]

2.4.2 Foundation for Intelligent Physical Agents (FIPA)

Grupo multi-disciplinario que promueve la estandarización de la tecnología de agentes, en sus especificaciones se definen las características que deben cumplir las plataformas de gestión de sistemas multiagente, la cual se centra en el comportamiento externo, dando libertad en el diseño del sistema [FIPA 4, 2016].

Este modelo tiene como principales especificaciones:

- *FIPA ACL Message Structure*: grupo de FIPA *Interaction Protocols*
- *FIPA Agent Management Model*: describe la plataforma de agentes del modelo.

A su vez, la plataforma de agentes está compuesta de:

- Sistema de gestión de agentes (AMS): elemento de gestión principal del sistema, este se encarga de añadir y eliminar agentes, controlar el intercambio de mensajes entre agentes, supervisión de agentes, control de los movimientos de los agentes y gestión de recursos compartidos
- *Directory Facilitator* (DF): identifica agentes por sus servicios, mediante páginas blancas.
- *Message Transport System* (MTS): se encarga del intercambio de mensajes ACL entre agentes en la misma plataforma, o entre plataformas distintas.

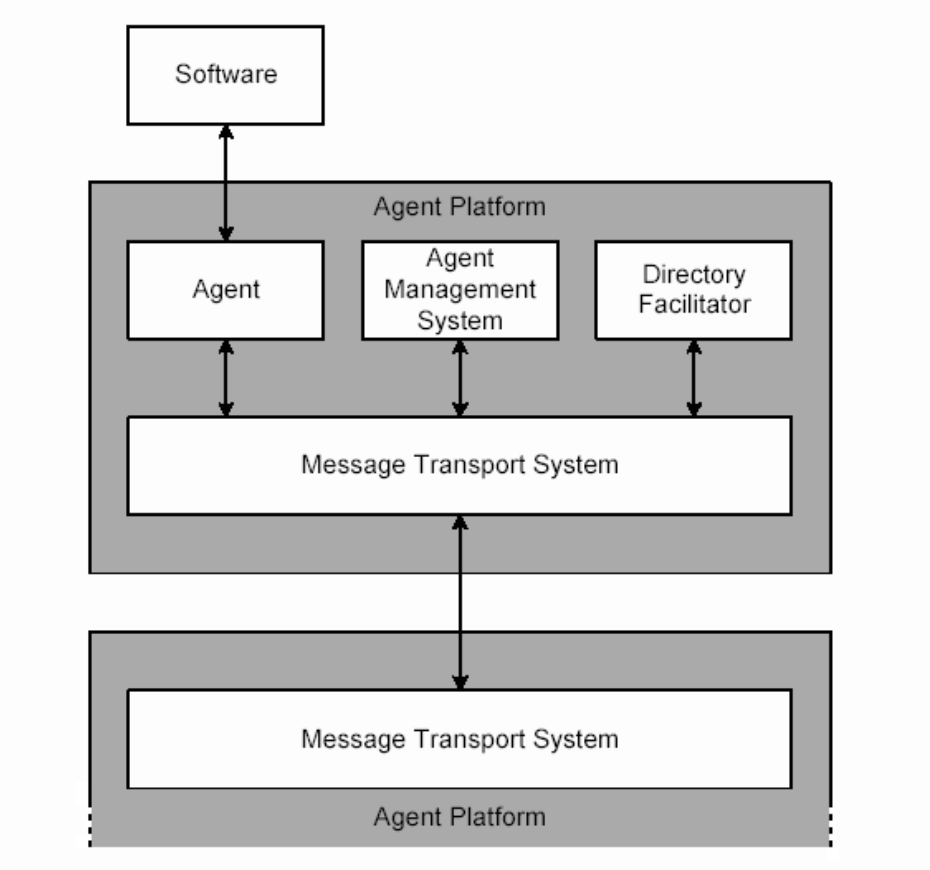

Figura 2.4.2.1 Modelo FIPA [Mas, 2005]

2.5 Middle Agents

Son estereotipos de agentes, y pueden ser considerados como agentes intermediarios entre otros agentes. Estos agentes proponen servicios de ofrecimiento, anuncio, búsqueda, combinación, uso, gestión o actualización.

Ejemplos:

- Facilitator
- Mediator
- Broker
- Matchmaker
- Blackboard

2.6 Plataformas de Agentes

Herramientas de programación para la construcción de sistemas de agentes. Ofrecen:

- Ambiente de agentes
- Infraestructura base de comunicación
- Clases para el comportamiento y mensajes de agentes
- Agente base para su extensión, incluyendo métodos para el envío y captación de mensajes entre agentes

La plataforma de agentes más conocida y usada es la plataforma JADE, hecha en colaboración entre Telecom Italia Labs y la Universidad de Parma [JADE, 2016].

3 El Yard Allocation Problem

El Yard Allocation Problem pretende resolver la ubicación de un contenedor de barco de tamaño estándar de la manera más eficiente, usando como métricas principales los tiempos de despacho o distancias entre el barco y la ubicación del contenedor que va a ser almacenado, o la determinación del sitio y el exterior, o vice versa. Para esto, hay que tener en cuenta el tipo de contenido del contenedor y los almacenes donde éste puede ser guardado, las distancias entre los sitios de atraque de los barcos y los almacenes de contenedores, el sitio del almacén donde puede ser guardado un contenedor, tomando en consideración los diversos datos del contenedor e inclusive el tiempo que permanecerá esperando para ser despachado un contenedor.

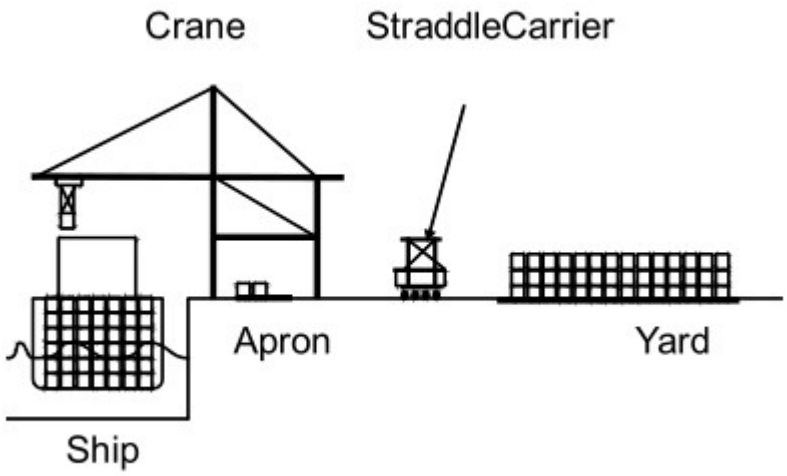

Figura 3.1 Ejemplo de puerto [WiniKoff et al, 2011]

Cabe también destacar que la posición misma del contenedor en el almacén de contenedores es otro factor más a considerar, esta posición se demarca por el sistema BAROTI en rows, bays (horizontales) y tiers (verticales).

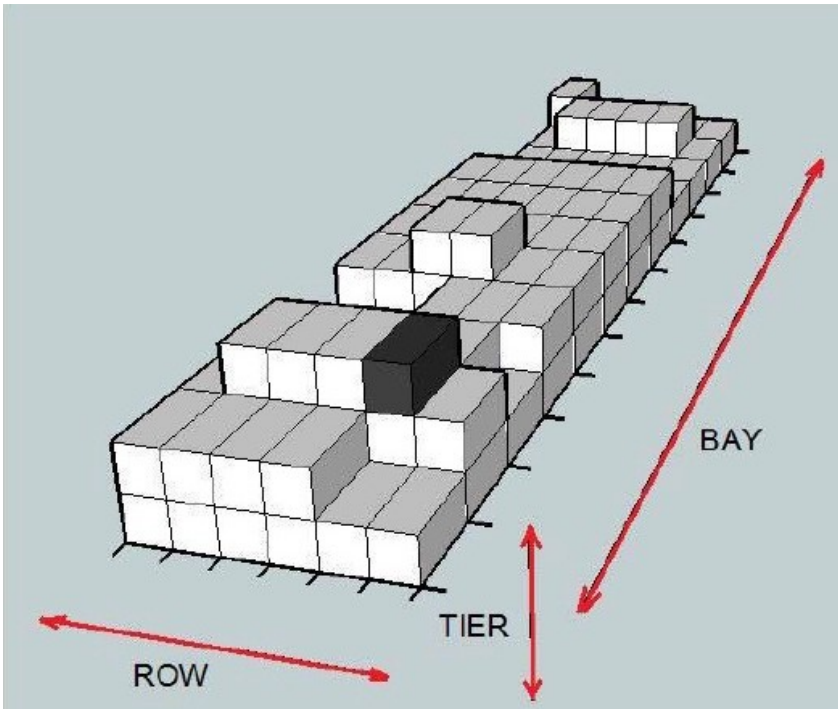

Figura 3.2 Diagrama del sistema BAROTI [Elorrieta, 2011]

Otro factor importante en caso del YAP es la planificación y replanificación de las llegadas y salidas de contenedores a los patios de contenedores. Dado que en muchos casos, como por ejemplo contenedores refrigerados tipo *reefer*, deben ser movidos bajo un estricto plan de trabajo, hay que tomar en cuenta este plan al momento de decidir dánde almacenar. Puede suceder que este último punto no sea suficientemente considerado y, dada en cuenta las características de la industria del transporte, puede provocar un serio revés en todos los planes y obligar, por tanto a cambios bruscos en cortos espacios de tiempo. La naturaleza de estos diferentes imprevistos: adelantos, atrasos, cancelaciones o arribos inesperados tanto del lado terrestre como de los navíos, va a obligar a la toma de decisiones rápidas, muy necesarias en estos casos, con la consiguiente planificación habida hasta ese momento.

Finalmente, considérese también el tipo de grúas dispuestas para el manejo de contenedores, las cuales van a determinar la forma en que se debe planificar los movimientos de los contenedores. Se debe tener en cuenta que en los puertos, se ocupan principalmente grúas tipo *Reach Stacker* y *Rubber Tire Gantry* (RGT).

Figura 3.3 Grúa tipo *reach stacker* [Transtad, 2015]

Figura 3.4 Grúa tipo RGT [Liebherr, 2015]

La principal diferencia en la empleabilidad de estos dos tipos de grúas que condicionan y determinan el problema del manejo de contenedores (algunos requieren de ciertos movimientos y otros cumplen otra función) obedece al menor tamaño de las grúas *reach stacker*, estos permiten necesitan mover los contenedores entre medio la grúa y el contenedor, aumentando los tiempos de despeje. En cambio, las grúas RGT pueden acceder directamente a un contenedor en el medio del patio, moviendo contenedores desde arriba.
Existen otros tipos de grúas existen, como las *straddle carrier*, pero solo las dos anteriores hay, en este momento, en Chile.

3.1 Soluciones al YAP

A través del tiempo se han propuesto y desarrollado muchas resoluciones para el YAP. Los mecanismo de estas resoluciones se han hecho mediante la implementación de heurísticas, algoritmos genéticos o evolutivos, simulaciones y, por último, sistemas basados en agentes inteligentes.

3.1.1 Resolución a través de heurísticas

La resolución basada en heurísticas se funda en obtener la mejor solución posible a través de reglas dadas en un principio y buscar la solución iterando a través de posibles soluciones que sigan estas reglas. Combinando heurísticas se puede llegar a soluciones basadas en meta-heurísticas que, además, buscan las mejores reglas con que buscar la solución. Ejemplos aplicados al YAP incluyen Tabu Search y Squeaky Wheel optimization [Chen et al, 2002], los que se van a presentar a continuación.

3.1.1.1 Squeaky Wheel Optimization

El Squeaky Wheel Optimization (SWO) es un tipo de heurísticas que se fundamentan en la identificación y resolución del “punto” donde se ubica el problema, de una manera similar al que una persona lo haría [Chen et al, 2002]. En el caso de YAP, se ocupa un algoritmo tipo *greedy* para encontrar posibles soluciones y SWO para ubicar qué resolución daría mejores soluciones a cuáles problemas [URL 2002], repitiendo el proceso hasta encontrar la solución.

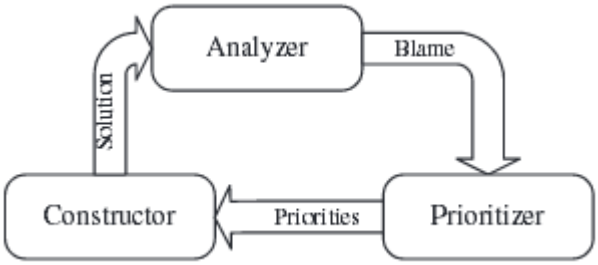

Figura 3.5 Diagrama de procesos del algoritmo SWO

3.1.1.2 Tabu Search

El Tabu Search es un tipo de metaheurística que trata de “moverse” de un óptimo local hacia otro sin pasar por ubicaciones “tabú”, o prohibidas, incorporando memoria adaptativa y exploración responsiva, siendo esta memoria a corto o a largo plazo [Chen et al, 2002].

En YAP se acopla el Tabu Search adentro de SWO para poder reducir más la lista de posibles soluciones y encontrar la solución óptima más rápidamente.

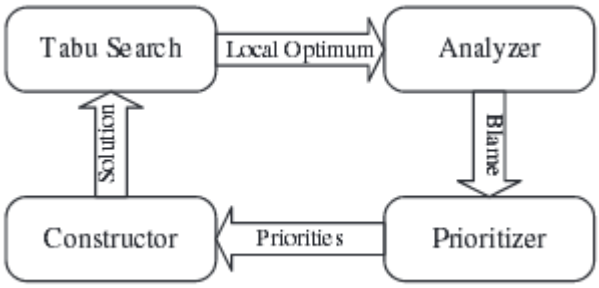

Figura 3.6 Diagrama de procesos del algoritmo SWO combinado con Tabu Search

3.1.2 Resolución a través de algoritmos genéticos

Los algoritmos genéticos son procedimientos de búsqueda que ocupan los mecanismos de selección natural y genéticas naturales [Chen et al 2, 2002] . Este método consiste en utilizar “cromosomas” hechos a partir de la información inicial estructurada de una manera adecuada al caso, a los que su información es reiteradamente cambiada entre cromosoma y mutada, eliminando cromosomas que no se acerquen a una solución aceptable según una fórmula aplicada a su información resultante. Aplicado al YAP es necesario el uso de vectores para poder representar correctamente la posición de los contenedores, teniendo siempre en cuenta evitar la duplicación de contenedores [Chen et al 2, 2002]

Ejemplo [Chen et al 2, 2002] :
Con | denotando puntos de corte:

p1 = (0 1 2 3 4 5 | 6 7 8 9)
p2 = (3 1 2 5 7 4 | 0 9 6 8)

Se hace el corte y se transponen los cromosomas:

p1 = (0 1 2 3 4 5 | 0 9 6 8)
p2 = (3 1 2 5 7 4 | 6 7 8 9)

Los que no son respuestas válidas. Esto se corrige con un algoritmo de reparación, en este caso, eliminando genes repetidos y rellenando con los faltantes al azar:

p1 = (7 1 2 3 4 5 | 0 9 6 8)
p2 = (3 1 2 5 7 4 | 6 0 8 9)

Otros algoritmos genéticos existen que pueden ser mejores en este u otros casos, como por ejemplo algoritmos con múltiples puntos de corte.

3.1.3 Resolución a través simulaciones

El objetivo de la resolución a través de simulaciones es el de pretender encontrar la solución representando virtualmente cada objeto físico involucrado en el problema a resolver, al actuar de esta manera se tienen las ventajas de poder encontrar interacciones influyentes a las cuales pudo, inicialmente, no habérsele otorgado la importancia debida, pero con las desventajas de ser relativamente pesado en recursos. En este caso, esto pasaría a ser una simulación del viaje de cada contenedor, desde el barco hasta el patio, pasando por cada grúa (tanto de barco como de patio) y camión de contenedores.

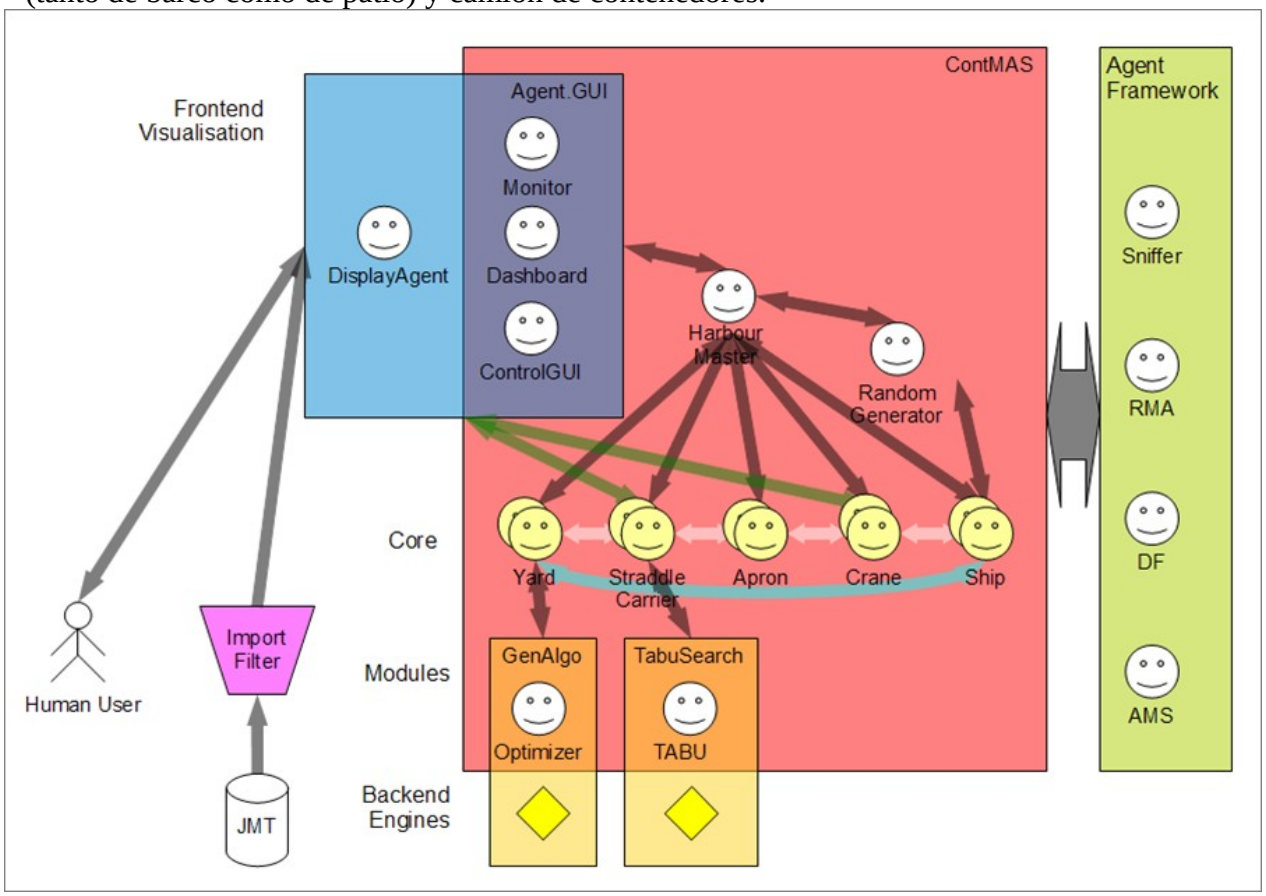

Figura 3.7 Arquitectura de emulación [WiniKoff et al, 2011]

3.1.4 Resolución a través de sistemas multiagente

Visto anteriormente los sistemas multiagente, es posible aplicar estos al YAP como un sistema basado en simulación, pero reduciendo el sistema a lo mínimo necesario y simulando solo las interacciones entre los componentes del problema. La gran mayoría de este tipo de resoluciones están basados en el protocolo Contract Net, dado su previamente mencionado extensivo uso en todo tipo de sistemas multiagente. Un ejemplo de resolución del YAP a través de sistemas multiagente se ve en el próximo capítulo.

4 Propuesta de resolución del YAP

En la memoria de título de Kevin Maturana [Maturana, 2014] se elaboró un modelo en base a cinco a seis agentes inteligentes diferentes: uno para recibir los contenedores a exportar, otro para gestionar las peticiones de las navieras, otro para gestionar en cuál patio hay que enviar los contenedores y cuándo hay que ponerlos de vuelta en circulación, otros tantos representando cada patio y un último que peticiona al agente gestionador de contenedores para enviarlos a los navíos. Este modelo va a ser el modelo de referencia para el modelo que se va a proponer durante el desarrollo de este proyecto, cuya intención va a ser más eficiente y posiblemente más simple que el modelo de referencia.

Este modelo de referencia consta de cinco agentes [Maturana, 2014]:

- Agente Gate: responsable de recibir los contenedores de exportación y comunicarle al agente Export Manager para que le indique el lugar donde serán almacenados.

- El Agente Reserve es el encargado de gestionar las peticiones de exportación de las navieras, lo cual es revisado y comunicado al agente Export Manager para que realice la reserva en el lugar de acopio, según su tipo, fecha de embarque, peso de los contenedores, sitio de atraque y destino de la nave.

- Export Manager: encargado de distribuir la solicitud de reserva a los Agentes Export Sector, donde espera las propuestas de los diferentes agentes y selecciona la mejor alternativa basándose en las características de los contenedores y el lugar de embarque de la nave; también es el responsable de llevar la bitácora de los contenedores que llegan, salen y los sectores que componen el patio de contenedores.

- Export Sector: agentes encargados de almacenar los contenedores de exportación, el agente recibe una solicitud del agente Export Manager para almacenar el contenedor, el cual evalúa dicha solicitud, buscando la mejor alternativa. El agente Export Sector responde con una propuesta y espera adjudicarla para reservar dicho sector. Una vez adjudicada marca los lugares reservados, hasta que lleguen dichos contenedores al lugar de almacenamiento. Luego el Agente Export Manager le comunica el retiro de los contenedores para su embarque.

- Quay Crane: recibe el comunicado de arribo de la nave, éste le informa al agente Export Manager para que envíe los contenedores pertenecientes a dicha nave.

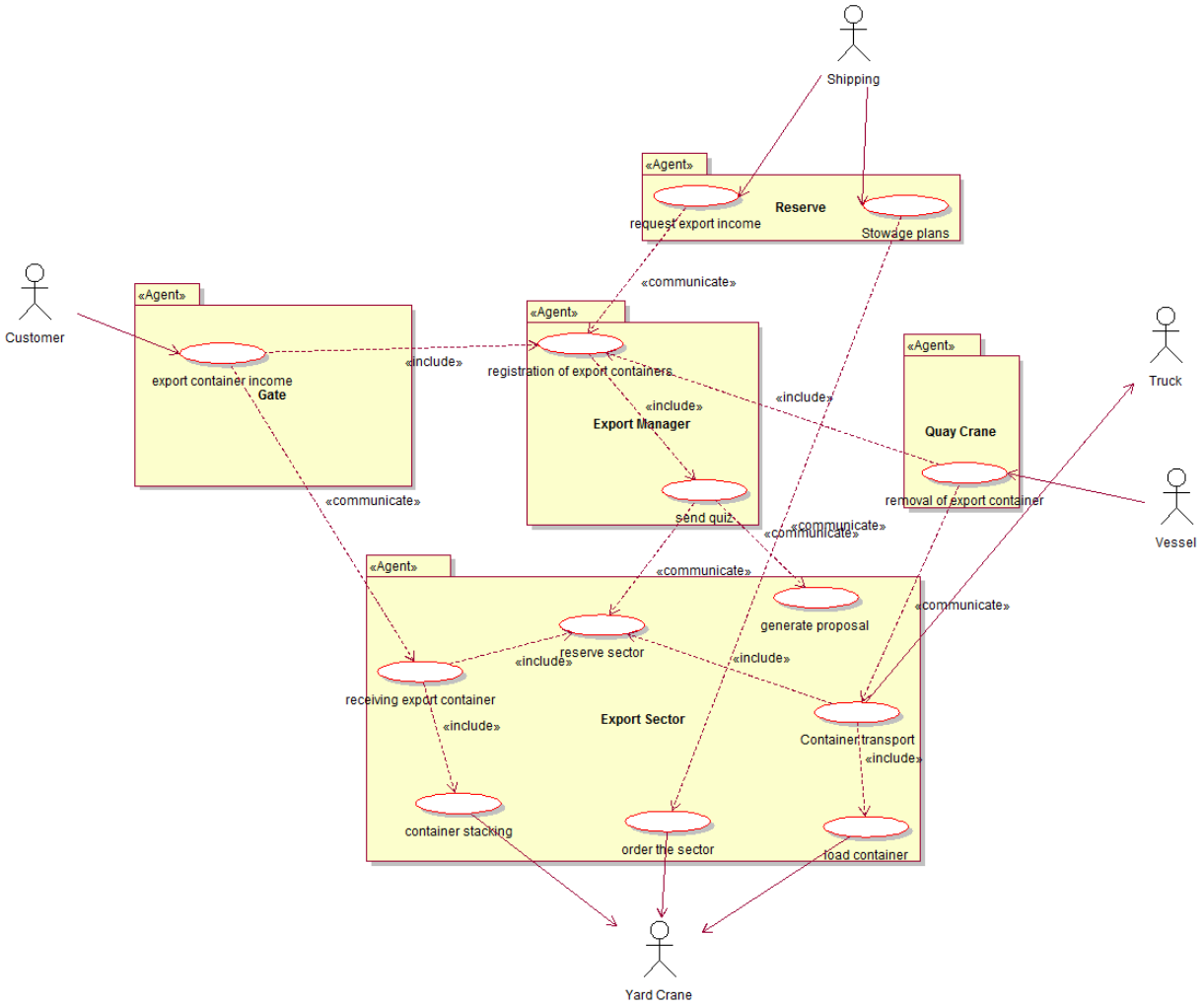

Figura 4.1 identificación de agentes del modelo de referencia [Maturana, 2014]

Tentativamente, se propone un modelo basado en tres tipos de agentes: un agente entrada de contenedores al puerto, un agente salida de contenedores del puerto, y agentes representando los patios de contenedores. Se propone usar un sistema tipo Contract Net o variante de este para determinar cual de los agentes patios es el más adecuado para almacenar el contenedor.

La decisión de usar un sistema basado en tres agentes se debe a que se decidió que el sistema basado en 5 agentes del modelo de referencia es considerado excesivamente simulacionista. Se propone, en cambio, solo las entradas de contenedores, salidas de contenedores y patios de contenedores como lo esencial para la búsqueda de la solución. En el caso a proponer los mismos agentes a cargo de la entrada y salida de contenedores hacen de manager en el Contract Net que inician con los agentes patio de contenedores, para proseguir así combinando las funciones de recibimiento de datos del exterior con la asignación de patios. Posiblemente el sistema a proponer pueda ser simplificado aún más combinando salida y entrada de contenedores.

En pocas palabras, lo que se propone es un sistema más eficiente con respecto al

modelo de referencia al reducir la complejidad del sistema existente. Además, todavía existe el problema de abarcar de mejor manera el problema de imprevistos con respecto a contenedores ya almacenados que tengan que ser retirados en una fecha anterior o posterior a la planeada, y si es necesario reordenar contenedores en ese caso.

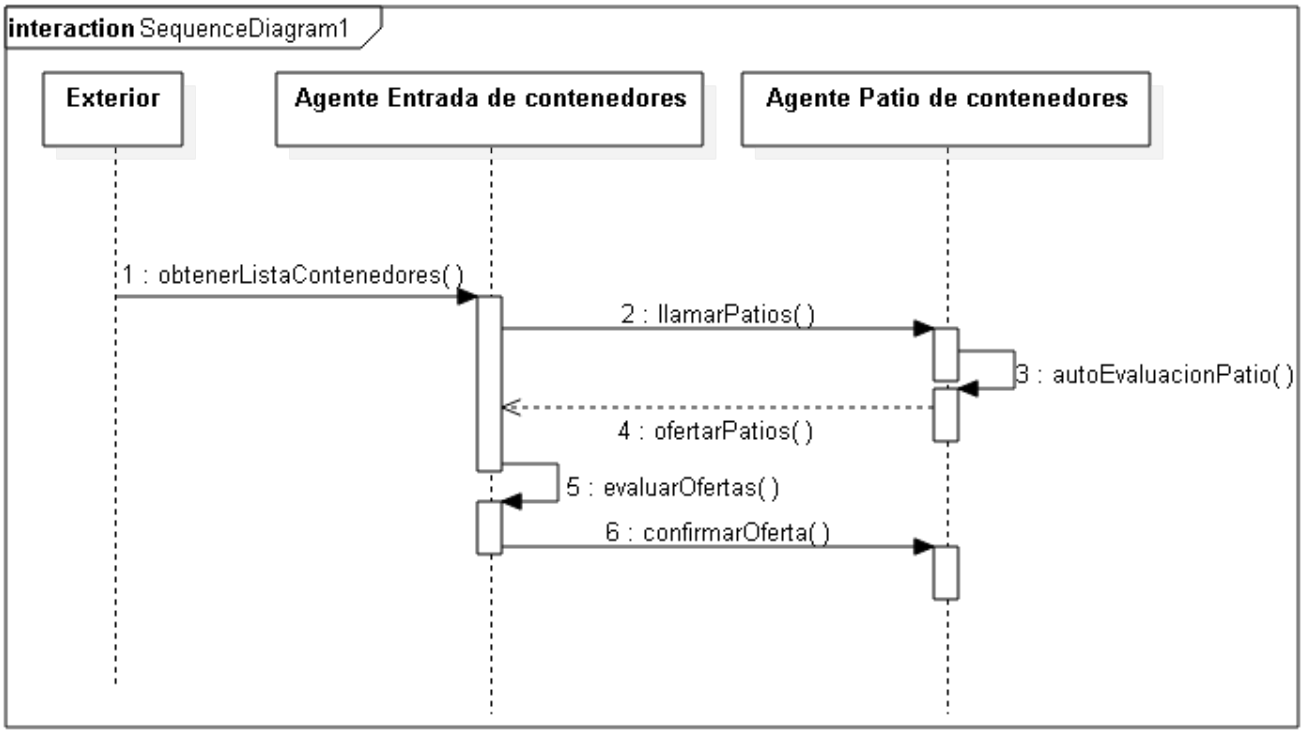

Figura 4.2 Modelo PASSI de ingreso de contenedores del exterior a los patios

En la figura 4.2 se presentó el modelo PASSI, o de secuencia, del proceso de ingresar contenedores a los patios. Usando el protocolo Contract Net el agente de entrada de contenedores tanto recibe la lista de contenedores a almacenar como hace de manager a los agentes patio de contenedores, que hacen de contractors. Usando los datos de la lista de contenedores los agentes patio de contenedores se ofrecen o no a realizar la tarea, y con los mismos datos, se eligen los patios más óptimos para cada contenedor.

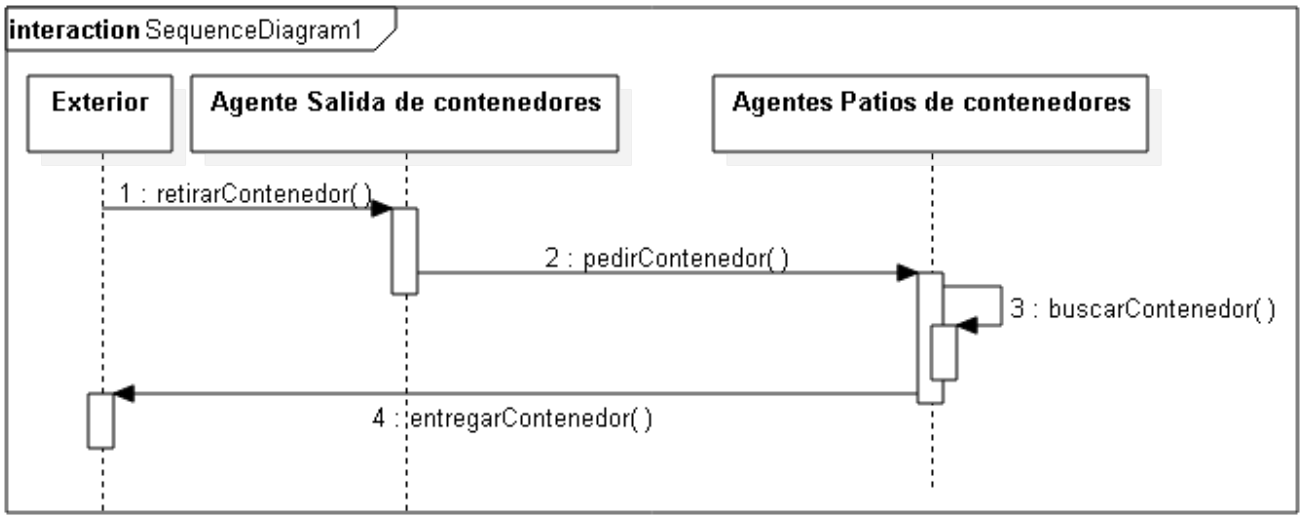

Figura 4.3 Modelo PASSI de retiro excepcional de contenedores de los patios

En la figura 4.3 se presenta el modelo PASSI de un caso excepcional en el que uno, o unos contenedores deban ser retirados antes de tiempo. En esta situación, lo que se hace es que el agente salida de contenedores pide a los agentes patio de contenedores cuales tienen los contenedores pedidos, para ser entregados al exterior, sean estos grúas de barcos o de camiones.

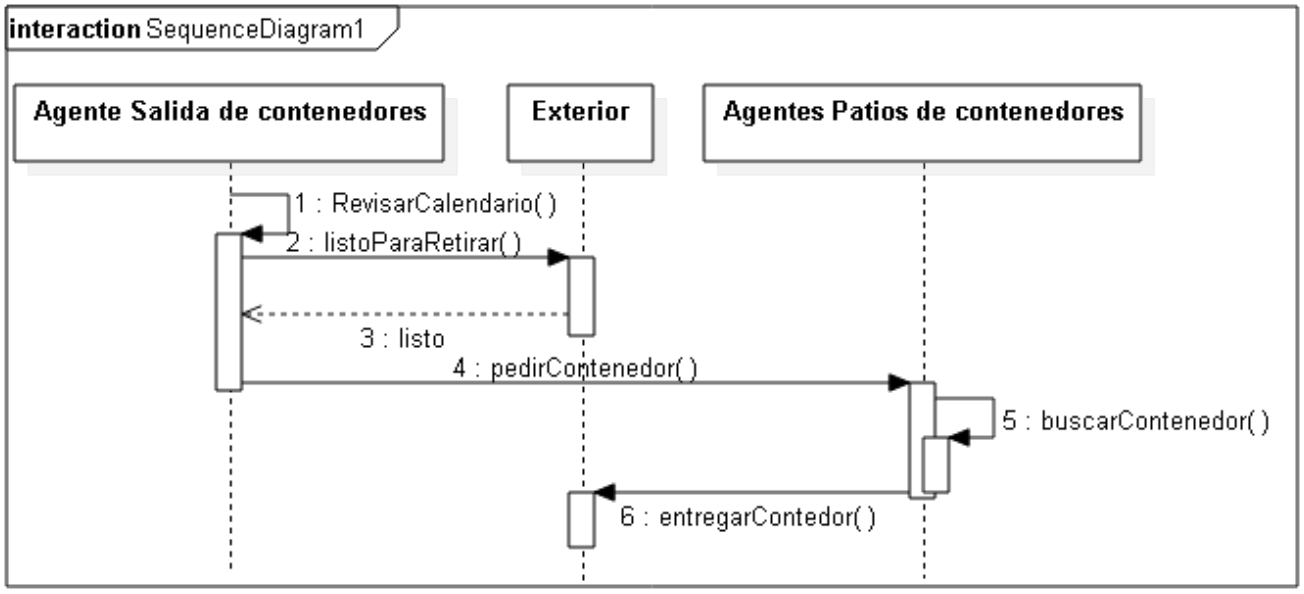

Figura 4.4 Modelo PASSI de retiro programado de contenedores de los patios

En la figura 4.4 se presenta el caso normal de retiro de contenedores, en el que salen a la fecha indicada previamente al almacenar el contenedor, primero preguntando que esté disponible el barco o camión para su retiro.

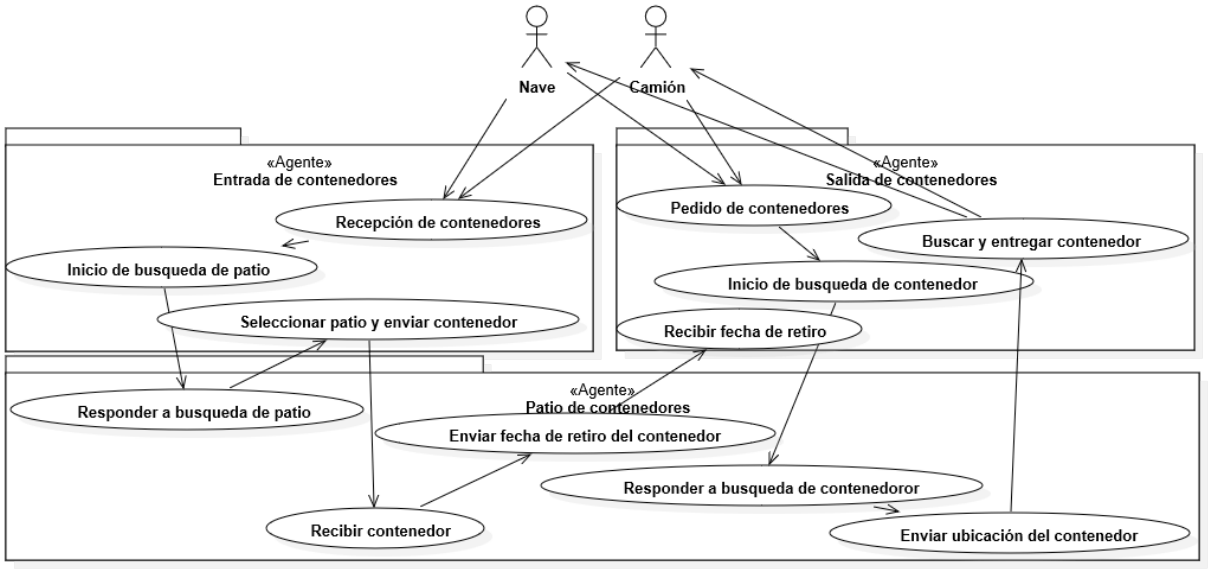

Figura 4.4 Modelo de identificación de agentes.

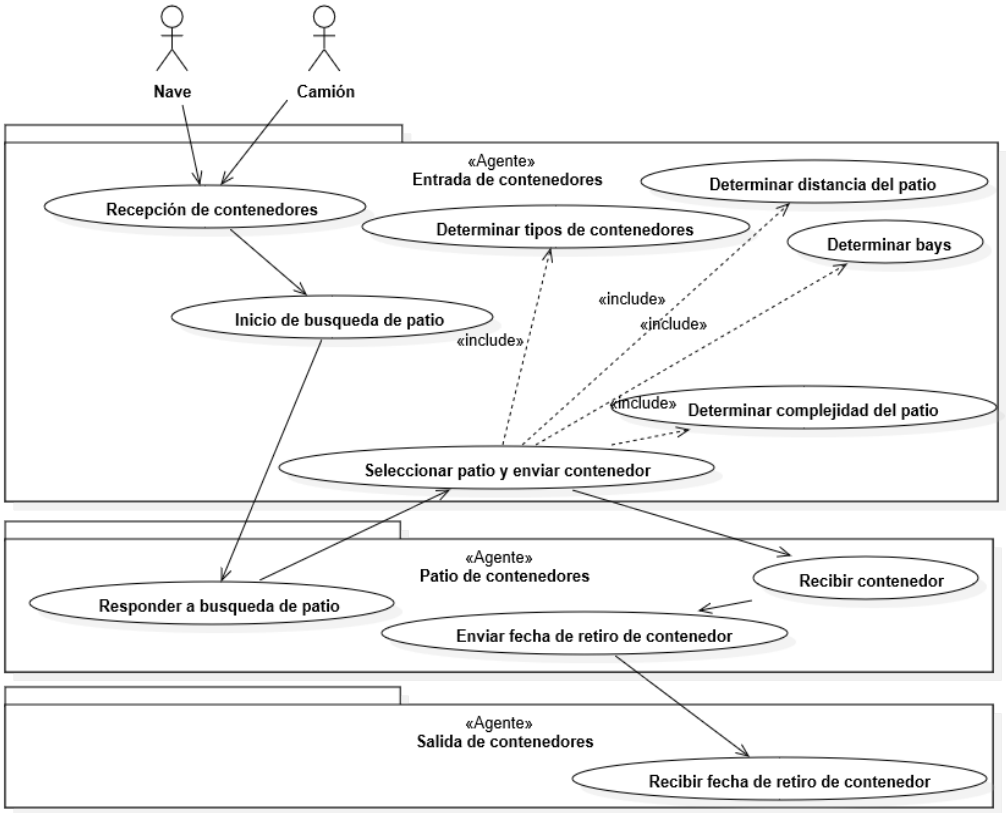

Figura 4.5 Diagrama de casos de uso detallado del sistema, *import sector*

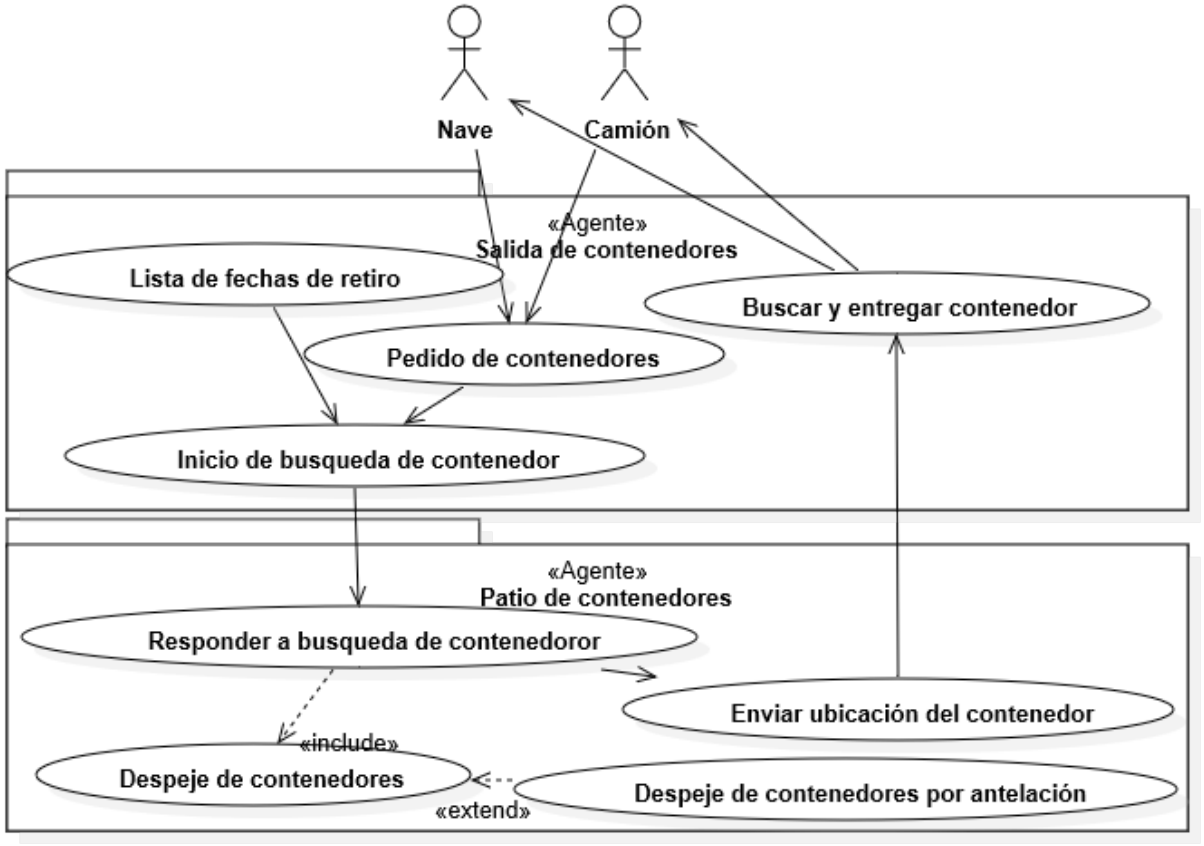

Figura 4.6 Diagrama de casos de uso del sistema, *export sector*

En las figuras 4.4, 4.5 y 4.6 se presentan los modelos de identificación y casos de uso del sistema, detallando entonces el funcionamiento interno del sistema final.

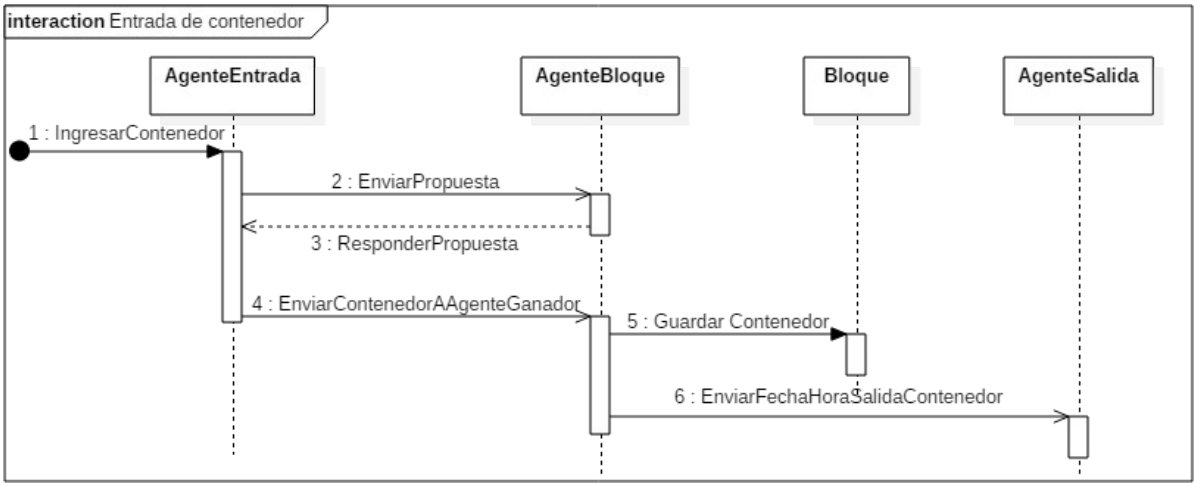

Figura 4.7 Diagrama de secuencia de ingreso de un contenedor

Finalmente, en la figura 4.7 se detalla el ingreso de un contenedor al sistema, específicamente como los agentes colaboran en este proceso.

4.1 Consideraciones adicionales

Solo un modelo de los agentes del sistema no es suficiente para poder llegar una solución, visto la cantidad de variables diferentes que pueden afectar a ésta. Principalmente, se considera que se ocupan grúas RGT en el puerto a simular, considerando la menor cantidad de de movimientos, o despejes, que éstos necesitan para extraer el contenedor objetivos, que cada despeje dura en promedio 30 segundos y que estas grúas tienen una velocidad promedio de 30 km/h, u 8,33 m/s. Se considera, además, que contenedores con el mismo destino se apilan en el mismo bay y la distancia entre el centroide y el punto de atraque se toma linealmente debido a la falta de datos para hacerlo mediante Manhattan (movimientos con solo giros en de 90°).

4.2 Función objetivo

Para poder llegar a una solución satisfactoria es también necesario definir cuales soluciones son aceptables. Para esto, se propone una función objetivo basado en la distancia entre el punto de atraque y los patios de contenedores y los despejes para poder retirar los contenedores.

Considerando todo lo anterior, entonces:

$$T_i = \frac{M_i}{8{,}33} + 30 \times (D + 1)$$

Con:

- T : tiempo para mover un contenedor en un sitio i a su navío objetivo, en segundos
- M : distancia entre el punto de atraque del navío y el centroide del sitio i
- D : cantidad de despejes necesarios para remover el contenedor.

Como M y D son determinados al posicionar el contenedor en un patio de contenedores, se puede buscar la solución óptima en función de la distancia del sitio y que tan "ocupado" esta el sitio, dado que un sitio menos lleno necesita menos despejes. Dado que se prioriza que cada contenedor en una misma columna (mismo *bay y row)* tienen el mismo destino, y por lo tanto son despejados al mismo tiempo, se puede decir que hay un despeje más por cuantos más destinos haya que columna.

Por lo tanto, la función objetivo final es:

$$T = Min\left(\frac{M_i}{8{,}33} + 30 \times \left(\left(\frac{G}{C_i}\right) + 1\right)\right)$$

Donde:

- G : Cantidad de grupos de contenedores con el mismo destino
- C_i : Cantidad de columnas del sitio i

5 Implementación del proyecto

En un principio se proponía primero implementar el sistema en la memoria de Maturana, pero debido razones de fuerza mayor se procedió directamente a implementar el sistema anteriormente detallado desde cero.

Se procedió entonces a codificar, en orden, las estructuras de datos que el proyecto vaya a necesitar, entrada y salida de datos, los agentes inteligentes en sí y las funciones que estos necesiten.

5.1 Estructuras de datos

Existen tres clases de datos principales usados para el manejo de la información del sistema, Contenedor, que contiene los datos de un contenedor individual, Bloque que describe los contenidos de un almacén de contenedores en el patio en el formato BAROTI, y sus distancias con los sitios de atraque, y finalmente la clase Baroti que describe un conjunto de coordenadas BAROTI mas las funciones para su manejo.

5.2 Entrada y salida de datos

Actualmente, todas las entradas de datos son por archivos de texto en formato CSV, detallando el puerto y los contenedores a procesar. Salidas mientras tanto son por consola, y pero se puede implementar salida a archivo de texto. Las funciones de procesamiento de archivos de texto están contenidas en sus propia clase, Archivos. Además, se puede usar la GUI de JADE para enviar mensajes al AgenteSalida para el retiro manual e inmediato de contenedores, enviando un mensaje con *performative REQUEST* con el código del contenedor.

5.3 Agentes Inteligentes

Ya detalladas las clases de datos, se empezó entonces a codificar el agente inteligente AgenteBloque, que hace de contractor en el modelo ContractNet. Cada AgenteBloque contiene un elemento Bloque con los detalles del espacio físico de este en sí, las funciones necesarias para manipular el bloque, segregación de contenedores y determinación si el bloque es adecuado y finalmente de mensajería entre agentes. Además, cada vez que se ingresa un contenedor a los bloques se envía un mensaje a AgenteSalida detallando la fecha y hora de salida del contenedor.

También se codifica el agente AgenteEntrada, que ser el *contractor* en el modelo ContractNet. Este recibe la lista de contenedores a ingresar desde el AgenteInicio y se encarga de proponer su ingreso a cada AgenteBloque.

Finalmente, se construye el AgenteSalida, que remueve contenedores de los bloques tanto según mensajes enviados a este o a través de una lista detallando a que fecha y hora cada contenedor debe ser removido, la que es revisada cada tres minutos.

5.4 Clases adicionales

Adicionalmente, existe un último agente inteligente, AgenteInicio, que se encarga de inicializar el sistema a través de usar la clase Archivos para la obtención de los datos de inicio y de inicializar AgenteEntrada, AgenteSalida y los AgentesBloque. Ya completada la inicialización del sistema, este agente es removido. La necesidad de hacer la inicialización del sistema a través de un agente es debido a limitaciones de la plataforma JADE.

5.5 Diagrama de clases

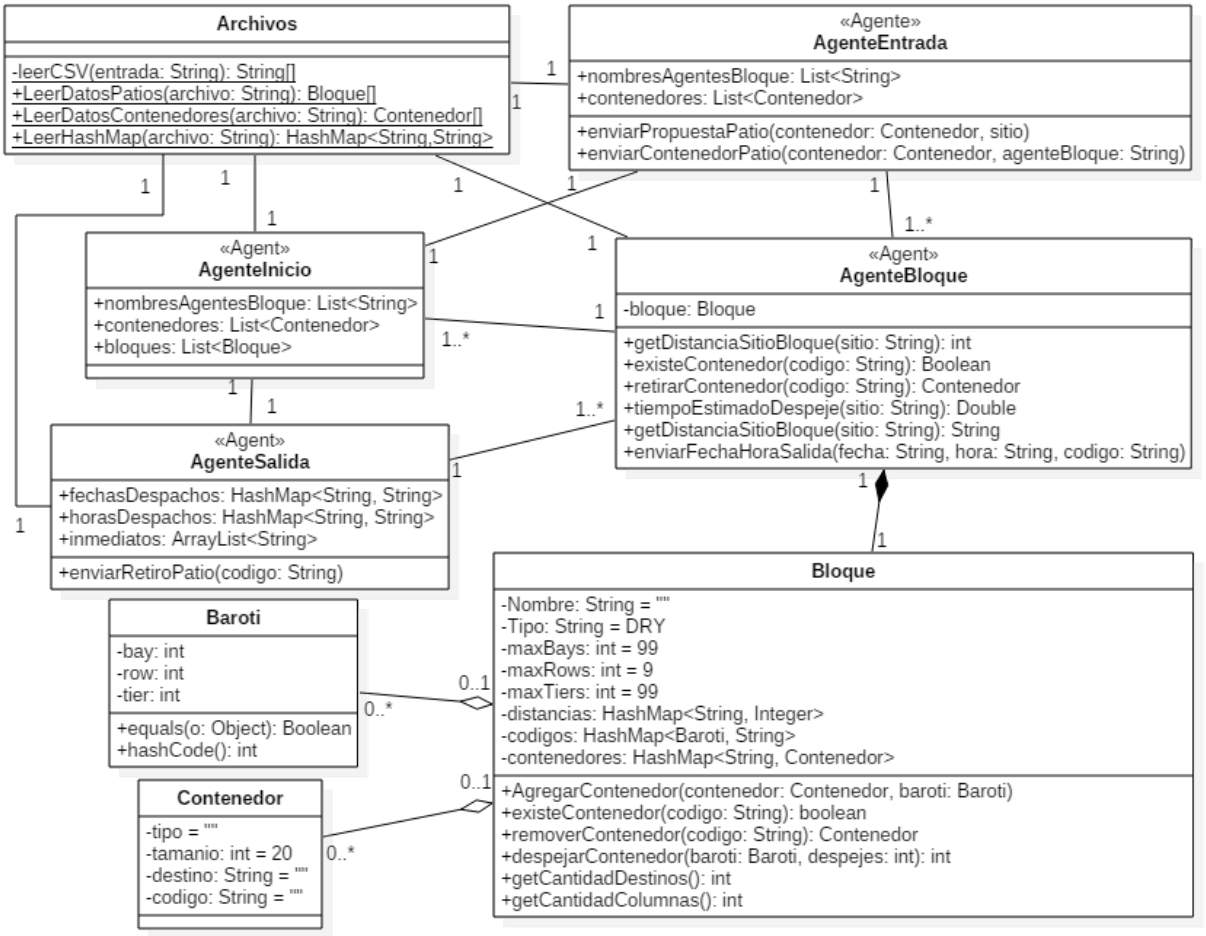

Figura 5.1 Diagrama de clases del proyecto

5.6 Ejemplos de código

De bloque.java:

```
private String nombre = "";
private String tipo = "DRY";  //DRY, REEFER, IMO
private int maxBays = 99;
private int maxRows = 99;
private int maxTiers = 99;
private HashMap<String, Integer> distancias = new HashMap<String,
Integer>(); //respecto a los sitios
private HashMap<Baroti, String> codigos = new HashMap<Baroti, String>();
```

```
      //lista elementos Contenedor
private HashMap<String, Contenedor> contenedores = new HashMap<String,
Contenedor>();

public void agregarContenedor(Contenedor contenedor, List<Integer> baroti){
    if( !(codigos.containsValue(contenedor.getCodigo()) ||
codigos.containsKey(baroti)) ){
        codigos.put(baroti, contenedor.getCodigo());
        contenedores.put(contenedor.getCodigo(),contenedor);
    }
}


public int despejarContenedor(Baroti baroti, int despejes){
    String codigo = codigos.get(baroti);
    if (codigo == null) return despejes;
    //empujar contenedores hacia abajo
    codigos.remove(baroti);
    Baroti nuevoBaroti = baroti;
    nuevoBaroti.setTier(baroti.getTier() - 1);
    codigos.put(nuevoBaroti, codigo);
    //fin empujar contenedores
    Baroti barotiArriba = baroti;
    barotiArriba.setTier(baroti.getTier() + 1);
    return despejarContenedor(barotiArriba, despejes+1);
}
```

De Contenedor.java

```
private String tipo = ""; //DRY, REEFER, IMO //OpenYard use codigos
diferentes, pero parecen ser todos DRY
private int tamanio = 20;
private String destino = "";  //cliente_retira
private String codigo = ""; //Numero en OpenYard
private String ubicacion = "";
private String fechaSalida = "";
private String horaSalida = "";
```

De Baroti.java

```
private int bay = -1;
private int row = -1;
private int tier = -1;

@Override
public boolean equals(Object o){
    if (!(o instanceof Baroti)) return false;
    Baroti b = (Baroti) o;
    if (this.bay == b.getBay() && this.row == b.getRow() && this.tier ==
b.getTier()){
        return true;
    } else {
        return false;
    }
```

```
}

@Override
public int hashCode(){
    return this.bay*961 + this.row*31 + this.tier;
}
```

De AgenteBloque.java

```
public double tiempoEstimadoDespeje(String sitio){
    if (sitio == ""){
        return 0; //existe error
    }
    float distancia = this.bloque.getDistancias().get(sitio); //entre
sitioAtraque y patio
    int cantidadDestinos = (this.bloque.getCantidadDestinos() < 1) ? 1 :
this.bloque.getCantidadDestinos();
    int despejes = this.bloque.getCantidadColumnas()/cantidadDestinos; //era
get cantidad bays
    return (distancia/8.33)+30*(despejes+1);
}
```

De AgenteEntrada.java

```
public void enviarPropuestaPatio(Contenedor contenedor, String sitio) throws
IOException{
    ACLMessage mensaje = new ACLMessage(ACLMessage.PROPOSE);
    Object[] datos = new Object[2];
    datos[0] = contenedor;
    datos[1] = sitio;
    mensaje.setContentObject(datos);
    for (String nombre : this.nombresAgentesBloque){
        mensaje.addReceiver(new AID(nombre, AID.ISLOCALNAME));
    }
    send(mensaje);
    System.out.println("Enviado propuestas de guardado de contenedor : " +
contenedor.getCodigo());
}

public void enviarContenedorPatio(Contenedor contenedor, AID agenteBloque){
    ACLMessage mensaje = new ACLMessage(ACLMessage.INFORM);
    try {
        mensaje.setContentObject(contenedor);
        mensaje.addReceiver(agenteBloque);
        send (mensaje);
        System.out.println("Enviado contenedor a patio : " +
contenedor.getCodigo() + "," + agenteBloque.getName());
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

De Archivos.java

```
private static String[] leerCSV(String entrada){
    Path archivo = Paths.get(entrada);
    try {
        return Files.readAllLines(archivo).toArray(new String[0]);

    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return null;
}

//determina bloques y sitios de atraque del patio
//distancias separadas por ;
public static Bloque[] leerDatosPatio(String archivo){
    Bloque[] retorno = null;
    String[] lineas = leerCSV(archivo);
    String[] data = null;
    String sitios = lineas[0]; //innecesario?
    for (int i = 1; i < lineas.length; i++) {
        data = lineas[i].split(",");
        retorno[retorno.length] = new Bloque(data[0], data[1],
            Integer.parseInt(data[2]), Integer.parseInt(data[3]),
            Integer.parseInt(data[4]), sitios, data[5].replace(';',','));
    }
    return retorno;
}

public static Contenedor[] leerDatosContenedores(String archivo){
    Contenedor[] retorno = null;
    String[] lineas = leerCSV(archivo);
    String[] data = null;
    for (int i = 0; i < lineas.length; i++) {
        data = lineas[i].split(",");
        retorno[retorno.length] = new Contenedor(data[0],
Integer.parseInt(data[1]), data[2], data[3]);
    }
    return retorno;
}
```

De AgenteSalida.java

```
public void enviarRetiroPatio(String codigo){
      System.out.println("Enviando mensajes para retiro de contenedor : " +
codigo);
      ACLMessage mensaje = new ACLMessage(ACLMessage.REQUEST);
      mensaje.setContent(codigo);
      for (String nombre : nombresAgentesBloque){
            mensaje.addReceiver(new AID(nombre, AID.ISLOCALNAME));
      }
      send(mensaje);
}
```

6 Resultados

La idea original de la construcción de este sistema fue la de comparar el tiempo necesario para procesar una cantidad determinada de contenedores, específicamente el ingreso de este. Debido a razones de fuerza mayor esto no ha sido posible, entonces quedando solo las métricas del sistema aquí detallado.

Tiempo de inicialización: 4 segundos
Tiempo para el procesamiento de 50 contenedores (datos de ejemplo): 6 segundos
Tiempo promedio para el ingreso de un contenedor: 0,12 segundos

6.1 Datos de entrada

Se usa una serie de 50 contenedores que fueron ingresados continuamente, con datos provenientes del puerto de Arica en un espacio de tiempo escogido aleatoriamente. Los datos son ingresados en formato CSV, y se ocupa el punto como separador decimal.

6.2 Ejemplos de resultados

Ejemplo de ingreso de contenedores a un bloque:

```
0,0,0
0,1,0

0,0,0
0,2,0
0,1,0

0,0,0
0,2,0
0,1,0
0,2,1

0,0,0
0,1,1
0,2,0
0,1,0
0,2,1

0,0,0
0,1,1
0,2,2
0,2,0
0,1,0
0,2,1
```

Ejemplo de ingreso de contenedores al sistema

```
Contenedor@1ee46fb enviado a agenteBloque_bloque01@FG-2:1099/JADE
Contenedor@510de0 enviado a agenteBloque_bloque02@FG-2:1099/JADE
Contenedor@69aa38 enviado a agenteBloque_bloque00@FG-2:1099/JADE
Contenedor@dd2fac enviado a agenteBloque_bloque00@FG-2:1099/JADE
Contenedor@1ff1f77 enviado a agenteBloque_bloque00@FG-2:1099/JADE
Contenedor@e33911 enviado a agenteBloque_bloque00@FG-2:1099/JADE
Contenedor@1c0cd01 enviado a agenteBloque_bloque00@FG-2:1099/JADE
Contenedor@15fd726 enviado a agenteBloque_bloque00@FG-2:1099/JADE
Contenedor@64406 enviado a agenteBloque_bloque01@FG-2:1099/JADE
Contenedor@8645c7 enviado a agenteBloque_bloque00@FG-2:1099/JADE
Contenedor@19706b1 enviado a agenteBloque_bloque00@FG-2:1099/JADE
Contenedor@1081116 enviado a agenteBloque_bloque00@FG-2:1099/JADE
Contenedor@9e3e72 enviado a agenteBloque_bloque01@FG-2:1099/JADE
Contenedor@117638f enviado a agenteBloque_bloque00@FG-2:1099/JADE
Contenedor@ea6da2 enviado a agenteBloque_bloque00@FG-2:1099/JADE
Contenedor@14b8600 enviado a agenteBloque_bloque00@FG-2:1099/JADE
Contenedor@1fa50c1 enviado a agenteBloque_bloque00@FG-2:1099/JADE
Contenedor@1feb0e9 enviado a agenteBloque_bloque01@FG-2:1099/JADE
Contenedor@14e4f8b enviado a agenteBloque_bloque00@FG-2:1099/JADE
Contenedor@c145cd enviado a agenteBloque_bloque00@FG-2:1099/JADE
Contenedor@690045 enviado a agenteBloque_bloque00@FG-2:1099/JADE
Contenedor@4e6935 enviado a agenteBloque_bloque00@FG-2:1099/JADE
Contenedor@53b20d enviado a agenteBloque_bloque01@FG-2:1099/JADE
Contenedor@10459b2 enviado a agenteBloque_bloque00@FG-2:1099/JADE
Contenedor@1124f8d enviado a agenteBloque_bloque00@FG-2:1099/JADE
Contenedor@1a52cd4 enviado a agenteBloque_bloque01@FG-2:1099/JADE
```

Ejemplo de registro de contenedores para despachos

```
Despacho: MSCU353438-8 01-01-2011 17:00
Despacho: MSCU355602-6 05-01-2011 15:53
Despacho: MSCU343660-6 01-01-2011 17:04
Despacho: MSCU136529-4 01-01-2011 17:09
Despacho: MEDU819234-5 02-01-2011 23:43
Despacho: INKU227198-1 02-01-2011 23:43
Despacho: TRLU547556-2 02-01-2011 23:43
Despacho: MSCU883591-2 02-01-2011 23:43
Despacho: MSCU802048-4 02-01-2011 23:43
```

7 Conclusión

Ya presentado el estado del arte de la resolución del Yard Allocation Problem, una vez determinadas las herramientas a usar para nuestra propia resolución del problema y siendo mostrado el modelo de referencia sobre el que el nuestro tiene como objetivo mejorar, se concluye que es posible construir un mejor sistema, más eficiente y simple.

Dicho de otro modo, se puede decir que a pesar de las similaridades con el modelo existente debido al uso del protocolo Contract Net, se puede estimar útil proceder con el proyecto debido a que todavía se puede investigar y desarrollar espacio para mejoras.

Se ha elegido la resolución por sistema multiagente debido a su relativa simplicidad de implementación especialmente comparado con otros métodos ocupados en la resolución del YAP, en conjunto con el uso del protocolo anteriormente mencionado.

Ahondando en el desarrollo del sistema, se destacan las mejoras respecto al sistema implementado por Kevin Maturana, estas siendo una menor cantidad de agentes necesarios para la implementación de este, un sistema mas abstracto y mas sencillo de adaptar a un puerto cualquiera, y el uso de la fórmula de estimación de tiempo de despeje, detallada anteriormente como función objetivo.

Además, se presentan la consideraciones del sistema construido para poder llegar a una solución adecuada a la realidad, considerándose un puerto usando solo grúas RGT, aproximando las distancias a una línea entre los sitios y los puntos de atraque, además determinándose el cómo del agrupamiento de contenedores para despejes más rápidos, y finalmente, una función objetivo adecuada al sistema.

Finalmente, se dan a conocer los detalles del funcionamiento del sistema ya construido.

7 Referencias

[Mas, 2005] Ana Mas, "Agentes Software y Sistemas Multiagente: Conceptos, Arquitecturas y Aplicaciones", Pearson - Prentice Hall, ISBN 84-205-4367-5
[SRI, 1999] http://www.ai.sri.com/pkarp/xol/xol.html
[UMD, 2001] https://www.cs.umd.edu/projects/plus/SHOE/
[W3C 1,2001] https://www.w3.org/TR/daml+oil-reference
[W3C 2, 2012] https://www.w3.org/OWL/
[UMBC, 1994] http://www.csee.umbc.edu/csee/research/kqml/papers/kbkshtml/kbks.html
[FIPA 1, 2002] http://www.fipa.org/repository/aclspecs.html
[FIPA 2, 2002] http://www.fipa.org/specs/fipa00029/SC00029H.html
[FIPA 3, 2001] http://www.fipa.org/specs/fipa00031/XC00031F.html
[OMG, 1997] http://www.omg.org/cgi-bin/doc?orbos/97-10-05.pdf
[FIPA 4, 2016] http://www.fipa.org/
[JADE, 2016] http://jade.tilab.com/
[WiniKoff et al, 2011] Michael Winikoff, Hanno-Felix Wagner, Thomas Young, Stephen Cranefield, Roger Jarquin, Guannan Li, Brent Martin, Rainer Unland, "Agent-based Container Terminal Optimisation", The Information Science Discussion Paper Series, ISSN 1177-455X
[Transtad, 2015] http://atanchor.transtad.nl/wp-content/uploads/2015/11/2_Reach_Stacker_fuer_Leer_Container.jpg
[Liebherr, 2015] https://www.porttechnology.org/images/uploads/equipment_products/Rubber_Tyre_Gantry_Crane_-_Liebherr_RTG-300x400.jpg
[Chen et al, 2002] Ping Chen, Zhaohui Fu, Andrew Lim, "The Yard Allocation Problem", Department of Computer Science, National University of Singapore, American Association For Artificial Intelligence
[Chen et al 2, 2002] Ping Chen, Zhaohui Fu, Andrew Lim, "Using Genetic Algorithms To Solve The Yard Allocation Problem", Department of Computer Science, National University of Singapore
[Elorrieta, 2011] Julián Elorrieta, “Planificación y Gestión de Operaciones Portuarias en Patio de Contenedores”, Pontificia Universidad Católica de Valparaíso, Memoria para optar al título de Ingeniero Civil Industrial, 2011.
[Maturana, 2014] Kevin Maturana, "Sistema Multiagente Para El Problema De Asignación De Patio", Pontificia Universidad Católica de Valparaíso, Memoria para optar al título de Ingeniero de Ejecución en Informática, 2014.