Construcción de un sub-sistema de software que permita generar movilidad de agentes artificiales de acuerdo con el rol que ellos desempeñan sobre el Sistema TLÖN cuyos recursos se comparten a través de una red Ad Hoc.

Germán Darío Álvarez Rodríguez

Universidad Nacional de Colombia
Departamento de Ingeniería de Sistemas e Industrial
Bogotá, Colombia
2017

Construcción de un sub-sistema de software que permita generar movilidad de agentes artificiales de acuerdo con el rol que ellos desempeñan sobre el Sistema TLÖN cuyos recursos se comparten a través de una red Ad Hoc.

Germán Darío Álvarez Rodríguez

Trabajo de grado presentado como requisito parcial para optar al título de:
Magister en Ingeniería - Telecomunicaciones

Director:
Ph.D., Jorge Eduardo Ortiz Triviño

Línea de Investigación:
Computación Aplicada y Sistemas Inteligentes
Grupo de Investigación:
TLON

Universidad Nacional de Colombia
Departamento de Ingeniería de Sistemas e Industrial
Bogotá, Colombia, 2017

Si descartas lo imposible, inmutable e intangible. Lo que queda es la verdad, por improbable que parezca - Sir Arthur Conan Doyle

Para todas aquellas personas que me acompañaron y me apoyaron, que entendieron que todo aquello que se hace en busca de la verdad, siempre estará más allá del bien y del mal... Que algunos sólo queremos respuestas y vemos nuestra vida como un camino para obtenerlas.

Para mis padres, mi hermana, mi sobrina y mi novia.

Resumen

En este documento se presenta el diseño, desarrollo e implementación del sub-sistema de movilidad **BEAMS** (Better Environment for Agent Mobility Subsystem) para agentes artificiales como un componente del sistema TLÖN, un sistema de cómputo virtualizado basado en los comportamientos de agentes. Un modelo de agente, ambiente y sistema multi-agente son creados con el fin de lograr y probar la movilidad de BEAMS. Como escenario de prueba de desplegó el Agente Móvil cuya meta fue capturar la información de los recursos computacionales que poseen los nodos para una movilidad física y virtual por las cuales este ha migrado; En el escenario virtual fueron medidos los tiempos de movilidad para un incremento lineal del tamaño de dicho agente, logrando ver la respuesta del sub-sistema ante diferentes tamaños de agente.
Palabras clave: Agente Móvil, Ad-Hoc, Sistemas Distribuidos, Agentes Inteligentes.

Abstract

This document presents the design, development and implementation of the mobility subsystem **BEAMS** (Better Environment for Agent Mobility Subsystem) for artificial Agents as a component of the TLÖN system, a computer system Virtualized based on the behavior of Agents.A model of Agent, Environment and Multi-Agent System are created in order to achieve and test the mobility of BEAMS.As a test scenario deployed the Mobile Agent whose goal was to capture the information of the computational resources possessed by the nodes for a physical and virtual mobility through which this has migrated, in virtual scenario were measured the mobility times for a linear increase in the size of said agent, managing to see the response of the sub-system to different sizes of Agents **Keywords: Mobile Agent, Ad-Hoc, Distributed Systems, Intelligent Agents**

Contenido

CAPÍTULO 1

Introducción

En general cuando se piensa en comunicar dos dispositivos o servicios, lo primero en lo que se piensa es un esquema cliente-servidor, esto se debe a que es el esquema más utilizado y acogido en diferentes sistemas. Sin embargo, para estructuras distribuidas, aunque este es totalmente funcional, existen alternativas que pueden dar un mejor ajuste a las necesidades de estas.

Los Agentes Móviles representan un paradigma para el mantenimiento e implementación de sistemas distribuidos [26], siendo estos Agentes una porción de código independiente de plataforma con características específicas que permite moverse, actuar y ejecutarse a través de la red [60]. Los Agentes no pueden verse como un cuerpo independiente y aislado, todo lo contrario, estos son situados dentro de un ambiente el cual es el encargado de brindar los recursos necesarios para que el mismo pueda ejecutar sus funciones [16].Un Agente puede convivir junto a otros en el mismo nodo u mismo ambiente, y es precisamente esta interacción lo que forma un Sistema multi-agente [19].

La principal razón y beneficio que otorgan los Agentes Móviles se refleja en la reducción de carga sobre la red gracias su características de ejecución asíncrona [28] [49]. Puesto que los mismos son orientados a metas no requieren una conexión permanente u envío constate de información sobre la red, esto permite desplegar un Agente Móvil con una tarea asignada sobre la red y esperar que el mismo la termine [44].

La pregunta que surgió es: ¿Qué tipo de sistema puede dar uso de los beneficios de un Agente Móvil?, y es allí donde se hace una introducción a las redes Ad-Hoc. Las redes Ad-Hoc son una colección de nodos participantes que se conectan entre sí a través de enlaces inalámbricos, al no existir un ente central cada nodo debe cumplir la función de enrutador y nodo al mismo tiempo [44]. Este tipo de redes provee un gran número de beneficios como su rápido despliegue al no necesitar una infraestructura previa, con estos beneficios también vienen

inconvenientes ó retos a superar, entre los cuales se encuentran: Anchos de banda limitados, autonomía de los dispositivos, estabilidad de la red y auto-organización [64].

Dicho esto, la programación de Agentes Móviles sobre redes Ad-Hoc se torna promisoria en el objetivo de mitigar algunos de los retos que estas presentan y obtener sus beneficios. En el desarrollo de este trabajo construyó el sub-sistema que permite generar la movilidad de Agentes de software, su integración con el ambiente de ejecución y como en el mismo estos agentes pueden ejecutar tareas moviéndose a través de la red. Para el desarrollo de la solución se utilizó el lenguaje de programación Python, el cual por su flexibilidad, plataforma independiente, integración y alta compatibilidad con dispositivos embebidos como Raspberry PI es elegido para la construcción del Ambiente, el sistema multi-agente y desde luego el sub-sistema que permitió la movilidad de los Agentes que lo componen.

Finalmente se logró que los Agentes creados para el sistema multi-agente posean la capacidad de movilidad, vista esta como el conjunto de protocolos, arquitectura, puertos, servicios e interacciones que permitió a los mismos realizar sus acciones en diferentes espacios y migrarse a estos a través de la red.

CAPÍTULO 2

Antecedentes y justificación

Generalmente los sistemas de red implican un software en el cliente y del lado del servidor, los cuales son dependientes entre sí para un esquema completo de funcionamiento. Por esta razón la programación de agentes móviles se torna cada vez más atractiva para la implementación y mantenimiento de sistemas distribuidos, que brinda una alternativa diferente al esquema cliente-servidor [26].

Durante el desarrollo del documento se describe como los Agentes Móviles han surgido como una opción para la construcción de aplicaciones sobre sistemas distribuidos y sistemas con baja capacidad de recursos, sus ventajas y como los mismos pueden coordinarse formando sistemas complejos y eficientes para la resolución de tareas y cumplimiento de metas. El concepto de Agente Móvil es primordial ya que el sub-sistema que ha sido desarrollado es la esencia de estos Agentes.

En las siguientes secciones se da inicio a la definición del lenguaje de programación, Agente Móvil, sus componentes y características con el objetivo de clarificar como la movilidad es un factor de influencia sobre este tipo de Agentes.

2.1. Python: las raíces del sub-sistema de movilidad

Actualmente existen dos versiones de Python: Python 2 y su más reciente versión Python 3, con el pasar del tiempo nuevas tecnologías y nuevas ideas emergen, y sobre los lenguajes de programación no existe esta excepción. Normalmente esta mejora continua se hace de forma incremental y es difícil de notar, sin embargo Python ha tenido un gran avance entre su versión base Python 2 y su más reciente Python 3, desde luego esto puede generar que algunos programas escritos en Python 3 requieran de cambios para ser ejecutados en Python 2, sin embargo el proyecto aquí presente tiene como objetivo la versión de Python 3, no sólo por ser la más reciente y la que representa el futuro de Python sino por una serie de ventajas

y posibilidades futuras que la hacen que la misma sea la más adecuada [40].

Python es un moderno y poderoso lenguaje de programación es fácil de aprender y altamente extensible. Soporta módulos y paquetes lo cual favorece la modularidad del software y re-utilización de código. Sofisticado, pero a su vez fácil de usar e integrar con diferentes funciones y estructuras que no necesariamente deben estar escritas en Python provee una ventaja para la computación moderna y científica [15].

2.2. Movilidad: un sub-sistema del Sistema TLÖN

La razón por la que se ha referido a la movilidad como un sub-sistema es porque ha sido diseñado para hacer parte de un sistema más grande, para este caso particular se ha tomado como base la definición y organización del sistema TLÖN, un sistema de cómputo virtualizado basado en capas que con Agentes artificiales y nodos conectados a través de redes Ad Hoc. Sin embargo, el sub-sistema puede ser expandido y modificado para su uso en sistema con diferentes propósitos.

2.3. Agente Móvil

Como un lenguaje orientado a objetos, Python soporta una totalidad de características como la herencia, polimorfismo y encapsulamiento. Para el caso de este proyecto el concepto de herencia es fundamental.

Las clases y la herencia permite mejorar y extender la funcionalidad del sub-sistema a través del tiempo, brindando flexibilidad en un ambiente en que los requerimientos pueden cambiar [45]. Además de ello reduce el código duplicado y genera una abstracción más clara sobre las relaciones existentes en los participantes del sub-sistema [46].

Recordemos que en nuestros escenarios esta presente un Agente Móvil, pero ¿Qué es un Agente móvil?, en una definición corta que podría ser aceptada es la de un Agente que posee movilidad [29], en todo caso nos llevaría a definir por separado lo que es un Agente y lo que la Movilidad. Sin embargo con el fin de no prolongar esta definición y enfocarnos en lo concerniente a este capítulo la abstracción que se hace de un Agente es todo lo que está en incluido en el módulo Agents, es decir, cualquier Agente que nace o "hereda"de las clases que constituyen ese modulo puede considerarse entonces un Agente como tal dentro del contexto de este proyecto.

Es aquí donde se observa como el concepto de herencia es representado por una definición y

utilidad del lenguaje de programación, efectivamente los Agentes son creados a partir de la clase Agente del módulo Agents lo cual es afortunado ya que las características de el Agente base siempre estarán presentes en todos aquellos que aunque puedan tener una función, atributos o formas diferentes siempre en su interior serán Agentes del sistema.

2.3.1. Definición

Un Agente Móvil es un software con la capacidad de migrarse a través de los nodos para así logar tareas como respuesta a condiciones cambiantes en el ambiente por las cuales el mismo fue lanzado [13]. Ventajas como un uso efectivo de recursos, reducción del tráfico y carga sobre la red e interacción en tiempo real con su ambiente son características óptimas para las redes Ad Hoc [50][4]. Para su desarrollo es necesario contar con una plataforma denominada Mobile Agent Platform (MAP) o ambiente por otros autores la cual es necesaria para que los agentes puedan operar. Un ambiente apropiado provee para la ejecución de los agentes, seguridad, persistencia, comunicación y mensajería [44] [17] [65][10]. En un típico Sistema Multi-Agente (MAS) puede existir múltiples agencias ejecutándose sobre los diferentes nodos, [39] siendo estas una capa de abstracción distribuida que provee los conceptos y mecanismos para movilidad y comunicación entre los agentes, brindando capacidad de migración de agentes [32].

En la figura **2-1** se observa como el código del agente puede ser migrado y ejecutado en otro nodo apoyándose en la plataforma o ambiente existente en cada uno de ellos.

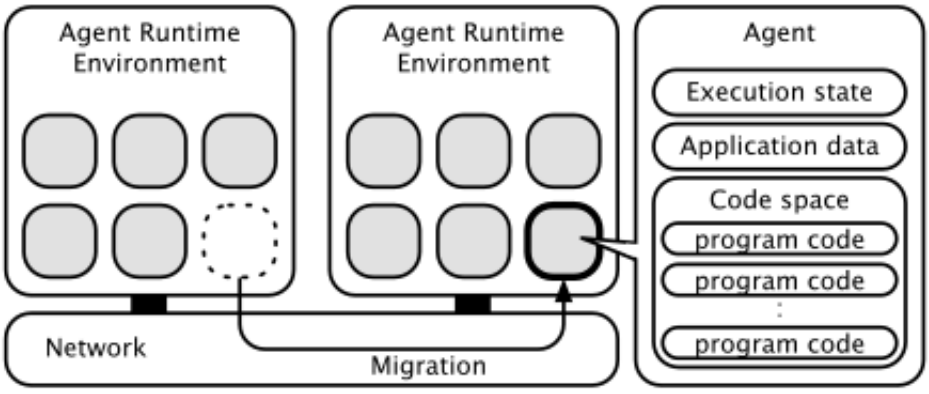

Figura 2-1: Migración de un Agente Móvil [26].

2.3.2. Identificador del Agente Móvil

El identificador del Agente o como es llamado a través del desarrollo del trabajo *AgentId* es un numero único asignado a cada agente al momento de su creación, el cual es calculado y registrado dentro del ambiente al momento de creación y proceso de registro de cada Agente perteneciente al sistema.

El AgentId es generado como Universally Unique IDentifier (UUID), los UUIDs son secuencias de (128 bits) en formato de 16 octetos relativamente corto referente a otras alternaitvas [37]. Una ve las ventajas de UUID es que no requiere un proceso central de registro y puede ser calculado fácilmente por los diferentes lenguajes, entre ellos claro está, Python.

La versión específica usada en el proyecto es la versión 4, para números aleatorios o pseudo-aleatorios. Referente a la seguridad no debe asumirse que los UUIDS son difíciles de adivinar por lo cual es importante no usar los mismos en estos procesos [37]. Como el objetivo del proyecto es el uso de los mismos como identificador único, es una solución apropiada para el sistema multi-agente. A continuación se muestra la forma y resultado para generar un UUID en Python:

```
>>> from uuid import uuid4
>>> uuid4()
UUID('d3bdac13-83ec-4ea4-8ab9-420c0b7763cb')
```

2.3.3. Características del Agente Móvil

El principio de un Agente Móvil o Agente en general fue definido con base a una Máquina de estados finitos (FSM), las máquinas de estados finitos son usadas en aplicaciones de computación y ciertamente en más cosas a nuestro alrededor de las que imaginamos. Esta e basa en un concepto simple pero eficaz de mantener estados internos los cuales recibirán entradas que permitirán el cambio de estado y a su vez la generación de una salida, como su nombre lo indica este número de estados deben ser finitos y solo se podrá estar en cada uno de ellos un tiempo a la vez [24]. La definición de Máquina de estados finitos (FSM) según Ralph Grimaldi se enuncia a continuación:

> "Una máquina de estados finitos M se define como una Quíntupla":

$$M = (S, \Upsilon, O, \nu, \omega) \tag{2-1}$$

Donde S = el conjunto de los estados internos para M
Υ = el alfabeto de entrada para M, O el alfabeto de salida para M.
ν es el siguiente estado de la función.
ω = la función de salida.

Este principio será la estructura de cada Agente, sin embargo este definirá su comportamiento con base en las percepciones del mismo. Además de esto los Agentes están constituidos de: un estado (estado de ejecución del agente), código (instrucción a ejecutar) y data (Información de las variables del agente) [17]. Por otro lado también es identificado por características tales como: Inteligencia, comunicación, autonomía, sociabilidad y movilidad, las cuales son detalladas a continuación [16] [1]:

Sociabilidad

Así como en las sociedades humanas, la comunicación es un factor que habilita la cooperación y recolección de información sobre el ambiente más allá que los intereses locales o individual. Un agente cooperativo y con habilidad social puede obtener información tanto de sus sensores individuales como de los demás agentes y el ambiente sobre el que este se ejecuta [16]. En efecto, un Agente que reconoce de su entorno y es capaz de percibir otros agentes, es un Agente comunicativo que puede valerse de eso para beneficio de sus objetivos [20][19].

Movilidad

Capacidad de migrarse a través de la red para desempeñar tareas específicas siendo esta primordial para el funcionamiento de un Agente [39]. En este orden de ideas si un agente decide bajo su definición y programación moverse hacia otro nodo, este deberá seguir el siguiente flujo con el fin de mantener sus propiedades y funciones [52].

Comunicación

Es la propiedad de intercambiar información con otros agentes dentro de su ambiente de ejecución es denominada comunicación [1]. Esta comunicación puede llevarse a cabo usando mensajes ACL, los cuales son de propósito para la comunicación entre agentes [3].

Inteligencia

Una de las particularidades y atractivos sobre el paradigma de los agentes móviles es su inteligencia, sea esta misma definida en un contexto como la capacidad de adaptarse, cambiar respecto a su ambiente basado en la información que el mismo le provee [1].

Autonomía

Es la capacidad de los mismos de controlar sus acciones, estrategias y comportamiento sin la necesidad de la interacción humana [39]. Un agente puede interactuar con otros agentes, utilizar los recursos disponibles para completar su tarea y aún así preservar su autonomía [65].

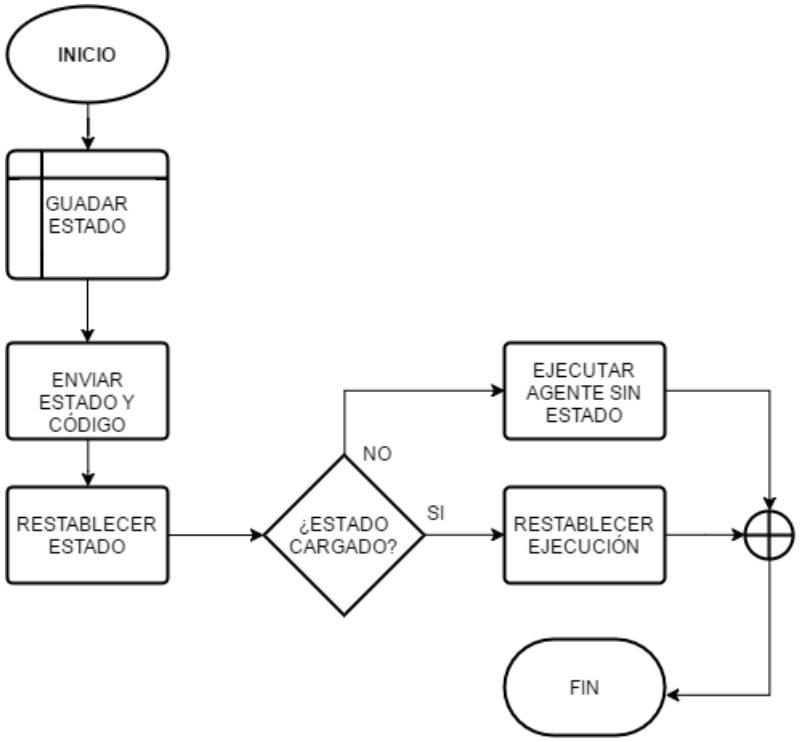

Figura 2-2: Diagrama de flujo - Agente Móvil construido con base en [52]

Cuando un Agente se mueve de un nodo a otro lleva consigo la información del nodo origen, de esta forma mantiene la información actualizada y a su vez se actualiza a sí mismo. Con el fin de que esto sea posible la información debe llevar un tiempo de referencia, así los nodos sabrán el estado del sistema respecto al tiempo, esto para el caso de los agentes que monitorean recursos o sincronizan información [14].

Los Agentes Móviles van a través de la red en busca de información, ejecutando una tarea, relacionándose con otros agentes que a su vez se mueven o permanecen dentro del mismo nodo, todo esto es posible ya que múltiples agentes pueden ser ejecutados bajo el mismo nodo [25][16]. Puesto que la característica de movilidad se convierte en la acción principal de todo Agente Móvil, surgen a su lado factores relevantes como patrón de movilidad, tiempo de ejecución, y destrucción del mismo [52].

Uno de los retos al desplegar Agentes Móviles sobre ambientes inalámbricos como lo son las MANET (Mobile Ad Hoc Network) es el patron de migración, ya que este afectará directamente el desempeño de la red. Existen grandes consecuencias al escoger este patrón de migración el cual decidirá el siguiente nodo que el Agente visitará [14].

Un agente, de acuerdo con su patrón de movilidad puede ser dividido principalmente en los siguientes tres tipos:

- Itinerario estático, orden estático
- Itinerario estático, orden dinámico
- Itinerario dinámico, orden dinámico

Sea el itinerario el conjunto de nodos que el agente deberá visitar una vez sea desplegado en la red [17].

También existe el modelo *Free roaming mobile agent* (FroMA), donde el Agente viaja a través de los nodos de la red sin una ruta definida, recorriendo los mismos de forma aleatoria [17]. Para el caso de sistemas cuyas condiciones de red son impredecibles generando rutas poco confiables este patrón de migración puede ser una buena opción a considerar, incluso algunas veces puede mostrar mejores resultados que patrones considerados más "inteligentes" [14]. Sin embargo, el siguiente nodo a recorrer puede ser decidido acorde a las características que el ambiente actual de ejecución ofrezca, haciendo uso de las propiedades inherentes del agente, tales como inteligencia y autonomía.

2.3.4. Beneficios del Agente Móvil

Considerando un Agente Móvil desplegado sobre una red de nodos existen diferentes utilidades en información que el mismo puede obtener u ofrecer, entre estas se encuentra:

1. El Agente Móvil puede capturar información sobre información de recursos computacionales como: memoria libre, librerías e información de software o hardware disponible.
2. El Agente Móvil puede ofrecer información sobre nodos que este ha visitados a Agente locales.
3. El Agente Móvil puede obtener una lista de los vecinos y elegir uno de los mismos para migrarse.
4. El Agente Móvil puede realizar una migración a través de los nodos y repetir la mismas con patrones predeterminados o cambiantes en el tiempo [23].

Los Agentes Móviles son implementados ampliamente en aplicaciones como comercio electrónico, manufactura, administración de red, sistemas de control en tiempo real, control de recurso y automatización de entornos [54][39][27]. Sin embargo, cuando se piensa en un Agente Móvil no puede pensarse en el mismo como un ente aislado, estos no pueden ser vistos como sistemas sin cuerpo, deben ser situados sobre un ambiente y es precisamente junto a este ambiente que los mismos son capaces de percibir, actuar, ser creados, ejecutados y finalmente destruidos si es el caso [16]. Por esta en la sección 2.4 se definirá un término inherente a todo Agente el cual es: el Ambiente.

2.4. Ambiente y su papel en la Movilidad

Los conceptos de Agente y Ambiente han sido considerados inseparables desde el principio de investigación de Agentes. Los Agentes no puede ser considerados independientes se su ambiente en el cual existen y a través del cual interactúan [63]. Algunas definiciones apuntan que el ambiente puede considerase como todo aquello fuera de los límites del Agente como tal y su estado cambia por la influencia de todos los agentes que se ejecutan con este, claro está, limitado por las restricciones que el mismo ambiente puede llegar a imponer.

Sin embargo, recientemente el Ambiente ha sido considerado como la primera clase de abstracción con sus propias responsabilidades fuera del Agente. Una de las principales confusiones al momento de identificar el ambiente es confundir el *ambiente* como una abstracción lógica relativa a los Agentes y sus modelos con la infraestructura necesaria para el despliegue de Sistemas Multi-Agentes, la confusión entre la abstracción teórica y lógica con la física [56].

En el contexto de los sistemas de información, una primera clase de abstracción es una pieza de software que provee el mecanismo, la interfaz para que los agentes puedan acceder a recursos y servicios que no están asignados a ellos en sí, regulando la interacción entre la comunicación por parte de los Agentes [56]. Existen servicios que corresponden solo al Ambiente, por ejemplo, un registro general de todos los Agentes, su identidad, su estado y funciones básicas del mismo serán almacenadas por el Ambiente y brindadas por el Agente. Es así como cada uno se encarga de su labor y se complementan entre sí.

El papel del Ambiente en los Sistemas Multi-Agente ha sido enfocado hacia la comunicación del Agente y como un contenedor del mismo. Sin embargo, el Ambiente posee responsabilidades tales como:

- Estructuración: al ser un espacio común para los Agentes, este podrá definir aspectos de organización como grupos, papeles, ubicaciones entre otros.
- Administrar recursos y servicios: el Ambiente actúa como un contenedor, es una entidad de control para el acceso a los servicios y recursos, en el cual el mismo puede asignar tareas o solicitudes a los Agentes que lo habitan.
- Comunicación: el Ambiente define la forma de comunicación y los mensajes a través de los cuales los Agentes se entienden y comunican unos a otros.
- Regir el Sistema Multi-Agente: el Ambiente puede definir las reglas y leyes sobre las cuales se rige el Sistema Multi-Agente [63].

El ambiente es lo que el Agente conoce y lo rige, en el caso computacional este ambiente puede sentirse como un **entorno vitual**. El diseño de este ha sido creado al igual que el

Agente como una máquina de estados finitos (FSM), de tal forma que las interacciones con los Agentes hacen que el mismo cambie sus estados internos.

2.4.1. Virtualización

En computación frecuentemente se refiere a virtualización como la abstracción de un componente físico en un objeto lógico que permite correr simultáneamente varios sistemas operativos bajo el mismo hardware, manteniendo cada objeto lógico o máquina virtual aisladas una de otra [48].

Al permitir abstraer y compartir recursos entre las diferentes máquinas virtuales [62] la virtualización genera una reducción en costos y aprovechamiento de recursos tanto para el usuario final como para el proveedor del los mismos [48] [11]. En efecto la virtualización proveerá esta base computacional a los Agentes en un nivel lógico, lo que permitirá que los mismos desde su perspectiva detecten sólo los recursos que han sido expuestos.

2.4.2. Nubes Móviles

La palabra nube usada recientemente en ambientes como *Cloud Computing, Cloud Storage, y Cloud* en general es una abstracción para un sistema que consiste en recursos distribuidos interconectados entre sí los cuales a su vez son compartidos sobre la nube para un propósito como la prestación de servicios [21].

Una nube móvil es un conjunto cooperativo de nodos dinámicos, conectados compartiendo recursos donde un nodo puede ser cualquier dispositivo con una interfaz inalámbrica, tales como celulares, tabletas, vehículos y computadores en general [21].

Dependiendo de la situación la nube móvil puede servir como una plataforma flexible para exponer un conjunto de recursos, y estos recursos son de hecho los recursos de los nodos, tales como físicos, de conectividad y aplicaciones entre otros [5]. En la figura **2-3** se observa como este conjunto de recursos confirman lo que se denomina una Nube Móvil:

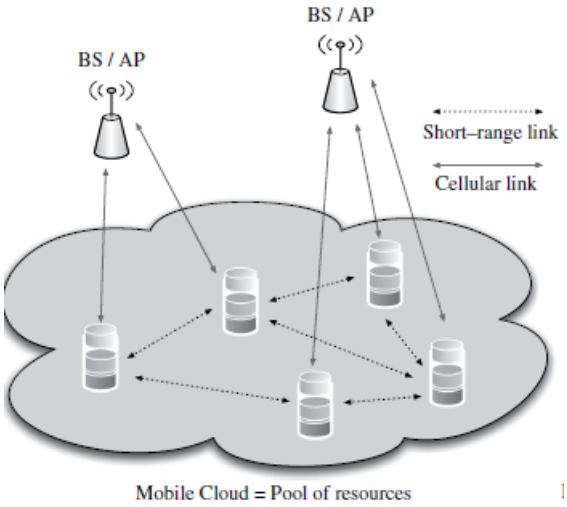

Figura 2-3: Nube Móvil = Conjunto de recursos [21]

Otro concepto que se menciona es el de sistema distribuido, el cual consiste en el uso de más de un recurso para crear un sistema [47].

Hoy día la computación de alto desempeño y las súper computadoras son amplia mente usadas en áreas de la ciencia y la industria. Sin embargo, estas computadoras requieren grandes esfuerzos y costos asociados, es por ello que los sistemas distribuidos han sido desplegados en aplicaciones como procesamiento de vídeo entre otras.

Los sistemas distribuidos permiten tener un poder de computación de alto desempeño apoyado en múltiples recursos de menores características [43]. Una de las principales necesidades para la construcción de un sistema de alto desempeño y características de computación es precisamente lograr ejecutar tareas de grandes requisitos que un computador u dispositivo normal no podría satisfacer, pero que desde luego un sistema distribuido [51].

La ejecución de tareas es una de las principales acciones para las que se construye un sistema, los Agentes pueden llevar a cabo estas actividades de forma sincrona (bloqueante) o asincrona (no bloqueante).

2.4.3. Programación Asincrónica

Existe un estilo de programación llamado asincrónico o programación no bloqueante, básicamente la diferencia inicial entre el método de programación sincrónica y asíncrona es la utilización de recursos y el bloqueo de los mismos para ejecutar una tarea. Supongamos que existen cuatro tareas a ejecutarse las figuras **2-4 2-5** muestran las mismas de forma sincrónica y asincrónica [47].

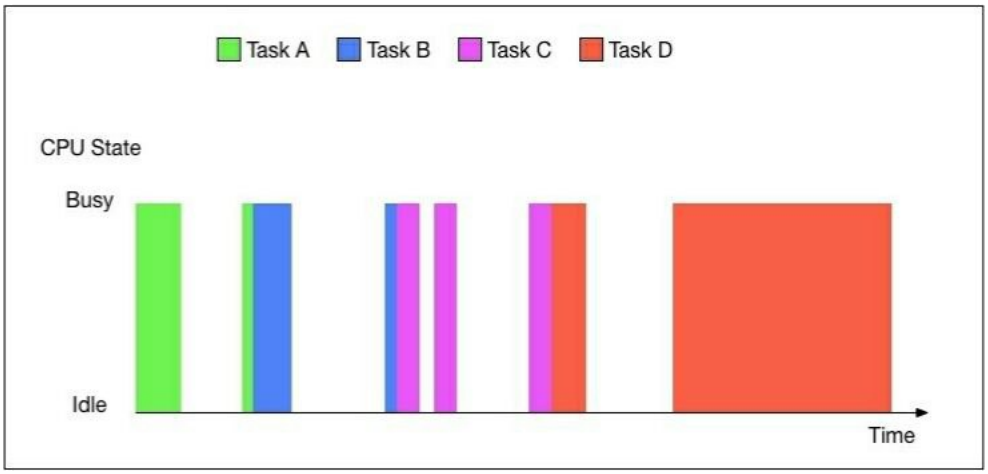

Figura 2-4: Distribución programación bloqueante [47]

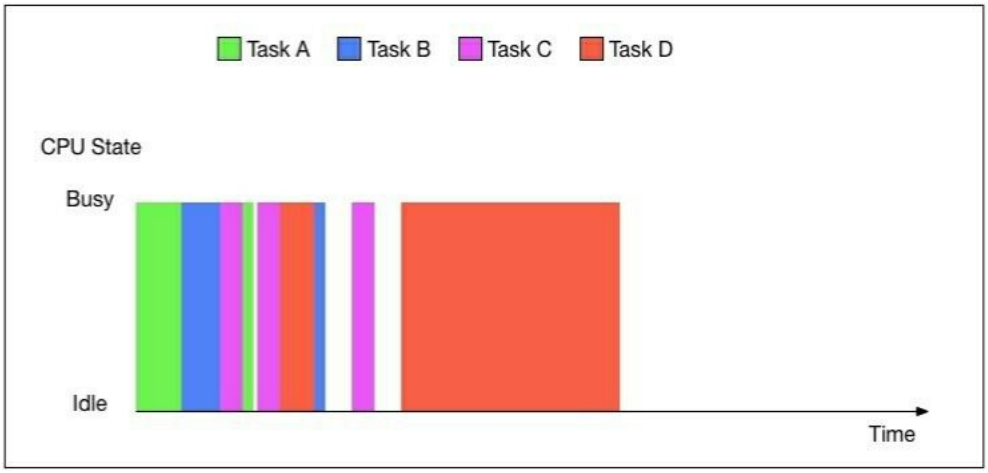

Figura 2-5: Red Ad-Hoc para movilidad nodo a nodo [47]

La progrmación asincronica no se encuentra dentro del alcance de este proyecto pero ha sido mencioanda pues la versión de Python3 que se usa para el mismo ya soporta este tipo de programación nativamente, lo que significa que en un trabajo futuro partes del sub-sistema o de todo el sistema puede usar la misma migrando o desarrolando nuevos módulos lo que lleva a una mejora de desempeño considerable.

Mientras que en la forma sincrónica los recursos de CPU por ejemplo son liberados hasta que la tarea es completada , lo que generan mayor uso de tiempo en los mismos, en la figura **2-5** se observa como en vez de bloquear y reservar la CPU hasta que cada tarea este completa todas ellas liberan la CPU cuando no la necesitan (porque están esperando por data).

Aunque aún en la figura **2-5** se observan intervalos en los que los recursos no son utilizados hay una considerable mejora en tiempo para la ejecución de las tareas. Por ello, es importante tener en la visión de un proyecto como este la posibilidad de una programación asincróna que aunque no esté en el alcance a corto plazo, puede ser planteada como una mejora de desempeño para trabajos futuros. Para el contexto de este proyecto ha sido adoptada y trabajada la definición de computación distribuida o sistema distribuido como:

> *"Computación distribuida es el uso simultaneo de uno o más computadores para resolver un problema"* [47]

2.5. Sistema Multi-Agente

Un Sistema Multi-Agente (MAS) por sus siglas en inglés, es un conjunto de Agentes individuales capaces de comunicarse, cooperar e interactuar entre sí principalmente a través del envío de mensajes. Esta comunicación permite a estos ser auto-organizados, siendo la auto-organización un enfoque común cuando los Agentes son desplegados sobre sistemas dinámicos en el tiempo. Para formar el Sistema Multi-Agente es necesario clarificar el papel que juega el Agente (como ente individual), el Ambiente, la comunicación entre los mismos y finalmente como la coordinación de estos elementos da vida a un Sistema completo.

En la sección 2.3 se ha descrito el concepto de Agente Móvil, sus principios, su ambiente y sus características principales en cuanto a fortalezas y debilidades en mayor detalle. Entre estos se mencionan los modelos de comportamiento para un Agente que consisten en una secuencia de percepción, razonamiento, procesamiento y realización de acciones discretas [8]. Por ahora sólo se usará una visión de lo que es un Agente y Ambiente con el fin de lograr posicionarlos dentro del sistema multi-agente que los contiene.

Existe diferentes tipos de Agentes pero inicialmente los Agentes desde el momento en que son creados basan sus acciones en percepciones que reciben del ambiente, y es precisamente este intercambio con el Ambiente o con otros Agentes lo que permite la colaboración y procesamiento orientados al cumplimiento de metas [7].

En la figura **2-6** se observa como el agente basado en sus percepciones provenientes del ambiente ejecuta sus acciones, como puede llegar a modificar el estado de su ambiente.

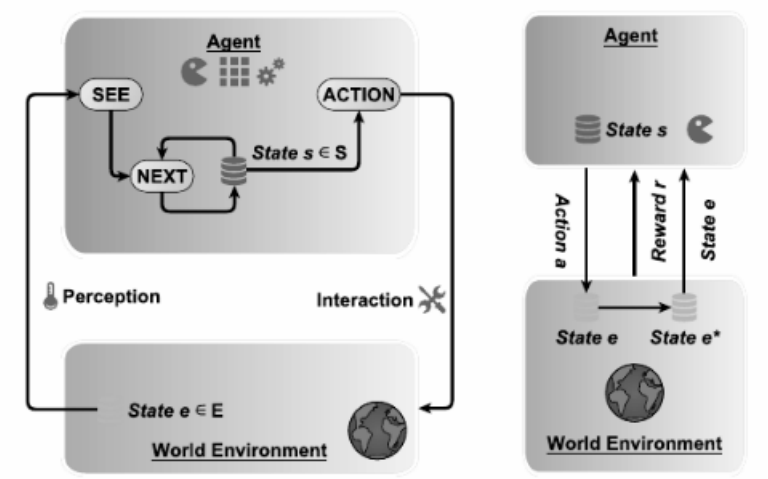

Figura 2-6: Agente interactuando con su ambiente [7]

Este conjunto de elementos, su interacción y ejecución con un propósito es la muestra de

un Sistema Multi-Agente. Ahora bien, ya que el sistema ha sido definido sólo es necesario determinar los recursos computacionales que soporten el mismo, y es allí donde la redes Ad-Hoc juegan un papel determinante al ser la base de estos recursos que son expuestos al Sistema lo que permitirá al mismos operar y de esta forma a cada Agente cumplir su propia meta.

2.5.1. Protocolos de comunicación

El sub-sistema de movilidad, cuya principal característica es la capacidad de moverse a través de la red, se apoya en protocolos de conexión dependiendo de las acciones que han sido establecidas sobre este [39].

El núcleo de este proyecto es basado en la capacidad del sub-sistema de movilidad en dar uso de los protocolos de Internet para acceder y comunicarse en red. Una de las razones por la que el lenguaje Python ha sido seleccionado para el desarrollo del proyecto es precisamente que el mismo soporta estos protocolos dentro de su librería nativa [18].

2.5.2. El protocolo TCP/IP

Cuando se construye una aplicación sobre IP, existen varias preguntas a las cuales el diseñador se enfrenta, preguntas como: ¿La aplicación deberá enfocarse en rapidez, agilidad y facilidad, o deberá tomar en cuenta la fiabilidad, trazabilidad y confiabilidad de la comunicación?. A continuación se describirán los tres mayores enfoques al construir una aplicación sobre IP:

- La mayoría de las aplicaciones del hoy día son construidas sobre TCP, el cual ofrece paquetes ordenados y confiables en un flujo de datos sobre aplicaciones de IP [30] [61].
- Unos pocos protocolos que usualmente usan un método corto de solicitud-respuesta, y clientes simples que se permiten repetir la solicitud en caso de que esta se pierda eligen UDP [59].
- Y Finalmente protocolos muy especializados evitan estas dos opciones y deciden el crear un nuevo IP-based protocol el cual define una forma totalmente nueva para conversar a través de una red IP.

TCP/IP está especificado en los documentos llamados Request for Comment (RFCs). Dentro de los RFCs existen un gran número especificaciones para el protocolo TCP/IP, para un ejemplo especifico IPv4 esta documentado en el RFC 791 [33].

2.5.3. Dirección IP

Las direcciones IP para el caso de IPv4 son un número de 32-bits el cual es normalmente escrito en formato decimal para la facilidad del usuario final. Para asignar este número a un dispositivo, las direcciones IP básicamente desempeñan dos papeles dentro de un red los cuales son mencionados a continuación [18]:

- Brindan a cada dispositivo una identificación única a nivel de dirección dentro de la red.
- Ayudan a enrutar el tráfico entres redes.

Acorde a la primera característica, cada dispositivo dentro de la red debe poseer una única dirección y no pueden compartir la misma dirección en la misma red.

2.5.4. Asignación de direcciones IP

Existen dos formas principales para la asignación de las direcciones IP: **Estática y Dinámica**. La asignación dinámica es configurada por medio el protocolo *Dynaic Host Configuration Protocol*(DHCP). Por otra parte la forma Estática es usada en redes de tamaño reducido y no se requiere un escalamiento o mantenimiento considerable [18].

2.5.5. Servicios y puertos asignados

Cuando vamos a definir los puertos para cada aplicación es importante tener claro los rangos que han sido definidos para ambos protocolos (UDP, TCP), existen tres tipos de rangos dados por la *The Internet Assigned Numbers Authority* (IANA) los cuales son:

1. "Well-Known Ports"(0-1023) : Estos son reservados para los más importantes y usados protocolos. En muchos sistemas Unix-like los usuarios no pueden usar estos puertos, así se evita conflictos y problemas con los protocolos designados para estos.
2. Registered Ports"(1024-49151): Estos no son usualmente tratados como puertos especiales por los sistemas operativos, siendo así cualquier usuario puede escribir su programa y tomar el puerto 5432 y pretender ser una base de datos PostgreSQL pero estos pueden ser registrados por la IANA para protocolos específicos.
3. Los puertos restantes (49152-65535) están libres para cualquier uso, es así como los sistemas operativos modernos usan este conjunto para generar puertos aleatorios cuando al cliente no le preocupa el puerto que se le es asignado [33].

CAPÍTULO 3

Movilidad del Agente

El propósito general del trabajo aquí presente fue generar la movilidad de Agentes. Desde luego la movilidad por sí sola aunque es un concepto sencillo esta fundamentada en una serie de procesos y funciones inherentes. Por ello, es necesario diseñar cada uno de sus componentes y la forma en que estos se comunican, los diseños presentados fueron realizados mediante SADT(Structured analysis and design technique) una técnica que ha permitido mostrar desde un diseño general hasta un diseño detallado donde han sido incluidas las diferentes tecnologías, formatos, módulos que fueron implementados y algunos que son planteados como mejoras en un trabajo futuro.

El diseño ha sido dividido en cuatro funciones: sub-sistema de movilidad, Movilidad, Itinerario y recursos, siendo estos últimos tres componentes principales del sub-sistema de movilidad. Este diseño ha sido base para la construcción del software y los escenarios de prueba. Sin embargo existen otras definiciones como los estados del Agente y la organización de la red que serán mencionadas en capítulos posteriores. En las figuras **3-13-23-3** se presenta el diseño del sub-sistema de Movilidad a través de diagramas SADT:

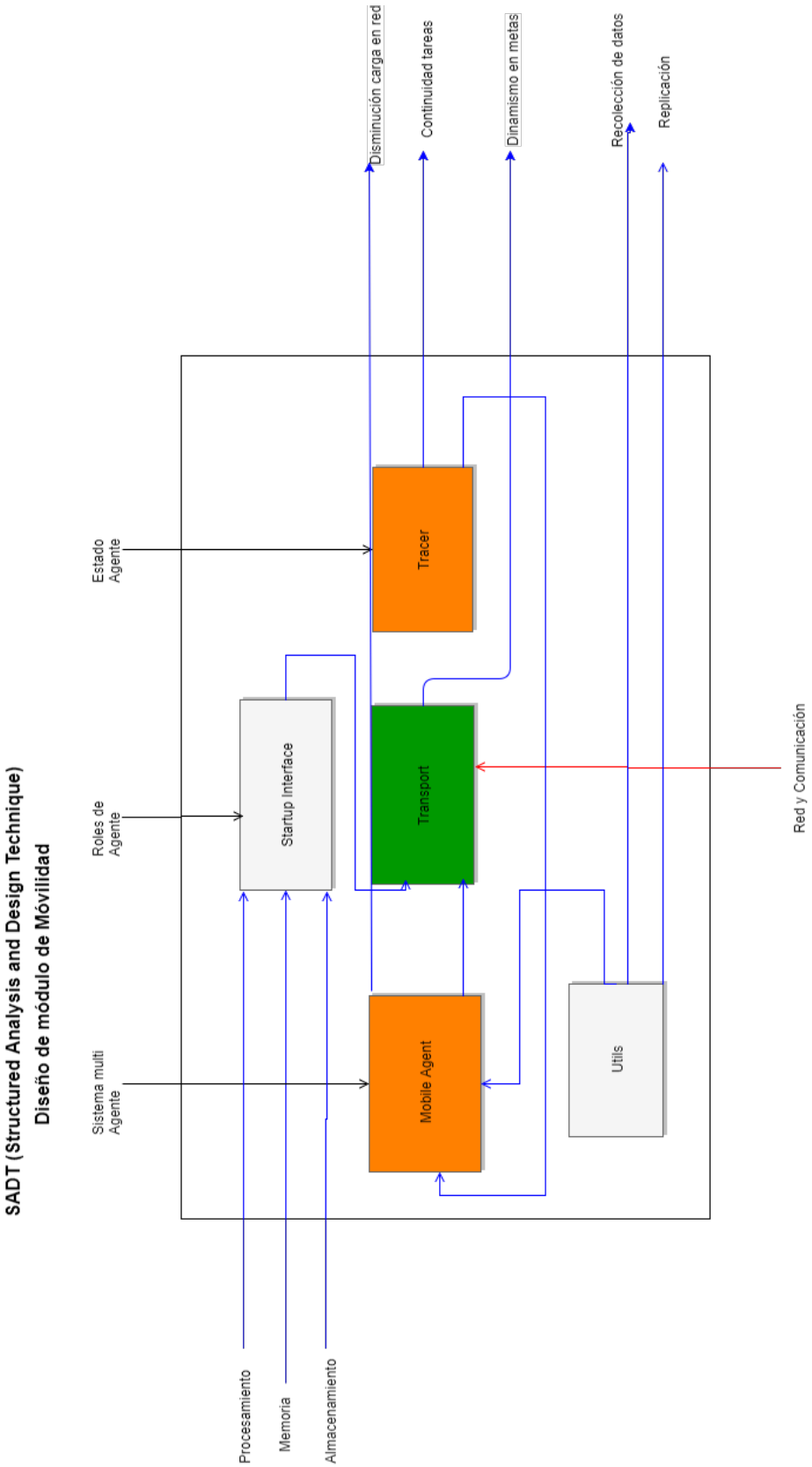

Figura 3-1: Diseño de sub-sistema de Movilidad - Movilidad [22]

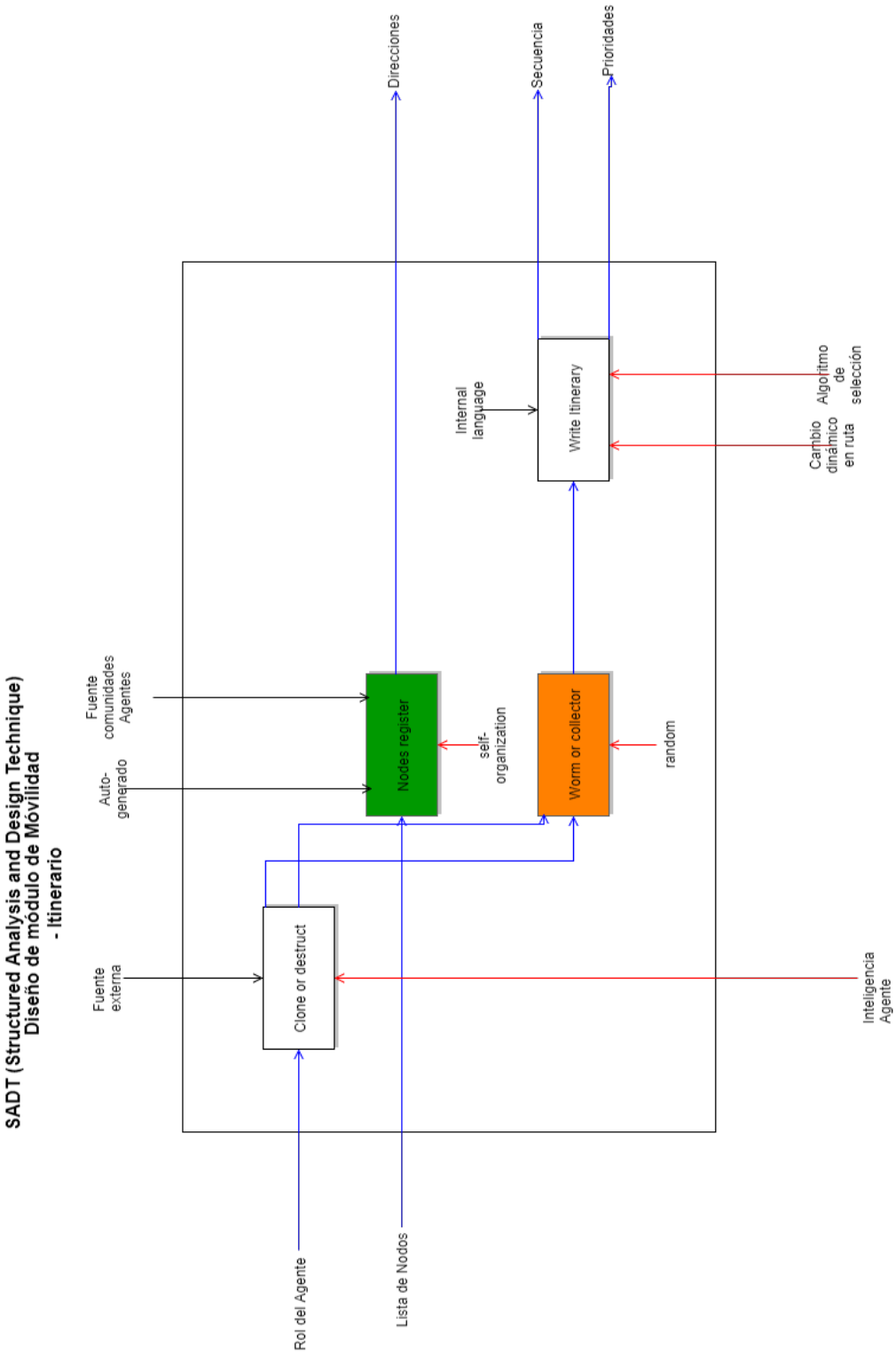

Figura 3-2: Diseño de sub-sistema de Movilidad - Itineraio [22]

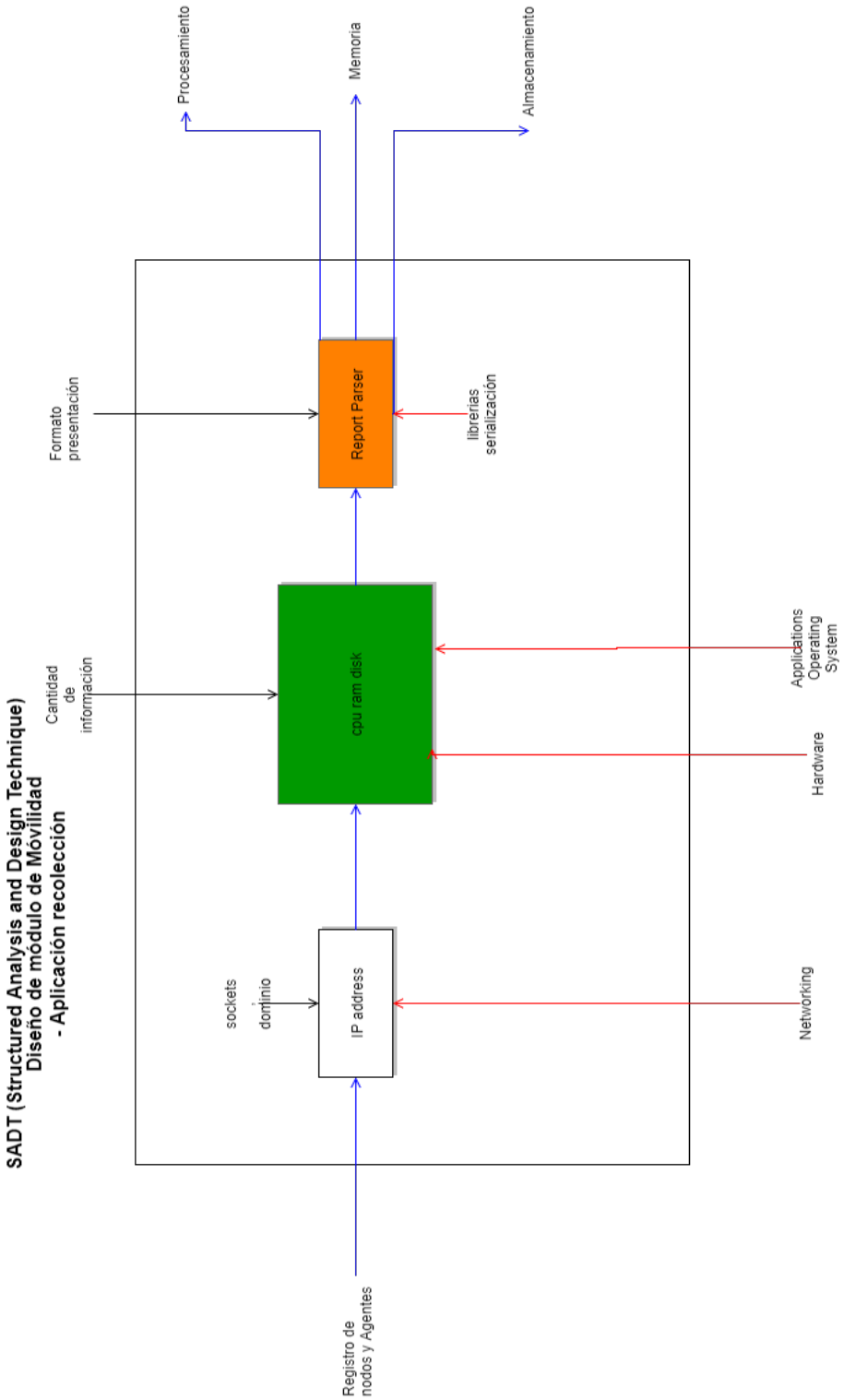

Figura 3-3: Diseño de sub-sistema de Movilidad - Recursos [22]

Previo a la movilidad es importante que el Agente logre su preparación la cual inicia con una

suspensión de sus actividades. Dentro de la preparación el guardar su estado de ejecución es el segundo paso a seguir.

En la figura **3-4**, es posible ver los eventos y acciones a los cuales responde el proceso de Movilidad, en sección 3.1 será explicado cada uno de estos estados y su finalidad en el proceso.

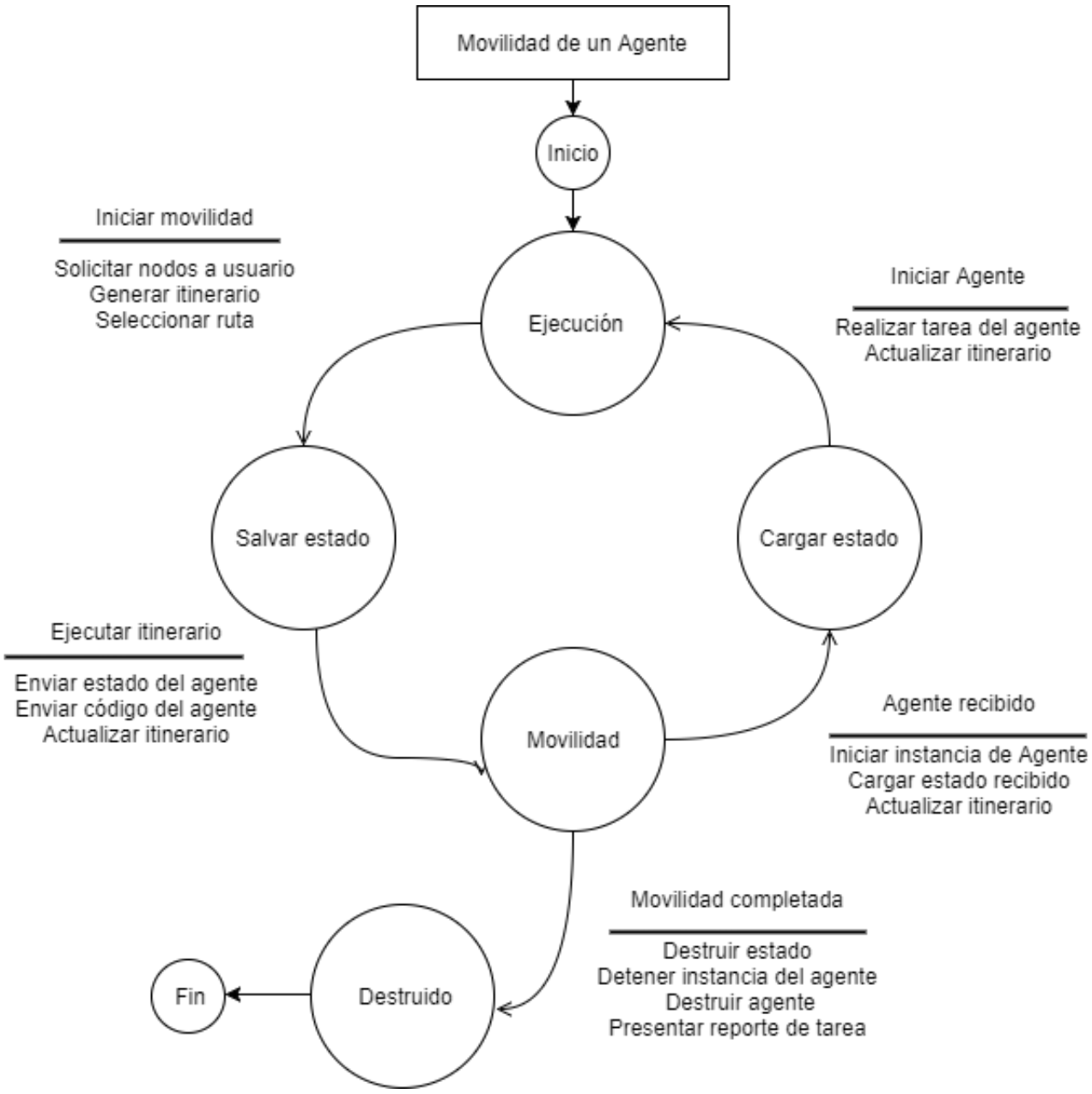

Figura 3-4: Máquina de estados finitos para el Agente de prueba implementado [22]

3.1. Cuerpo del Agente

El código del Agente es el cuerpo del mismo, la sentencia de lineas que permiten que este pueda ejecutarse en cada uno de los nodos por los que el mismo viaja, este código es leído y ejecutado por el Interprete de Python al momento de llamar el mismo. Dicho esto si el código estuviese corrupto o dañado, el Interprete no podría ejecutar el mismo y el Agente no podría cargar su estado o iniciar su tarea.

3.1.1. Estado del Agente

El poder preservar y restaurar el estado de ejecución para un Agente antes de ser migrado es aquello que permitirá al mismo dar continuidad a sus tareas. Es así que si esto no fuese posible el Agente deberá iniciar desde el principio y perderá el progreso que ha sido alcanzado durante su ejecución en el nodo actual.

Por esta razón el paso inicial para la movilidad parte en una serializción a través de la herramienta del lenguaje Python: **Pickle**

El módulo pickle implementa un fundamental pero poderoso algoritmo para serizalición y de-serialización de una estructura de objeto en Python. Pickle es el proceso donde un objeto de Python es convertido en flujo de bytes y un-pickle la operación inversa [41].
Entre los objetos que podemos convertir se encuentra:

1. listas
2. tuplas
3. funciones
4. clases

Convertir datos a string o bytes para almacenamiento o trasmisión sobre la red es una operación común en la computación. El nombre genérico que este recibe es serialización, Pickle es una forma de representación en Python sin embargo no es la única; Por ejemplo el JavaScript Object Notation (JSON) es otra de estas usada a través de gran cantidad de lenguajes especialmente en la web [6][15].

Pickle por si mismo no es una solución de administración o persistencia de datos, solamente convierte los objetos en secuencias de bytes [36]. Una de las principales ventajas y por la que se ha decido incluir esta herramienta en el desarrollo del proyecto es precisamente el paradigma de el estado de un Agente. Durante el contexto del proyecto: el estado hace parte del Agente y no puede descartarse en la movilidad, todo lo contrario debe ser una de las

piezas en consideración. Pickle nos permite resolver este dilema, salvar el estado de ejecución y restaurarlo en otro lugar y tiempo siempre y claro podamos acceder al archivo que ha sido generado o al flujo de bytes enviado a través de la red.

Para fines de desarrollo en el proceso de migración este estado almacenado es creado en un archivo temporal, que representa la información del Agente, sus variables,estructura y lo que caracteriza a este agente como único. Una vez este archivo sea enviado a través de la red a la ubicación destino del Agente será eliminado.

Envío de estado del Agente

El estado del Agente representa su memoria, representa la información que este ha obtenido en el transcurso de su creación hasta su destrucción. Por esta razón es importante que en el proceso de migración, o dicho de otra forma cuando el Agente se mueva a través de la red sea garantizado la integridad de esta información, de lo contrario el Agente perdería el trabajo realizado y debería empezar su tarea nuevamente para alcanzar el objetivo por el fue creado.

El protocolo para envío del estado ha sido designado como TCP, por sus características y orientado a conexión nos brinda una mayor fiabilidad sobre la información que se está enviando.

Serialización

Igualmente el guardar y restaurar son pasos que están divididos por el envío de la información, una vez completado el paso de guardar estado, se da cabida a el envío de este estado al destino. En efecto el envío debe ser realizado a través de la red, por lo que se da uso de protocolos orientados a conexión para garantizar la correcta migración de la información.

Restaurar estado del Agente

Siempre que los pasos anteriores hayan sido exitosos, el cargar el estado y la restauración de la ejecución del agente deberán existir para el éxito del proceso, el ambiente es de vital importante en la recepción del Agente, es el encargado de ofrecer la plataforma computacional y de servicios necesarios para que los agentes que lleguen, puedan continuar la ejecución de sus tareas y así el cumplimiento de sus metas.

3.1.2. Finalización del Agente

Un Agente debe tener una finalidad, una meta por cumplir, un propósito por el cual fue creado. En efecto, es muy común que este propósito tenga un inicio y un fin, por esta razón el Agente debe ser capaz de entender cuando su propósito ha sido culminado, entregar la

información que ha obtenido y ser destruido para mantener el ciclo.

Esta percepción de su ubicación temporal puede darse en sus acciones o en tiempo real como tal. Con el fin de mantener la sincronía y el mismo punto de referencia para todos los Agente se usa un tiempo a partir de los nodos como se describe en la siguientes sección. Sin embargo este servirá para mantener actualizados los datos y las comunicaciones pero como se ha mencionado no es la única forma en la que un Agente sepa en que estado de ejecución se encuentra, esto puede darse por acciones completadas [23].

3.2. Sincronización del Sistema

El tiempo, al igual que en nuestra vida cotidiana, es de gran importancia en los sistemas computacionales es aún mayor, y el MAS no es una excepción a este, todos los componentes deben estar sincronizados con el mismo tiempo, ya que en base a este se determina el estado del sistema como tal, si la información que se tiene es aceptable en un intervalo determinado o si por otra parte es necesario desplegar agentes que brinden información actual.

El sistema operativo que soporta el sub-sistema de movilidad se recomienda sea sincronizado a través del protocolo NTP (Network Time Protocol), el cual permite que cada nodo participante dentro del sistema distribuido tenga en mismo tiempo de referencia. Sin embargo dentro del alcance sólo se tendrá en cuenta el tiempo local que cada nodo posee. Este tiempo será usado para determinar la vida de los Agentes y otros factores de medida.

El tiempo de referencia utilizado es *POSIX* time, UNIX time el cual es definido como el numero de segundos que han pasado desde 00:00:00 Coordinated Universal Time (UTC), Jueves, 1 Enero 1970 [34]. El tiempo será usado al momento de la creación de un agente, en los mensajes de supervivencia enviados por el mismo a su ambiente y para cada información que este obtenga de la ejecución de sus tareas.

3.3. Roles de la movilidad

En principio la movilidad puede verse como un proceso básico de cambio de espacio-tiempo, sin embargo existe un gran número de posibles incógnitas cuando la palabra "moverse" entra en juego.

En Python, las funciones proveen una forma organizada de código que puede ser re-usado. En esencia esto permite usar las funciones predefinidas en el sistema o definir nuestras propias funciones. Las funciones permiten recibir variables, procesarlas y retornar el resultado del

procesamiento o acción que se ha requerido.

Las funciones del sub-sistema como lo son la función de itinerario y rol no van muy lejos de la definición de las funciones del lenguaje de programación. Se ha generado una acción la cual obedece a una serie de variables y parámetros propios del Agente que determinarán las forma en que el mismo realiza sus acciones en lo referente a movilidad, cabe resaltar que un Agente puede tener una o múltiples acciones y propósitos y la movilidad acá planteada no es más que un medio, no es más que un vehículo que sirve al mismo para finalmente lograr su objetivo o meta inicial para el cual fue creado [45].

3.3.1. Roles del Agente

Dentro del sistema es necesario identificar los roles o papeles que poseen los Agentes al momento de su creación. La importancia de los mismos se basa en que son los roles quienes otorgan al Agente de un propósito, este se comportará y tendrá características inherentes a sí dependiendo del rol que desempeñe [9].

Para este trabajo han sido definidos dos roles iniciales, los cuales darán al Agente un comportamiento particular y su propia forma de usar la movilidad desarrollada. Los roles definidos son:

Collector

Un Agente *colector*, será encargado de viajar a través de su ambiente y los nodos que lo conforman con el objetivo de recopilar información de cada destino por el cual el mismo se desplace. Este deberá obtener en su punto final toda la información ejecutándose de forma temporal en cada paso dentro de su itinerario previamente planeado.

Worm

Un Agente *worm* es diferenciado esencialmente al usar un esquema de replica dentro del proceso de movilidad, es decir, el Agente se ejecutará en cada uno los destinos que se encuentran en su itinerario, pero vez de usar una forma temporal el mismo al terminar el proceso de movilidad dejará una copia persistente de su ejecución por cada nodo que ha recorrido.

3.4. Itinerario

El itinerario de un Agente son los puntos definidos por los cuales el mismo debe moverse, la planificación del itinerario deberá determinar el orden de los nodos los cuales serán visitados durante la migración del Agente. Este orden de recorrido tendrán un impacto significativo

sobre el desempeño en el consumo de energía de todo el sistema multi-agente [12]. Ha sido expuesto que el encontrar un itinerario óptimo es aún un área de estudio e investigación, puesto que el no poseer un apropiado algoritmo puede desencadenar en ciertos inconvenientes como los son [12]:

- Un retraso sobre la ejecución.
- Una carga sobre la red que no puede ser balanceada eficientemente.
- Un consumo mayor de energía sobre aquellos nodos que deberán alojar el agente un tiempo después que el mismo haya obtenido más información del sistema como parte de su estado.
- Inseguridad en los datos obtenidos por el agente al aumentar la probabilidad de pérdida directamente proporcional al tamaño del agente, dado que este incrementa continuamente mientras visita los nodos.

Dicho esto, la planificación de itinerario en el sistema multi-agente es un factor a considerar en el desempeño del mismos y dependiente de las características y escalabilidad que se desea sobre el sistema y la red. Las técnicas para planificación del itinerario pueden ser divididas en tres diferentes tipos, dependiendo de los factores que sean necesarios y son:

3.4.1. Itinerario estático:

En un enfoque estático para la planificación del itinerario, la información global de toda la red es usada para identificar y ejecutar de forma eficiente una ruta para el Agente al momento de su creación, es decir, antes que el Agente sea desplegado en la red este ya conoce la ruta por la cual deberá realizar la movilidad [2].

Los actores que intervienen en el modelo asimilado por el proyecto son identificados como el Agente y el Ambiente y el usuario final. Por esta razón, el Agente sólo será un receptor y ejecutor de la ruta que le sea entregada, esto permite flexibilidad y objetividad pues es el ambiente ó usuario quien al poseer mayor información sobre todo el sistema inter-conectado determinará la ruta para el Agente.

Un posible inconveniente para este tipo de enfoque es que el Agente deberá llevar como parte de su estado el itinerario completo, y si este llegase a ser muy extenso la longitud y del Agente y el consumo de energía se verá incrementado. Sin embargo, la eficiencia computacional, la facilidad de implementación, configuración para la autonomía que brinda al Agente son factores determinantes al momento de elegir un itinerario este tipo en redes que no sean a gran escala [2].

Movilidad con itinerario estático

En estudios previos la planeación de itinerario ha sido una solución basada en un agente simple, esto significa que el consumo de energía, el tamaño del agente y desde luego la respuesta en tiempo real sobre el sistema son factores despreciables dado su tamaño y función principal. Sin embargo, para sistemas más complejos que requieran un dinamismo mayor, se recomienda implementar un algoritmo avanzado de selección de rutas.

Algunos ejemplos de este tipo de algoritmos son :

1. **Local closest First (LCF)**: Los agentes buscan el siguiente destino al nodo más cercano basado en distancia desde su ubicación local.

2. **Global closest First (GCF)**: Los agentes buscan el siguiente nodo como la distancia más corta no desde su ubicación local sino el más a la instancia del ambiente en que este se encuentra. Estos dos algoritmos son asociados a una baja complejidad y requerimiento computacional. Sin embargo al momento de escalar a miles o millones de nodos puede que el algoritmo deba ser reemplazado por uno que se adapte al crecimiento del sistema de mejor manera. Debido a que el tamaño del Agente crece a medida que cada nodos es visitado a gran escala puede que el mismo agente consuma todo el canal del sistema en cuyo caso un algoritmo que distribuya la información producto de la tarea del Agente o que envíe la misma a una entidad de convergencia es recomendado [2].

3.4.2. Itinerario dinámico:

En este tipo de itinerario la ruta que deberá seguir el agente es determinada en tiempo real, en el aire. Lo que requiere inteligencia y poder de decisión por parte del Agente en sí [12][38].

3.4.3. Itinerario Híbrido:

En el itinerario híbrido la ruta a seguir es determinada al momento de creación del Agente ya sea por su ambiente o por el usuario final, sin embargo el orden en el cual este la sigue es determinado por el Agente en tiempo real.

Aunque el rol puede ser cambiado como atributo de un Agente durante su ejecución, es el itinerario el cual cambia cada vez que el mismo se desplazada de un lugar a otro. Para lograr resolver tanto la necesidad de leer un itinerario inicial como el de ir actualizado el mismo se ha definido el mismo dentro de un archivo, el cual el Agente escribirá y leerá cuantas veces el mismo lo requiera.

Una increíble cantidad de información esta disponible en archivos de texto. Información desde trafico, socio-económica, literatura y configuraciones, sin embargo estos archivos también pueden ser usados para representar información o simplemente guardar los resultados de procesos ejecutados. Existen una variedad de partes del sub-sistema que usa los archivos tanto como entrada como salidas, un caso particular de salida es el Módulo Resources el cual deberá captura información del sistema y escribir la misma en un archivo en formato JSON una vez haya sido completada su tarea. A continuación se muestra la forma en la que podemos leer información de un archivo como la lista de nodos los cuales formarán la ruta que deberá seguir la movilidad dentro del sistema en un caso de uso [40].

Mediante el uso del contexto (with) y el archivo **tracer** en el cual el estado del Agente es guardado a través del lenguaje de programación Python es posible leer su contenido como se muestra a continuación:

```
with open('tracer') as file_object:
    contents = file_object.read()
    print(contents)
```

De forma similar es posible actualizar o crear un nuevo archivo, durante el proceso se ejecución el Agente leerá su itinerario y en la movilidad llevará consigo el mismo para una vez termine la misma el Agente dará como visto el nodo que acaba de dejar hasta terminar con toda la lista del itinerario inicial.

3.5. Recursos: la meta del Agente Móvil

Una recopolidación de datos forma eficiente, cuando existe una gran cantidad de los mismos, ha sido un puerta de investigación y desarrollo en sistemas basados en el uso de Agente Móviles, los cuales son procesos capaces de migrar de un nodo a otro con el objetivo de satisfacer las necesidades del usuario [49].

La forma en que se obtiene información sobre los recursos de almacenamiento, el procesamiento y las configuraciones lógicas en una computadora depende del sistema operativo que esté ejecutando. Por esta razón en el desarrollo de este documento se mostrará la construcción de un módulo de software que permite la extracción de esta información y que tiene el potencial de trabajar en varios sistemas operativos, permitiendo así el acceso a través de una capa de programación creada bajo lenguaje de programación Python [11].

Finalmente, el módulo de software se ejecutará bajo una distribución de Linux que permitirá verificar el funcionamiento del mismo y generar un informe sobre los recursos e información del sistema actual.

3.5.1. Diseño del módulo de recursos

Los módulos son fundamentales para la mayoría de los ambientes de programación donde esta no te torna de una forma trivial, estos permiten a los programas dividirse en pequeñas partes y de esta forma a su vez permitir la reutilización de código a través de diferentes proyectos. En Python, los módulos son simples archivos cuya terminación es .py y localizados en algún lugar del sistema donde el lenguaje pueda ubicar. Dentro del proyecto esto no es diferente, el mismo sub-sistema ha sido divido en módulos los cuales en su interior desempeñan una función principal; tal como el caso del modulo Transport.py el cual administra y define todo lo referente a la comunicación y transporte del modulo Agents.py que define las propiedades inherentes al Agente.

En general podría decirse que el sub-sistema de movilidad es un paquete compuesto de módulos internos que se complementan entre sí brindando la habilidad de moverse y cumplir sus metas a los Agentes que han sido creados para beneficiarse el mismo [36].

Para el diseño se propone generar una clase denominada *SystemResources* y dentro de las mismas funciones indicadas utilizando únicamente módulos nativos de Python, de modo que eliminemos posibles dependencias en el momento de ejecución del módulo en otro entorno. El código de Resources.py puede finalmente ser utilizado y migrado a diferentes distribuciones e importado por aplicaciones externas para llamar a sus funciones. La versión de Python3 se utilizará para crear el módulo.

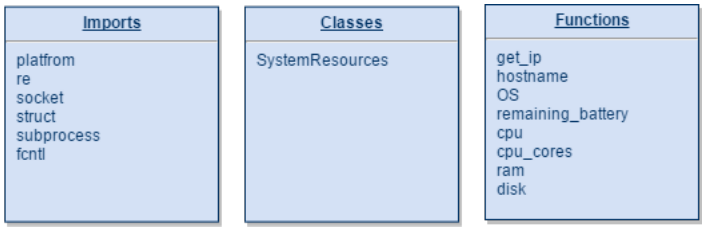

Figura 3-5: Módulos, clases y funciones para obtener información sobre recursos del sistema [22]

La figura **3-5** muestra un diagrama de los módulos, clases y funciones que se encuentran en Resources.py. Algunos de éstos serán explicados e implementados en los siguientes párrafos, sin embargo la mayoría serán probados en trabajos futuros.

3.5.2. Recursos del sistema y configuración de red

Sistema Operativo

Python ofrece la posibilidad de identificar el sistema operativo en el que se está ejecutando el intérprete, mediante el módulo **platform** que incluye información adicional sobre el sistema, entre otros:

- architecture
- collections
- dist
- machine
- processor
- python version
- system
- uname
- version

La función generada para devolver el tipo de sistema operativo en el que se ejecuta el módulo es muy útil ya que cada sistema proporciona características específicas que pueden generar

problemas en el momento de la ejecución de cualquier aplicación desarrollada. La siguiente es la definición de la función OS:

```
@staticmethod
def OS():
    """Return system platform Linux, Windows, MacOS"""
    return platform.system()
```

Todo el sub-sistema cumple con PEP8 de Python y docstrings estarán presentes para explicar el uso de cada función.

RAM

Las utilidades en las diferentes distribuciones de Linux como gratuitas nos permiten determinar la información relevante de la RAM (Random Access Memory) del sistema, sin embargo en aquellas que la aplicación no está presente podemos determinarla por el archivo /proc/memeinfo, usando expresiones regulares a través de Python es posible leer y presentar de forma sencilla las características de la memoria como el valor total, utilizado y libre.

IP Address

En el día de hoy donde la conexión y las redes han adquirido gran importancia, cada sistema, PC, smarthphone, debe ser identificado en la red a través de una dirección IP (Internet Protocol), por lo que saber en qué red somos y si tenemos acceso o salida a Internet son las principales acciones a identificar en un dispositivo con interfaces de red.

La función definida a continuación mira el sistema operativo diferente y devuelve la dirección IP del host que ejecuta la acción basada en los módulos (socket, struct) de Python.

```
@staticmethod
def get_ip(ifname='eth0'):
        """Get the IP number of the main interface"""

        ip = socket.gethostbyname(socket.gethostname())
    if ip.startswith('127') and platform.system() == 'Linux':
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        ip = socket.inet_ntoa(fcntl.ioctl(
            s.fileno(),
            0x8915,
            struct.pack(b'256s', ifname[:15].encode('utf-8'))
        )[20:24])
    elif ip == None: ip = "Could not get IP Address"
    return ip
```

Interfaz de configuración

Para lograr ver la movilidad del Agente de prueba, se usa la interfaz de parametrización, la cual permite a el usuario determinar el modo de funcionamiento del sistema y el itinerario si de desea generar de modo interactivo y manual. En la siguiente imagen se observa el acceso a la interfaz donde sera necesario la configuración de los nodos por lo cual el agente será desplegado.

```
[german@sp mobile-agent]$ python startup.py
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>Welcome to mobility setup module>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>


please enter the number of nodes :2
please enter the IP address for node 0 :"65.200.195.205"
please enter the IP address for node 1 :"65.200.195.206"
loading information into sub-system...
sending mobile agent across network...
```

Internamente la interfaz se encarga de desplegar automáticamente componentes como lo es por ejemplo el ambiente, y permite que el usuario sólo se concentre en lo que es de su interés. Desde luego en caso de requerir el ingreso de una gran cantidad de nodos, o que los mismos provengan un software externo se ha dejado la opción de leer los mismos desde un archivo o a través de la red. Sin embargo puesto que el objetivo no es comprobar las posibles opciones del itinerario como una entrada para el proceso de movilidad, se utilizará el ingreso de nodos por modo manual usando la interfaz mencionada en 3.5.2.

Software y requerimientos

Dentro de los requerimientos necesarios de software ha sido posible reducir los mismos a los módulos y librerías nativas de Python, es decir, el sub-sistema ha sido diseñado usando los módulos nativos, esto es de gran utilidad ya que puede ser ejecutado sobre cualquier sistema que posea el lenguaje Python en su versión 3 previamente instalado. A continuación se denota una lista de los requisitos de entorno para poder dar uso de todas las funcionalidades del sub-sistema de movilidad acá descrito:

1. Python3
2. Linux OS
3. Paquete mobile-agent

CAPÍTULO 4

Escenarios de prueba

4.1. Movilidad del Agente sobre una red Ad Hoc

Uno de los escenarios sobre los que se plantea el sub-sistema de movilidad es una red Ad Hoc creada a partir de dispositivos embebidos que para este caso particular será la Raspberry Pi 3 Model B, la cual en esta versión posee un adaptador inalámbrico integrado. Es así que con un conjunto de tres nodos conectados entre sí por una red Ad-Hoc permitirán evaluar la movilidad de Agentes sobre una implementación real.

Una aclaración se hace pertinente ya que uno de los objetivos es desplegar el sistema de movilidad sobre un sistema distribuido, donde bajo nuestra definición del mismo los dispositivos o nodos de Raspberry PI son los computadores y el problema a resolver es la comunicación y el permitir la movilidad del Agente sobre la red que estos han creado.

La computación distribuida sobre múltiples computadores es una estrategia cuando se usan sistemas que son capaces de hablarse entre sí a través de la red. La razón principal para construir una sistema distribuido es poder dividir un problema tan grande que un sólo computador no puede manejarlo todo por si sólo, pero al dividir el mismo múltiples computadores pueden comunicarse para resolver cada una de sus partes.

Uno de los ejemplos familiares son las películas animadas de Pixar o DreamWorks, procesar las animaciones de 3D, a 30 cuadros por segundos en una película de dos horas es una gran carga, por lo cual los estudios requieren dividir en granjas de computadoras para que computadores individuales procesen una sólo parte de la película logrando con ello el objetivo final en un tiempo razonable [15].

4.1.1. Organización de la red Ad-Hoc

En la siguiente figura se muestra el esquema y direccionamiento de red con el que son conectados los dispositivos en mención:

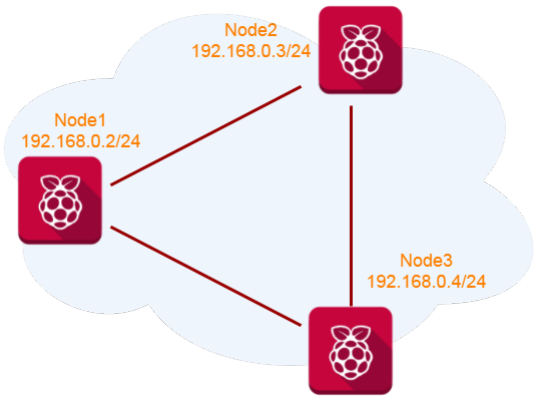

Figura 4-1: Red Ad-Hoc para movilidad nodo a nodo [22]

Para este escenario ha sido necesario la configuración de tres nodos con el fin de permitir una movilidad completa y un itinerario mayor que sólo una red con dos nodos. La razón principal de este escenario es observar el funcionamiento de la movilidad en un ambiente físico real, permitiendo la expansión del mimo a otros nodos heterogéneos que cumplan con los requisitos y desde luego a mayor cantidad de los mismos. Sin embargo, el despliegue en redes de mayor complejidad será planteada como trabajo futuro del aquí presente.

4.1.2. Sistema Operativo de los nodos

Para estos dispositivos ha sido instalado como sistema operativo Raspbian basado en Debian JESSIE con kernel 4.4, una de las distribuciones recomendadas por el fabricante del dispositivo Raspberry PI.

4.1.3. Instalación del sub-sistema de movilidad

Una vez creada la red es necesario añadir a cada uno de los nodos participantes el paquete mobile-agent el cual contiene los módulos necesarios para el funcionamiento tales como Ambiente, Agente, recursos entre otros. Como ha sido mencionado para que un Agente pueda existir debe estar sobre un ambiente es por esta razón que el primero módulo en ejecución deberá ser el de ambiente. El envio de este sub-sistema se realiza accediendo por SSH (secure

shell) directamente a cada uno de los dispositivos [53].

Los nodos designados han sido configurados para la prueba de la movilidad, con fines de verificación ha sido creado un Agente (MobileAgent.py) cuya meta será moverse por cada uno de los nodos recolectando información del sistema (CPU, RAM, IP, Almacenamiento) y finalmente generar un reporte con la misma.

4.2. Movilidad del Agente sobre un sistema virtualizado

4.2.1. Organización del sistema virtualizado

El sub-sistema de Movilidad y los módulos que lo componen han sido diseñados para trabajar tanto un ambiente físico como sobre uno virtual el cual será desarrollado en los siguientes párrafos.

Para la virtualización de los nodos ha sido configurado a través de Docker, el cual es un software de virtualización con contenedores que comparten el mismo kernel y que permiten configuran el ambiente y requisitos dentro de cada uno, para el caso aquí mencionado debe contener el lenguaje de programación Python.

Python3 por otra parte, constantemente esta obtiene nuevas y mejoras que nunca llegarán a Python2, por esta y muchas razones del lenguaje se ha decido realizar el proyecto sobre versiones de Python3 en general [55].

En la figura **4-2** se observa el esquema de conexión de los contenedores virtuales por donde el Agente usando su módulo de movilidad se desplazará:

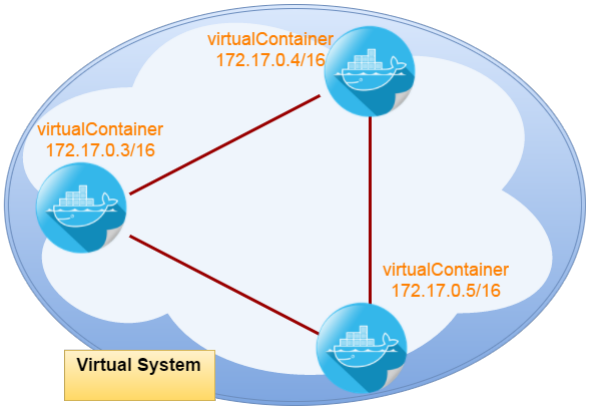

Figura 4-2: Movilidad sobre un sistema virtualizado [22] [42]

La figura **4-2** surge dentro del Modelo de capas del sistema TLON, recordando que este diseño del sistema ha inspirado el diseño y desarrollo del sub-sistema de movilidad contendido en este trabajo.

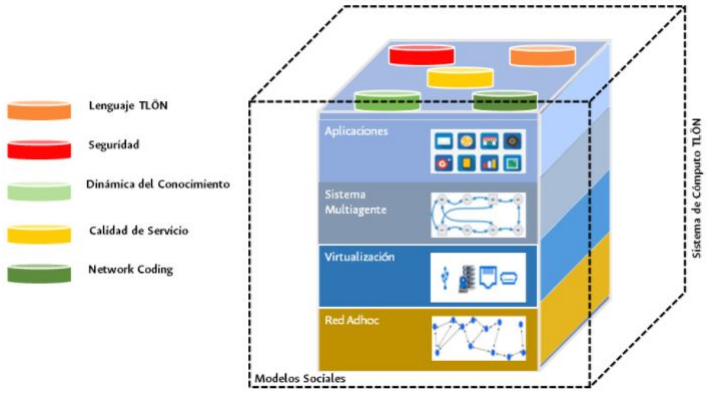

Figura 4-3: Modelo de capas TLON [31]

La diferencia principal es que para este sistema sólo necesitamos un nodo físico, es así que este virtualizará los contenedores creando desde la perspectiva del Agente como si se moviera a través de sistemas y nodos físicos, cuando realmente el mismo esta desplazándose sobre un sistema virtualizado gracias a Docker. Esto nos permite intuir que la movilidad de un Agente no necesariamente siempre se ejecuta sobre dispositivos físicos, también es posible sobre dispositivos virtuales sin cambiar en sí lo que la movilidad representa. Este tipo de virtualización difiere a la conocida como máquinas virtuales al compartir el kernel entre los contenedores [42][35].

En la ejecución del mismo la movilidad funciona sin complicación alguna, sólo que esta vez ha sido reemplazado el itinerario el cual es:

4.2.2. Itineario

1. 172.17.0.3
2. 172.17.0.4
3. 172.17.0.5

4.2.3. Contenedores

Un contenedor es una forma ligera, ejecutable de una pieza de software que posee librerías, herramientas del sistema, configuraciones y todo lo que necesite para ser ejecutado. Los contenedores son una forma de aislamiento para el software que corre dentro de ellos de su

alrededor, es la forma de virtualización que ofrece docker para sus usuarios y servicios [57][42].

Para este escenario han sido creados tres contenedores, los cuales poseen la misma imagen base de tal forma que poseen similitudes entre ellos, sin embargo las direcciones de red, identificadores y nombres los difieren.

Así como en la sección 3.3 se ha mencionado los roles definidos y el comportamiento de movilidad que cada Agente adquiere acorde este rol. En el reporte extraído del sistema virtualizado se logra identifica como el orden de los datos no corresponden estrictamente a el orden de itinerario dado inicialmente, esto se debe a que el rol que ha sido asignado al Agente ha sido *collector*, lo que le da habilidad de recorrer aleatoria mente su itinerario.

Una vez configurado nuestro sistema virtualizado por medio de Docker, El primer paso deberá ser iniciar el ambiente en cada contenedor, planear su itinerario y desplegar el Agente para que el mismo pueda cumplir su objetivo. A continuación se observa como cada nodo corre el ambiente y sólo el MobileAgent.py está presente en el nodo número 172.17.0.3 (El primer salto del itinerario), el itinerario del Agente para este caso será estático y recorrerá los siguientes nodos en orden aleatorio.

4.2.4. Creación de un contenedor

El Dockerfile es el archivo que define las características de la imagen que se desea construir, imagen base para la creación de cada uno de los contenedores del sistema virtual. Para este caso a sido usada como base una imagen con sistema operativo Ubuntu con características sencillas que incluyen dentro de sí el sub-sistema de movilidad generado en el desarrollo de este trabajo.

```
FROM ubuntu

# Update Software repository
RUN apt-get update
RUN apt-get install -y python3
RUN apt-get install -y net-tools
ADD mobile-agent /
```

El Agente será creado en Node1 y terminará en Node3 con la creación del reporte de recursos que el mismo debe generar. Una vez ejecutado el Agente el reporte en formato JSON generado es el que se muestra a continuación:

```
{
    "node172.17.0.5": {
```

```
        "OS": "Linux",
        "host": "4074d48bb7c6",
        "disk": {
            "%use": "9%",
            "total": "1.8T",
            "free": "1.6T",
            "used": "150G"
        },
        "ram": {
            "total": 31783.93359375,
            "free": 3645.625
        },
        "ip": "172.17.0.5"
    },
    "node172.17.0.3": {
        "OS": "Linux",
        "host": "4a9211aae28d",
        "disk": {
            "%use": "9%",
            "total": "1.8T",
            "free": "1.6T",
            "used": "150G"
        },
        "ram": {
            "total": 31783.93359375,
            "free": 3646.140625
        },
        "ip": "172.17.0.3"
    },
    "node172.17.0.4": {
        "OS": "Linux",
        "host": "377512b44e4c",
        "disk": {
            "%use": "9%",
            "total": "1.8T",
            "free": "1.6T",
            "used": "150G"
        },
        "ram": {
            "total": 31783.93359375,
```

```
            "free": 3645.26953125
        },
        "ip": "172.17.0.4"
    }
```

JSON: es un estándar de representación de objetos simples, como listas o diccionarios en forma texto, aunque originalmente fue desarrollado para JavaScript, por ello su nombre JavaScript Object Notation(JSON) esto es independiente de lenguaje, iviano , flexible y capaz de manejar gran antidad de datos. Es así que se hace ideal para intercambiar información sobre protocolos como HTTP y un gran número de web APIs las cuales usan este como su principal formato de datos [18].

CAPÍTULO 5

Resultados y análisis de resultados

Una vez el Agente ha llegado a su destino como nodo final de su itinerario el mismo ha generado el reporte de recursos por el cual fue creado, es decir ha cumplido su meta. Por ello el Agente detiene su ejecución y puede dar por terminada su tarea.

Es interesante como en este escenario se observa cada una de las características técnicas y conceptuales de la movilidad, el Agente que ha sido creado en Node1 puede desplazarse a través de la red sobre su ambiente por Node2 y Node3 respectivamente desarrollando una función determinada por cada nodo que este recorre.

De igual forma la meta que se ha propuesto para este Agente de prueba es pertinente en el caso que nos permite ver como el Agente posee memoria, como el mismo lleva consigo no sólo su código fuente sino además los datos y lo que ha aprendido durante su recorrido(estado). Esto es fundamental para que el Agente pueda tener memoria y además de ello controlar su itinerario en progreso. En las figuras **5-3 5-4** se observan datos de los tiempos de movilidad y ejecución de este Agente sin y con un incremento de payload (tamaño) del mismo. Esto se realiza con el fin de determinar si la movilidad puede realizarse para Agentes de diferentes tamaños y funciones.

Nodo	Nodo 1(ms)	Nodo 2(ms)	Nodo 3(ms)
Ronda			
1	13.25323712081	11.58814039	12.67810725
2	15.48183337	11.20261382	11.53095067
3	13.72570917	12.44049985	12.41615508

Figura 5-1: Tiempos de movilidad entre nodos [22]

Nodo 1(ms)	Nodo 2(ms)	Nodo 3(ms)	Payload (bytes)
17.27611385	14.0317129	13.37316167	1024
16.73824433	15.75741917	15.75074252	10240
30.07273283	31.38254676	32.52156358	102400
165.8888739	197.9421154	193.48008	1024000

Figura 5-2: Tiempos de movilidad entre nodos con tamaño incremental [22]

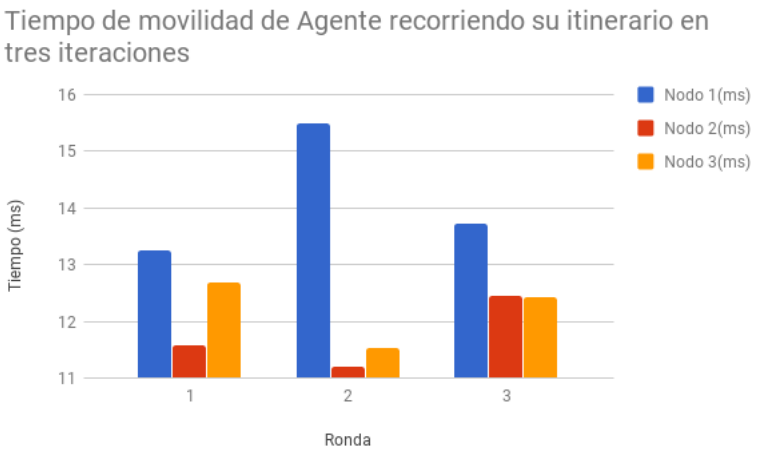

Figura 5-3: Gráfica sin carga [22]

Figura 5-4: Gráfica con carga [22]

En un sistema multi-agente: el tamaño, la meta e incluso la definición de cada Agente puede variar, por esto es importante que el sub-sistema sea capaz de adaptarse a estas necesidades y brinde a cada uno de los Agentes la capacidad de moverse.

CAPÍTULO 6

Conclusiones, recomendaciones y trabajo futuro

6.1. Conclusiones

Python puede definirse como un popular lenguaje de alto nivel, para propósito general, interpretado y dinámico. La filosofía de Python está enfocada en la claridad y fácil entendimiento. Además de ello la sintaxis del mismo permite expresar conceptos con gran calidad y fácil entendimiento en pocas líneas comparados con otros lenguajes populares como Java y C [58].

La creación del sub-sistema de movilidad incluyó una serie de módulos, funciones y decisiones que fueron tomadas siempre con lineamientos base del comportamiento de los Agentes y la posible influencia de su naturaleza, es decir, se ha inspirado las habilidades del mismo con la intención de cubrir sus necesidades de movilidad en un escenario real.

Sobre el trabajo desarrollado puede destacarse la modularidad, flexibilidad y arquitectura del proyecto, como cada una de los componentes juega un papel dentro del sub-sistema y como se integran entre sí, en el caso del Agente y su Ambiente se logra observar la comunicación y complemento de los mismos. Además de ello, la ventaja que ofrece un despliegue del sub-sistema como único requerimiento del lenguaje Python el cual esta presente por defecto en la mayoría de sistemas operativos que usan el kernel Linux o Unix-like.

Es importante observar el nivel de abstracción que ofrece la movilidad para el sistema Multi-Agente, como ha sido probado en los escenarios el sub-sistema es capaz de funcionar bajo un esquema de recursos físicos, un esquema de virtualizar contenedores como lo es Docker o incluso un sistema de virtualización completa como el caso de Oracle VirtualBox aunque este ultimo no fue incluido dentro del alcance del proyecto. Gracias al uso de librerías nativas, de protocolos de comunicación y siguiendo con las recomendaciones y buenas practicas ha si-

do generado un proyecto escalable, modular y que puede integrarse a sistemas más complejos.

Entre las funciones que permiten planear y ejecutar el itinerario y el modo de llevarlo a cabo fue tomado en consideración el rol del Agente dentro del sistema como tal, lo cual sin lugar a duda determinó su comportamiento en lo que a movilidad se refiere, estas funciones reciben características del Agente como lo es un rol, sin embargo cada vez que la función es llamada requiere este parámetro, esto representa una ventaja en el tiempo puesto que al ser el rol un atributo de Agente puede ser cambiado en un momento dado y el módulo movilidad que provee esta función entenderá que ahora que el Agente posee otro rol, seguramente requiere un esquema de movilidad o itinerario sea acorde a su nueva definición.

El objetivo o meta es una de las primordiales características de un Agente, para el desarrollo de los escenarios la meta del Agente utilizado ha sido de gran ayuda al demostrar un caso de uso real. Obtener los recursos del sistema se logró sin necesidad de instalar en cada nodo o contenedor un Agente estático que envíe la información a un punto de concentración, fué posible el mismo resultado a través de un solo Agente, y sobre un sistema distribuido el cual era una de los objetivos del proyecto, dentro del trabajo futuro sera incluida una comparación de el alcance de esta meta con Agentes sin movilidad comparados con Agentes Móviles.

Finalmente en los escenarios de prueba tanto el que incluye el despliegue sobre un esquema físico como el que lo hace sobre una red y esquema virtual ha sido posible usar y observar la interacción y funcionalidad de cada uno de los componentes y módulos del sistema. Aunque el escenario cambio de un ambiente físico a un ambiente virtual el sub-sistema no requirió cambio alguno, gracias a que el diseño del mismo no lo limita a un sistema, fabricante o dispositivo físico.

6.2. Recomendaciones

Este proyecto presenta oportunidades de mejora continua, tanto en los módulos existentes como en aquellos que no hacen parte de los objetivos pero que si pueden llegar a impactar directamente la movilidad. De esta forma se plantean las siguientes recomendaciones por parte del autor:

Extender la funcionalidad del módulo de recursos: aunque el módulo actual obtiene gran parte de la información del sistema este puede ser extendido por medio de proyectos escritos en Python como lo son *psutil*. Esto con el fin de garantizar que nuevos usos y captura de información estén disponibles para los Agentes y para la movilidad.

Complementar el itinerario: una de las características determinantes de la movilidad fue el itinerario, la forma en la que se obtiene el mismo actualmente puede ser desde un archivo

plano o bajo una interfaz de usuario. Lo que se recomienda es extender el mismo para que se pueda generar a través de un servicio de red. Esto quiere decir que el Agente puede podrá solicitar al ambiente de forma automática su itinerario o el ambiente enviárselo al mismo Agente. Esto garantiza mayor autonomía y flexibilidad, ya que el sistema esta diseñado a trabajar en red la comunicación que se realice entre la red aportara a completar sus metas.

Extraer la documentación del módulo de software: todo el software fue realizado con los principios y estructuras de PEP8, los Docstrings y comentarios están incluidos en cada uno de los módulos. Sin embargo se recomienda para una mayor compresión de todo el sistema extraer los mismos a una visualización web. Proyectos como *sphinx* permiten realizar esto de forma automática y rápida al usuario.

Con base a las pruebas sobre el ambiente virtual creado sobre Docker, se pudo observar que la capa de seguridad del motor interfiere con la movilidad de los Agentes. Para dar solución a eso los contenedores fueron creados con la opción *–privileged*. Se sugiere que a futuros trabajos los permisos sean limitados acorde a los casos de uso que se dará a la movilidad y a los Agentes desplegados dentro del sistema de contenedores.

Integrar lenguaje ACL (Agent Communication Language): el ACL propuesto por la FIPA (Foundation for Intelligent Physical Agents), es un lenguaje de comunicación entre agentes a través de XML. Integrar un sistema de comunicación permite abrir el sistema a integraciones con otros sistemas que hayan sido creados en diferentes lenguajes y sistemas operativos. Es una gran oportunidad para interoperar con el mundo y dar visibilidad al software como tal.

6.3. Trabajo Futuro

Como trabajo futuro se plantea el despliegue del sub-sistema de movilidad y la incursión de nuevos Agentes Móviles sobre una red con mayor tamaño, con un incremento en los nodos participantes. De esta forma puede verse la movilidad para diferentes funciones fuera de la recolección de recursos presentada en este trabajo, también puede verse como habilidad para configuración, control y mantenimiento.

Una de las ventajas que han sido mencionadas sobre la Movilidad de agentes ha sido precisamente el ahorro y desempeño sobre redes limitadas que estos ofrecen referente a anchos de banda y reducción de tráfico. Podría plantearse un escenario en el que se cuantifique el desempeño de la movilidad para una tarea en un modelo de Agentes Móviles cooperativos contra un modelo de Agentes sin movilidad.

Para dar por terminado se sugiere añadir al sub-sistema de movilidad la función de mover

o transferir no sólo aquello que hace parte del Agente , sino además objetos externos al mismo, es decir, que un Agente podría pensar en viajar y llevar junto a este un Agente estático que se encuentre a su alcance sin modificar o hacer cambios en las características de este acompañante.

Bibliografía

[1] Abdel-Halim, Islam T. ; Fahmy, Hossam Mahmoud A. ; Bahaa-Eldin, Ayman M.: Agent-based trusted on-demand routing protocol for mobile ad-hoc networks. En: *Wireless Networks* (2015), Nr. 2, p. 467. – ISSN 1022–0038

[2] Alsboui, Tariq ; Alrifaee, Mustafa ; Etaywi, Rami ; Jawad, Mohammad A.: Mobile Agent Itinerary Planning Approaches in Wireless Sensor Networks- State of the Art and Current Challenges. Springer, Cham, oct 2017, p. 143–153

[3] Ametller, Joan ; Robles, Sergi ; Borrell, Joan: Agent Migration over FIPA ACL Messages. En: *Mobile Agents for Telecommunication Applications* 2881 (2002), Nr. PRO-VE 2, p. 379–386. – ISBN 978–3–540–20298–1

[4] Bahrami, Saeed ; Torgheh, Fatemeh: Genetic algorithms for finding optimal locations of mobile agents in scalable active networks. En: *2011 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC)*, IEEE, aug 2011. – ISBN 978–1–4577–0535–9, p. 7448–7453

[5] Bardhi, B. ; Claudi, A. ; Spalazzi, L. ; Taccari, G. ; Taccari, L.: Chapter 6 – Virtualization on embedded boards as enabling technology for the Cloud of Things. En: *Internet of Things*. 2016. – ISBN 9780128053959, p. 103–124

[6] Bassett, Lindsay: *Introduction to JavaScript object notation : a to-the-point guide to JSON*. Sebastopol, CA : O'Reilly Media, 2015. – ISBN 978–1491929483

[7] Bosse, Stefan: Intelligent microchip networks: an agent-on-chip synthesis framework for the design of smart and robust sensor networks. En: Riesgo, Teresa (Ed.) ; Conti, Massimo (Ed.): *Proceedings of the SPIE 2013, Microtechnologie Conference, Session EMT 102 VLSI Circuits and Systems, 24-26 April 2013, Alpexpo/Grenoble, France* Vol. 8764, 2013, p. 87640R

[8] Bosse, Stefan: Agent-based Solutions for Industrial Environments composed of Autonomous Mobile Agents, Modular Agent Platforms, and Tuple Spaces. En: *Proceedings of 2nd International Electronic Conference on Sensors and Applications*. Basel, Switzerland : MDPI, nov 2015, p. S5001

[9] Cabri, Giacomo ; Ferrari, Luca ; Leonardi, Letizia: A role-based mobile-agent approach to support e-democracy. En: *Applied Soft Computing* 6 (2005), nov, Nr. 1, p. 85–99. – ISSN 15684946

[10] Cao, Jiannong ; Das, Sajal K.: *MOBILE AGENTS IN NETWORKING AND DISTRIBUTED COMPUTING*. 2012

[11] Cassell, Laura ; Gauld, Alan: *Python Projects*. 1. Wrox, 12 2014. – ISBN 9781118908662

[12] Chen, Min ; Gonzalez, Sergio ; Zhang, Yan ; Leung, Victor C. M.: Multi-Agent Itinerary Planning for Wireless Sensor Networks. 2009, p. 584–597

[13] Chen, Min ; Yang, Laurence T. ; Kwon, Taekyoung ; Zhou, Liang ; Jo, Minho: Itinerary planning for energy-efficient agent communications in wireless sensor networks. En: *IEEE Transactions on Vehicular Technology* (2011). – ISBN 0018–9545

[14] Cicirello, V.A. ; Mroczkowski, A. ; Regli, W.: Designing decentralized software for a wireless network environment: evaluating patterns of mobility for a mobile agent swarm. En: *IEEE 2nd Symposium on Multi-Agent Security and Survivability, 2005.*, IEEE, 2005. – ISBN 0–7803–9447–X, p. 49–57

[15] Dalcin, Lisandro D. ; Paz, Rodrigo R. ; Kler, Pablo A. ; Cosimo, Alejandro: Parallel distributed computing using Python. En: *Advances in Water Resources* (2011). – ISBN 0309–1708

[16] Dean, Justin W. ; Macker, Joseph P. ; Chao, William: A study of Multiagent System operation within dynamic ad hoc networks. En: *MILCOM 2008 - 2008 IEEE Military Communications Conference*, IEEE, nov 2008. – ISBN 978–1–4244–2676–8, p. 1–7

[17] Dhanalakshmi, K ; Kadhar Nawaz, G M.: Fault-Tolerant Scheme for Matrix Hop Mobile Agent (MHMA) System. En: *Arabian Journal for Science & Engineering (Springer Science & Business Media B.V.)* 38 (2013), Nr. 12, p. 3339–3347. – ISSN 13198025

[18] Dr. M. O. Faruque Sarker, Sam W.: *Learning Python Network Programming*. Packt Publishing - ebooks Account, 2015. – ISBN 1784396001,9781784396008

[19] Esmaeili, Ahmad ; Mozayani, Nasser ; Jahed Motlagh, Mohammad R. ; Matson, Eric T.: A socially-based distributed self-organizing algorithm for holonic multi-agent systems: Case study in a task environment. En: *Cognitive Systems Research* 43 (2017), jun, p. 21–44. – ISSN 13890417

[20] Franchi, E.: A Domain Specific Language Approach for Agent-Based Social Network Modeling. En: *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, IEEE, aug 2012. – ISBN 978–1–4673–2497–7, p. 607–612

[21] Frank H. P. Fitzek, Marcos D. K.: *Mobile Clouds: Exploiting Distributed Resources in Wireless, Mobile and Social Networks.* 1. Wiley, 2014. – ISBN 0470973897,9780470973899

[22] German Dario, Alvarez R.: *Construcción de un sub-sistema de software que permita generar la movilidad de agentes artificiales de acuerdo con el rol que ellos desempeñan sobre el sitema TLON cuyos recursos se comparten a través de una red Ad Hoc.* Cra 45, Bogotá, CO : Universidad Nacional de Colombia, 2017

[23] Gert Smolka (auth.), Peter Van Roy (.: *Multiparadigm Programming in Mozart/Oz: Second International Conference, MOZ 2004, Charleroi, Belgium, October 7-8, 2004, Revised Selected and Invited Papers.* 1. Springer-Verlag Berlin Heidelberg, 2005 (Lecture Notes in Computer Science 3389 : Programming and Software Engineering). – ISBN 3540250794,9783540250791

[24] Grimaldi, Ralph P.: *Discrete and combinatorial mathematics : an applied introduction.* 5th ed. Pearson Addison Wesley, 2004. – ISBN 0201726343,9780201726343

[25] Higashino, M. ; Hayakawa, T. ; Takahashi, K. ; Kawamura, T. ; Sugahara, K.: Management of Streaming Multimedia Content Using Mobile Agent Technology on Pure P2P-based Distributed E-learning System. En: *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, IEEE, mar 2013. – ISBN 978–1–4673–5550–6, p. 1041–1047

[26] Higashino, Masayuki ; Takahashi, Kenichi ; Kawamura, Takao ; Sugahara, Kazunori: Mobile Agent Migration Based on Code Caching. En: *2012 26th International Conference on Advanced Information Networking and Applications Workshops*, IEEE, mar 2012. – ISBN 978–1–4673–0867–0, p. 651–656

[27] Hooshangi, Navid ; Asghar Alesheikh, Ali: Agent-based task allocation under uncertainties in disaster environments: An approach to interval uncertainty. En: *International Journal of Disaster Risk Reduction* 24 (2017), sep, p. 160–171. – ISSN 22124209

[28] Horlait, Eric: *Mobile Agents for Telecommunication Applications.* Kogan Page Science, 2003 (Innovative technology series. Information systems and networks). – ISBN 9781903996287,1–9039–9628–7

[29] ILARRI, S ; MENA, E ; ILLARRAMENDI, A: Using cooperative mobile agents to monitor distributed and dynamic environments. En: *Information Sciences* 178 (2008), may, Nr. 9, p. 2105–2127. – ISSN 00200255

[30] Information Sciences Institute ; University of Southern California ; 4676 Admiralty Way ; Marina del Rey, California 9. *RFC 793 TRANSMISSION CONTROL PROTOCOL*. 1981

[31] Grupo de Investigación TLON, Universidad Nacional de C.: Grupo de Investigación en Redes de Telecomunicaciones Dinámicas y Lenguajes de Programación Distribuidos.

[32] Jha, Sudan: Implementation of the Mobile Agent System using Resource Allocation Problem. En: *International Journal of Multidisciplinary Approach & Studies* 2 (2015), Nr. 5, p. 44–56. – ISSN 2348537X

[33] John Goerzen, Brandon R.: *Foundations of Python 3 Network Programming, Second Edition*. 2. 2010. – ISBN 1430230037,9781430230038

[34] K., Thomson: *UNIX programmer's manual*. 1971

[35] Katzmarski, Bernhard ; Schomaker, Gunnar ; Nebel, Wolfgang: Mobile agents based on virtual machines to protect sensitive information. En: *Communications in Computer and Information Science* 470 (2014), p. 97–107. – ISSN 18650929

[36] Laura Cassell, Alan G.: *Python Projects*. 1. Wrox, 2014. – ISBN 111890866X,9781118908662

[37] Leach, Paul J. ; Mealling, Michael ; Salz, Rich: A Universally Unique IDentifier (UUID) URN Namespace.

[38] Lohani, Divya ; Varma, Shirshu: Dynamic mobile agent itinerary planning for collaborative processing in wireless sensor networks. En: *2015 Annual IEEE India Conference (INDICON)*, IEEE, dec 2015. – ISBN 978–1–4673–7399–9, p. 1–5

[39] Malik, Najmus S. ; Ko, David ; Cheng, Harry H.: A secure migration process for mobile agents. En: *Software: Practice and Experience* 41 (2011), jan, Nr. 1, p. 87–101. – ISSN 00380644

[40] Matthes, Eric: *Python Crash Course: A Hands-On, Project-Based Introduction to Programming*. No Starch Press, 2015. – ISBN 1593276036,9781593276034

[41] Matthes, Eric: *Python crash course : a hands-on, project-based introduction to programming*. San Francisco : No Starch Press, 2016. – ISBN 978–1593276034

[42] Matthias, Karl: *Docker : up and running*. Sebastopol, CA : O'Reilly, 2015. – ISBN 978–1491917572

[43] Mishchenko, Polina V.: Distributed computing system for solving applied tasks. En: *2015 16th International Conference of Young Specialists on Micro/Nanotechnologies and Electron Devices*, IEEE, jun 2015. – ISBN 978–1–4673–6718–9, p. 154–157

[44] N. S. Nithya ; K. Duraiswamy. *Control, Automation, Communication and Energy Conservation, 2009. INCACEC 2009. 2009 International Conference on.* 2009

[45] O'Connor, TJ: *Violent Python A Cookbook for Hackers, Forensic Analysts, Penetration Testers and Security Engineers.* Syngress, 2012. – ISBN 978–1597499576

[46] Peter Van Roy, Seif H.: *Concepts, techniques, and models of computer programming.* MIT Press, 2004. – ISBN 9780262220699,0262220695

[47] Pierfederici, Francesco: *Distributed Computing with Python.* Packt Publishing, 2016

[48] Portnoy, Matthew: *Virtualization Essentials.* 1. Sybex, 2012. – ISBN 1118176715,9781118176719

[49] Prapulla S B ; Chandra, Jayanth ; Mudakavi, Madhwesh B. ; Shobha G ; Thanuja T C: Multi Mobile Agent itinerary planning using Farthest Node First Nearest Node Next (FNFNNN) technique. En: *2016 International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS)*, IEEE, oct 2016. – ISBN 978–1–5090–1020–2, p. 105–111

[50] Qadori, Huthiafa ; Zulkarnain, Zuriati ; Hanapi, Zurina ; Subramaniam, Shamala: A Spawn Mobile Agent Itinerary Planning Approach for Energy-Efficient Data Gathering in Wireless Sensor Networks. En: *Sensors* 17 (2017), jun, Nr. 6, p. 1280. – ISSN 1424–8220

[51] Ramji, Tangudu: Adaptive resource allocation and its scheduling for good tradeoff between power consumption and latency in OFDMA based wireless distributed computing system. En: *2015 International Conference on Computation of Power, Energy, Information and Communication (ICCPEIC)*, IEEE, apr 2015. – ISBN 978–1–4673–6525–3, p. 0496–0501

[52] In: Satoh, Ichiro: *Mobile Agents.* Boston, MA : Springer US, 2006, p. 231–254. – ISBN 978–0–387–27972–5

[53] Scheifler, Robert W. ; Gettys, James: *X Window System : the complete reference to Xlib, X Protocol, ICCCM, XLFD.* Digital Press, 1992. – 1000 p.. – ISBN 1555580882

[54] Singh, Rajwinder ; Dave, Mayank: Antecedence Graph Approach to Checkpointing for Fault Tolerance in Mobile Agent Systems. En: *IEEE Transactions on Computers* 62 (2013), feb, Nr. 2, p. 247–258. – ISSN 0018–9340

[55] Slatkin, Brett: *Effective Python: 59 Specific Ways to Write Better Python (Effective Software Development Series)*. 1. Addison-Wesley Professional, 3 2015. – ISBN 9780134034287

[56] Terziyan, Joonas K. ; Vagan ; Alkhateeb, Faisal (Ed.): *Multi-Agent Systems - Modeling, Interactions, Simulations and Case Studies*. InTech, apr 2011. – ISBN 978–953–307–176–3

[57] Turnbull, James: *The Docker Book*. 2014. – ISBN 9780988820234

[58] University, Cyberpunk: *Python: The No-Nonsense Guide: Learn Python Programming Within 12 Hours!* CreateSpace Independent Publishing Platform, 1 2017. – ISBN 9781542589406

[59] USC/Information Sciences ; Institute. *RFC 768 User Datagram Protocol*. 1980

[60] Vigilson Prem, M ; Swamynathan, S: Mobile Agents for Information Retrieval: Detection and Recovery From Failures. En: *Arabian Journal for Science & Engineering (Springer Science & Business Media B.V.)* 39 (2014), Nr. 4, p. 2817–2829. – ISSN 13198025

[61] <weddy@grc.nasa.gov>, Wesley M. E.: TCP SYN Flooding Attacks and Common Mitigations.

[62] Wen, Heming ; Tiwary, Prabhat K. ; Le-Ngoc, Tho: *Wireless Virtualization*. Cham : Springer International Publishing, 2013 (SpringerBriefs in Computer Science). – ISBN 978–3–319–01290–2

[63] Weyns, Danny ; Ricci, Alessandro ; Holvoet, Tom ; Viroli, Mirko ; Schumacher, Michael: Environments for Multiagent Systems. 2005. – Report AgentLink Technical Forum Group, Ljubljana

[64] Yamanouchi, Akihiro ; Hashimoto, Takeshi ; Ohta, Tomoyuki ; Kojima, Hideharu ; Kakuda, Yoshiaki: Resource Management Middleware Using Mobile Agents for Mobile Ad Hoc Networks. En: *2010 IEEE 30th International Conference on Distributed Computing Systems Workshops*, IEEE, jun 2010. – ISBN 978–1–4244–7471–4, p. 1–6

[65] Yousuf, Farzana ; Zaman, Zahid: A Survey of Fault Tolerance Techniques in Mobile Agents and Mobile Agent Systems. En: *2009 Second International Conference on Environmental and Computer Science*, IEEE, 2009. – ISBN 978–1–4244–5590–4, p. 454–458