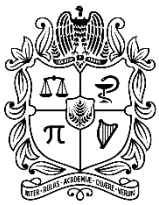

UNIVERSIDAD
NACIONAL
DE COLOMBIA

Construcción de un traductor para un lenguaje de programación para el modelo de computación social-inspirado TLÖN

Andrés Felipe De Orcajo Vélez

Universidad Nacional de Colombia
Facultad de Ingeniería, Departamento de Ingeniería de Sistemas e Industrial
Bogotá, Colombia
2020

Construcción de un traductor para un lenguaje de programación para el modelo de computación social-inspirado TLÖN

Andrés Felipe De Orcajo Vélez

Trabajo final presentado como requisito parcial para optar al título de:
Magíster en Ingeniería - Ingeniería de Sistemas y Computación

Director:
Ph.D. Jorge Eduardo Ortiz Triviño

Línea de Investigación: Computación Aplicada
Grupo de Investigación TLÖN (Grupo de Investigación en Lenguajes de Programación Distribuidos y Redes de Telecomunicaciones Dinámicas)

Universidad Nacional de Colombia
Facultad de Ingeniería, Departamento de Ingeniería de Sistemas e Industrial
Bogotá, Colombia
2020

Dedicatoria:

A mis padres y hermana.

Agradecimientos

Agradezco a mi alma mater, la Universidad Nacional de Colombia, cuyas oportunidades y enseñanzas que me otorgó son invaluables.

Agradezco a las labores docentes que aportaron a mi aprendizaje y preparación para la realización de este trabajo. A mi director y su constante interés por mi crecimiento académico.

Resumen

Este trabajo se enmarca dentro de recientes arquitecturas de computación distribuida donde se presentan características como la movilidad, recursos estocásticos y dinámicos, se crean nuevos retos que deben satisfacer sus herramientas como lo son los lenguajes de programación que permitan explotar sus potencialidades. En este trabajo final de maestría se presenta la creación de un lenguaje de programación y de un lenguaje intermedio, que son requeridos para escribir aplicaciones social-inspiradas dentro del modelo de computación TLÖN. A su vez, se muestra cómo fue el desarrollo de una herramienta traductora que genera código intermedio para su computador virtual distribuido. Esto conlleva a la propuesta de un modelo social de computación que toma elementos de diferentes modelos de computación concurrente, distribuida y móvil, como el modelo de actores y el cálculo de ambientes [42], junto con un amplio grupo de conceptos sociales, como lo son la institucionalidad [24] y los estados. Se presentan ejemplos que exhiben el potencial de la herramienta construida. Todo esto reunido, contribuye una propuesta de construir aplicaciones con esta herramienta, a través de una nueva forma de desarrollar aplicaciones inspiradas en la creación de una obra de teatro.

Palabras clave: modelo TLÖN, compiladores, lenguajes de programación, lenguaje intermedio, computador virtual distribuido, computación social-inspirada.

Abstract

This work is based on recent distributed computing architectures which presents characteristics like mobility, stochastic and dynamic resources. New goals must be satisfied by its tools such as programming languages that allow them to exploit their potential. On this final master's work, the creation of a programming language and an intermediate language are presented, which are essential to write social-inspired applications in the TLÖN computing model. On the other hand, the development of a translating tool which generates an intermediate code for virtual distributed computer is shown. This drives into the proposal of a social computing model that takes different concurrent distributed and mobile computing elements, such as the actor model and the ambient calculus [42], alongside a wide set of social concepts, like institutionalism [24] and states. Examples are shown where the potential of the built tool is evident. All this united, contributes to the proposal for building applications with this tool through a new application development method inspired on a theater play.

Keywords: TLÖN model, compilers, programming languages, intermediate language, distributed virtual computer, social-inspired computing.

Este Trabajo Final de maestría fue calificado en octubre de 2020 por el siguiente evaluador:

Felipe Restrepo Calle PhD.
Profesor Asociado - Departamento de Ingeniería de Sistemas e Industrial
Facultad de Ingeniería
Universidad Nacional de Colombia

Contenido

1. Introducción

Las arquitecturas computacionales clásicas, como la arquitectura de John Von Neumann [16], son determinísticas, estáticas y no poseen inteligencia artificial integrada en ellas. Evolucionaron en arquitecturas modernas que presentan nuevas características como la movilidad, además son estocásticas y dinámicas [15]. Estocásticas, ya que sus recursos ahora varían en el tiempo y se modelan con variables aleatorias; y dinámicas, porque los dispositivos (o nodos) que conforman el sistema son móviles. A partir de estas nuevas arquitecturas se crean retos para la creación de nuevas herramientas como los lenguajes de programación, los cuales deben permitir explotar las potencialidades de estos nuevos sistemas.
En el mundo moderno en el que nos encontramos actualmente, estamos rodeados de una gran variedad de dispositivos que se han integrado completamente a nuestras vidas y nos entregan un sinfín de herramientas. Estar inmersos en este mundo cada vez más automatizado y tecnológico nos permite realizar más procesos en menos tiempo, de manera simultánea y/o distribuida; y es gracias a esto que, la dificultad de las actividades que realizamos diariamente disminuye, permitiéndonos dar un paso adelante en la resolución de problemas cada vez más y más complejos. Por otro lado, poder lograr la combinación de los recursos (hardware) de estos dispositivos permitiría aumentar las posibilidades de cómputo y comunicación, por ejemplo, en lugares aislados del entorno citadino [15].

En las arquitecturas modernas de computación es donde se quiere implementar un modelo de computación social-inspirado, como lo propone del grupo de investigación TLÖN[1] con su computador virtual [4], que, de manera descentralizada, se concibe con la interconexión de los nodos de la red ad hoc que lo conforman, por lo tanto, no requiere de una infraestructura prediseñada.

Para escribir programas y aplicaciones que se ejecuten en el computador virtual de este sistema, es indispensable la creación de un lenguaje de programación y su compilador, ya que los lenguajes de programación existentes no están diseñados para este contexto. Siendo TLÖN, un modelo de computación social-inspirado que usa una variedad específica de agentes inteligentes (ver Subsección 2.2.2), el cual requiere que a través de un lenguaje de

[1] Grupo de Investigación en Redes de Telecomunicaciones Dinámicas y Lenguajes de Programación Distribuidos.

programación se puedan acceder a todas las capas de su modelo (ver Sección 2.2) de forma robusta y completa, junto con la capacidad de gestionar sus recursos.

Poder aprovechar las capacidades de computación del computador virtual de TLÖN aumentaría las posibilidades de realizar cómputo en múltiples entornos, de manera flexible y dinámica, donde los recursos disponibles de los nodos de la red ad hoc que lo conforman son vistos como un todo. De manera espontánea este computador virtual emergería con las solicitudes de los servicios que ofrece, que además serían realizadas por sus usuarios bajo criterios como la equidad y la cooperación. Así como en los lenguajes de programación distribuidos como el lenguaje CScript, que provee abstracciones para desarrollar aplicaciones de consistencia mixta, mediante la replicación de datos nativa [13]; o el lenguaje AmbientTalk, que se propone como solución para desarrollar aplicaciones móviles P2P [41].

El doctor Joaquín F. Sánchez, quien en su tesis doctoral "Prototipo de un lenguaje de programación para la implementación de servicios a través de comunidades de agentes social inspirados sobre redes ad hoc" [33], propuso un primer prototipo de un lenguaje de programación para el sistema de computación TLÖN, con el que logró modelar interacciones entre agentes inteligentes artificiales mediante el paso de mensajes, cuyas capacidades de procesamiento resuelven problemas computacionales dentro de sistemas embebidos. Pero debido a las divergencias con el paradigma social-inspirado que propone el grupo de investigación TLÖN con su modelo, se recurrió a la necesidad de proponer un nuevo lenguaje de programación con un modelo que sí logre evidenciar conceptos sociales y la filosofía original de este modelo. Por lo tanto, la pregunta de investigación que encapsula este proyecto, cuyo propósito es desarrollar un traductor que desde código escrito en lenguaje de programación social-inspirado genere código intermedio para el computador virtual del modelo TLÖN, es la siguiente:

¿Cómo desarrollar una herramienta de software traductora que a partir de un lenguaje de programación social-inspirado genere código intermedio para el computador virtual del sistema TLÖN?

Y para responder a esta pregunta, se formularon y se cumplieron los siguientes objetivos:

Objetivo general:

- Construir un traductor para un lenguaje de programación del modelo de computación social-inspirado TLÖN que permita la generación de código intermedio para su computador virtual.

Objetivos específicos:

1. Realizar un marco conceptual sobre bigrafos, compiladores, agentes sociales artificiales, redes ad hoc y el modelo de computación social-inspirado TLÖN.

2. Definir formalmente un conjunto de microinstrucciones y microservicios para el computador virtual del sistema TLÖN mediante bigrafos.[2]

3. Escribir la gramática libre de contexto de un lenguaje de programación bajo el paradigma social-inspirado para el sistema TLÖN usando E-BNF.

4. Escribir la gramática libre de contexto para el código intermedio del computador virtual del sistema TLÖN usando E-BNF.

5. Construir un generador de código intermedio que reciba como entrada código escrito en el lenguaje de programación propuesto anteriormente para sistema TLÖN utilizando ANTLR.

Se hizo uso de una metodología enfocada en la *investigación basada en el diseño* (*DRM*, en inglés), propuesta por Lucienne Blessing, quien afirma que este tipo de investigación tiene dos objetivos: la formulación y la validación de las teorías y modelos acerca de los fenómenos que se presentan en el diseño. La autora considera que para que haya un producto satisfactorio se deben haber cumplido una serie de objetivos, y que este producto satisfactorio se crea a partir de influencias [9]. Por lo tanto, aplicando la metodología *DRM* en este trabajo final de maestría, se realizó inicialmente una revisión literaria sobre los conceptos necesarios y teorías claves y la descripción de lo que es el modelo de computación social-inspirado TLÖN. Luego de conocer y estudiar las influencias, se definieron los objetivos general y específicos de este trabajo, los cuales se encargarían de determinar si el producto final era satisfactorio o no. Con el interés en cumplir los objetivos de la fase anterior, se procedió a diseñar y a escribir las gramáticas de los lenguajes de programación e intermedio con notación E-BNF, junto con la propuesta de un listado y el modelamiento de un conjunto de microinstrucciones y microservicios mediante bigrafos; cuyo aporte es importante para el modelamiento, adaptación y evolución del concepto del computador virtual del sistema TLÖN, al igual que lo que debiera ser y considerarse necesario para su compilador. Finalmente se desarrolló un intérprete traductor que genera código intermedio, con el que se presentaron y validaron diferentes escenarios de prueba. Y por último, al comparar el cumplimiento de los objetivos propuestos del producto final, se consideró que se logró crear un producto satisfactorio.

[2] Se eliminó el término "microinstrucciones-servicios" que estaba presente originalmente en este objetivo ya que se consideraba redundante.

Con los esfuerzos de investigación y de creación para este trabajo final de maestría, nace el lenguaje de programación social-inspirado DiUNisio[3] 2.0[4], junto con la definición de la gramática para el lenguaje intermedio LIST[5]; con el que también se presenta la propuesta de una nueva forma de escribir un programa social-inspirado, que integra conceptos de computación concurrente y distribuida haciendo uso de conceptos y sintaxis de modelos formales de computación como: el cálculo π, el cálculo join y los modelos de actores y de ambientes.

Para la adquisición de conocimiento necesario para el desarrollo del traductor, se asistió a la asignatura de Temas avanzados en lenguajes de programación dada por el docente Felipe Restrepo, cuyas enseñanzas fueron de gran utilidad para aprender conceptos y estrategias para la generación de código intermedio; donde también se desarrolló el primer prototipo del traductor para este proyecto, que desde el bosquejo de lo que debería ser un lenguaje de programación social-inspirado (con el lenguaje "TLÖNSocial") se generaba código para el primer prototipo de lenguaje intermedio ("TLÖN3AC", inspirado en el código de tres direcciones).

Este documento está estructurado así: el Capítulo 2 responde al primer objetivo específico; el Capítulo 3 junto con el Apéndice A y el Apéndice B al objetivo específico 3; el Capítulo 4 junto con el Apéndice C responde al objetivo específico 4; el Capítulo 5 junto con el Apéndice D y el Apéndice E al objetivo específico 2; y, finalmente, el Capítulo 6 y el Capítulo 7 cumplen con el objetivo específico número 5.

[3]Nombre inspirado en Dionisio de Tracia, un gramático al que se le atribuye la escritura de la primera gramática griega.

[4]Una versión actualizada de DiUNisio, un lenguaje de programación imperativo inspirado en la gramática de PASCAL [22] con palabras clave en español. Desarrollado por el autor de esta obra y Jorge E. Ortiz, en el año 2017.

[5]LIST, cuyas siglas se derivan de Lenguaje Intermedio para el Sistema TLÖN.

2. Marco teórico

Para la comprensión de este trabajo se van a presentar una serie de temas y conceptos que fundamentan las decisiones e inspiración tomadas para el desarrollo de una herramienta de software que traduce desde un lenguaje de programación hacia código intermedio (que requiere el sistema de computación TLÖN como aporte importante para su continuo desarrollo como modelo de computación).

Empezando con el concepto que se presenta a continuación, sobre el cual se cimienta el modelo de computación TLÖN, que son las redes ad hoc, en la Sección 2.1. Después se presenta el modelo TLÖN junto con su computador virtual y sus agentes artificiales en la Sección 2.2. Luego de esto, para la comprensión de lo que se requiere para construir un traductor (para que desde código escrito en lenguaje de programación se genere código intermedio), se expone el concepto de *compilador* en la Sección 2.3, el cual en este proyecto abarca las fases que van desde que recibe un *código origen* (escrito en un lenguaje de programación) hasta el punto en que se genera *código destino* (en este caso código intermedio). Para el diseño y construcción del lenguaje de programación se presentan una serie de modelos de computación teóricos en la Sección 2.4, los cuales fueron una importante fuente de inspiración para la sintaxis del lenguaje de programación que se presenta en el Capítulo 3. Luego se presentan los *bigrafos* en el Capítulo 5, que son un marco de referencia que permite modelar computación ubicua (la cual se da en el modelo de computación TLÖN), que fueron usados para modelar cómo sería la interacción de los componentes de hardware y software de los nodos y del computador virtual de TLÖN al momento de ejecutarse en el sistema. Y por último, se presentan en la Sección 2.6 trabajos relacionados al trabajo desarrollado en este documento.

2.1. Redes ad hoc

Las redes ad hoc, son redes que difieren de las redes tradicionales, en el sentido en que estas pueden ser móviles y dinámicas, ya que los dispositivos (o nodos) que la conforman están conectados por interfaces inalámbricas y pueden generar un comportamiento autoorganizativo y de autogestión: cada nodo es autónomo para gestionar los recursos que otorga y para la decisión sobre su participación en esta red. Los enlaces de los nodos que la conforman varían con el tiempo, ya que estos resultan de un comportamiento de adaptación y de auto-

organización [45].

Otras características de estas redes son:

- Se crean de forma espontánea, dado que sus enlaces son el producto de una reacción inmediata.
- Su control es descentralizado (Ver figura **2-1**), con el cual no se requiere de dispositivos intermediarios, los cuales funcionan como puntos de acceso, ya que la conexión se determina por cercanías entre nodos, estados de enlace y demás factores.
- Permiten la conexión y desconexión de sus nodos flexiblemente y, por ende, a partir de su autoadaptación se conformen nuevas organizaciones.
- Se pueden establecer, por ejemplo, en lugares alejados de fuentes de alimentación eléctrica (a cuestas del uso sus baterías), donde su potencial puede ser ilimitado.

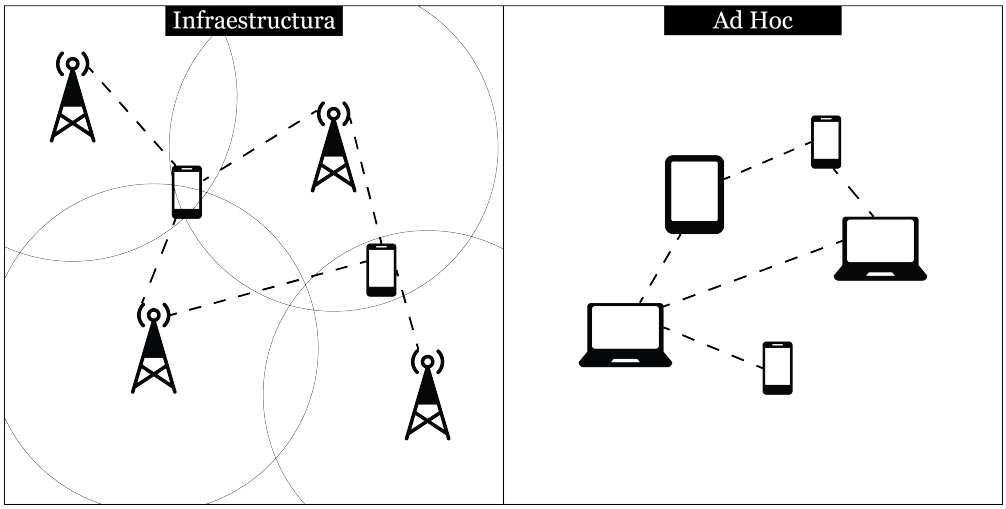

Figura 2-1.: Comparación entre una red basada en infraestructura (izquierda) con una red ad hoc (derecha). Adaptada de: https://learn.sparkfun.com/tutorials/connectivity-of-the-internet-of-things/infrastructure-and-ad-hoc-networks-.

2.2. Modelo de computación social-inspirado TLÖN

El modelo de computación social-inspirado TLÖN [4] es el resultado de una recopilación e integración de ideas de gran variedad de conceptos filosóficos y sociales, con los cuales

se concibe una metáfora de jerarquía social donde el territorio donde se implementa este sistema social son las redes ad hoc, como:

- La *Justicia* de John Rawls, que la define bajo el concepto de *equidad* a través de un acercamiento con el problema de la justicia distributiva. Rawls establece dos principios. El primero declara que todas las personas tienen derecho a libertades esenciales. Y el segundo, dice que es necesario solucionar las desigualdades sociales y económicas para esperar favorecer al colectivo. Para Rawls las instituciones deberían asegurarse de que todas las personas tengan acceso a ellas y también deberían tener un mecanismo de redistribución que los favorezca a todos por igual [31]. La idea de justicia se aplica al modelo TLÖN en el sentido de la distribución de los recursos sobre la red ad hoc, indicando que aunque haya diferencias entre los nodos del sistema, va a haber un trato de igualdad de condiciones sobre los participantes de la red.

- *Inmanencia* de Baruch de Spinoza, es una propiedad esencial de un sistema o de uno de sus elementos, que depende de ellos pero que puede ser estudiado o distinguirse de forma independiente. Se contrapone a la *trascendencia* [38]. Este concepto se relaciona en el sentido en que los conceptos de computación social-inspirada cobran sentido solo si se implementan en un sistema de este tipo, y pueden ser estudiados de manera independiente.

- *Paradigma* de Thomas Kuhn, que lo define como axiomas científicos universalmente reconocidos, los cuales presentan un modelamiento y una solución a los problemas de una comunidad científica [27]. Se relaciona al proyecto mediante la idea de paradigma de programación, que es donde se define una forma de programar diferente a los paradigmas tradicionales, que aunque presentan computación universal, la solución con ellos puede ser posible pero no viable.

- *Estado* de Thomas Hobbes, que presenta en su obra *Leviatán* [23] como una doctrina de derecho sobre los cimientos de las sociedades y los gobiernos legítimos. Donde define el Estado como un poder común cuya función es gobernar los elementos públicos y se establece a partir de la suma de voluntades individuales en busca de la adquisición de beneficios comunes. Admite tres tipos de estado: la democracia, la aristocracia y la monarquía [23]. Está relacionado a la idea de país y territorio del modelo TLÖN.

- *Existencia* y *Esencia* de Jean P. Sartre [34], conceptos que se elaboran en la Subsección 2.2.2.

El modelo TLÖN está definido como una pila de 4 capas (ver Figura **2-2**) las cuales son

atravesadas por 5 ejes longitudinales, donde vistas como un todo conforman un computador virtual. Las capas están definidas de la siguiente manera:

- La primera capa, la de la red ad hoc, representa lo que son físicamente las redes ad hoc donde se despliega este sistema.
- La segunda capa, introduce la virtualización de los recursos presentes de la capa inferior, para su gestión y su representación holística mediante un sistema operativo virtual.
- La tercera capa, aquella que define los agentes sociales artificiales integrantes de un sistema multiagente, los cuales se encargarán de realizar computación a través de sus interacciones.
- La cuarta, y última capa, es donde la coordinación y gestión de sistemas multiagente permiten la creación y despliegue de aplicaciones.

Y los ejes longitudinales se definen como:

- Lenguaje TLÖN, el lenguaje de programación que puede acceder a todas las capas del modelo debe ser capaz de gestionar cada elemento existente en el computador virtual.
- Seguridad, la cual debe estar presente en todas las capas del modelo, ya que es indispensable para el uso correcto y confiable del computador virtual.
- Dinámica del conocimiento, donde la interacción de los elementos del sistema debe ser dinámica y generarse de manera social.
- Calidad del servicio, el cual es necesario para mantener estabilidad en el uso del computador virtual.
- Network coding, la cual debe permitir la representación codificada en la red de los diferentes elementos que conforman el modelo social del computador virtual.

El modelo de computación TLÖN también introduce una metáfora social (ver Figura **2-3**) en la que sus diferentes capas juegan un rol dentro de esa sociedad virtual:

- La primera capa se refleja como el **Territorio**, que al igual que un país, es un entorno delimitado que contiene una frontera, donde los elementos que están en las capas superiores se desplazan y actúan.
- La segunda capa, que representa la **Institucionalidad**, es donde existe una jerarquía de instituciones, donde cada una es capaz de ofrecer servicios y gestión de recursos. Estas pueden simular diferentes entidades: de gobierno, de relaciones públicas, de comunicación, de seguridad, de transporte, etc.; al igual que en una sociedad tradicional.

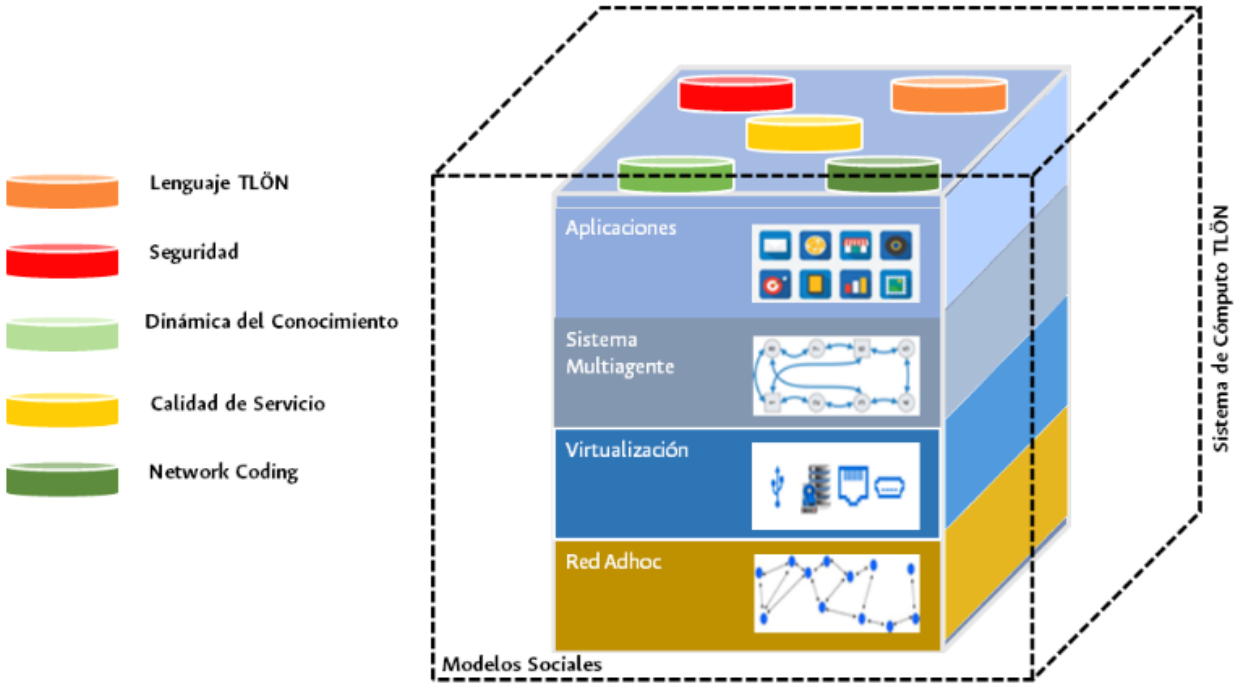

Figura 2-2.: Modelo de computación TLÖN. Tomada de: http://www.tlon.unal.edu.co/proyecto-tlon/acerca

- La tercera capa, la de las **Personas**, es aquella que representa cómo los agentes sociales artificiales son vistos como las personas de esta sociedad virtual, y que, gracias a ellas, es que los procesos de computación son realizables.

- La última capa, en este caso, es la que representa a las **Comunidades**, y es de ellas que se espera que el sistema ofrezca un repertorio de aplicaciones que estén disponibles para su ejecución.

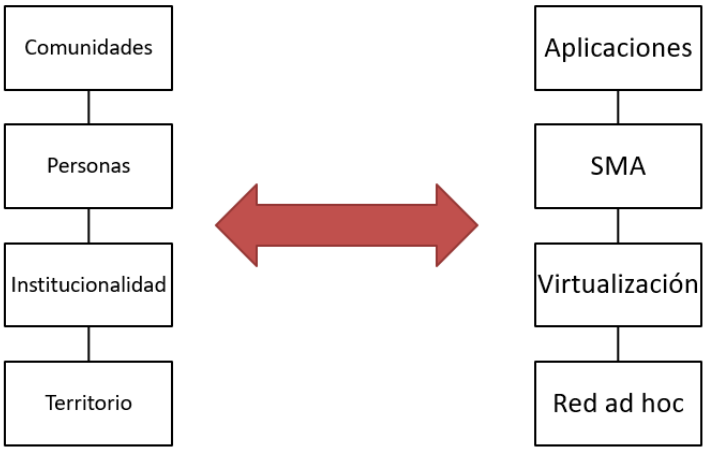

Figura 2-3.: Analogía del modelo social con el sistema TLÖN. Creación propia.

2.2.1. Computador virtual del Sistema TLÖN

El concepto abstracto del computador virtual del sistema TLÖN se define por simplicidad como un computador con la arquitectura de John Von Neumann, pero en este caso estocástica, debido a que los nodos de la red ad hoc que lo conforman, lo harán de forma dinámica en la medida en que se conecten, se desconecten, cambien sus enlaces, o cambien los recursos que le otorgan al sistema. Aun así, el computador virtual contendría los mismos componentes de esa arquitectura (ver Figura **2-4**), que son: la memoria, la unidad de control, la unidad aritmeticológica y un conjunto de dispositivos que sirven como entrada o salida. Este computador virtual también debe de contener y ejecutar su propio sistema operativo, el cual debe permitir la gestión de recursos distribuidos, generar redundancia de la información, entre otras funciones.

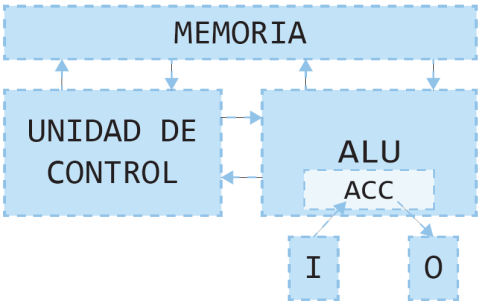

Figura 2-4.: Representación de una arquitectura Von Neumann para el computador virtual del sistema TLÖN. Creación propia.

2.2.2. Agentes sociales artificiales

La definición de agentes sociales artificiales del sistema TLÖN, llamados *Agentes Sartreanos* (ver Figura **2-5**), reciben este nombre, ya que se fundamentaron principalmente en los modelos filosóficos de Jean Paul Sartre, con los que describe su concepto de lo que es ser *persona* en su ensayo "El ser y la nada" [34]. En el que establece afirmaciones como: en las cosas la esencia es el existir; la existencia precede a la esencia; el cuerpo no es ser persona; para que haya persona debe haber existencia y esencia. También introduce conceptos como: *en sí*, en el que se piensa por sí mismo; *para sí*, donde se desenvuelve en un ambiente; y, por último, *para el otro*, donde se trasciende (se sale de sí mismo) para ser consciente de la existencia de otros, se actúa de manera coordinada para vivir en comunidad, existe sociabilidad y existen comportamientos altruistas y de convivencia.

Pero, así como un agente sartreano es social-inspirado, en este caso, también es bio-inspirado, ya que introduce conceptos como: *filogenia*, en la que existe evolución a largo plazo [40]; *ontogenia*, donde existe capacidad de reproducción [39]; y, *epigenia*, que permite la adaptación instantánea [12]. Y, por ejemplo, de estos conceptos también se inspiran técnicas como los autómatas celulares, las redes neuronales artificiales, los algoritmos genéticos, entre otros.

2.3. Compilador

El *compilador* (ver Figura **2-6**) es un programa que permite traducir desde un lenguaje *fuente* a otro lenguaje *destino* determinado. Una funcionalidad importante que ofrece este programa es la de reportar errores encontrados en el código fuente si fueron detectados en el proceso de traducción [3].

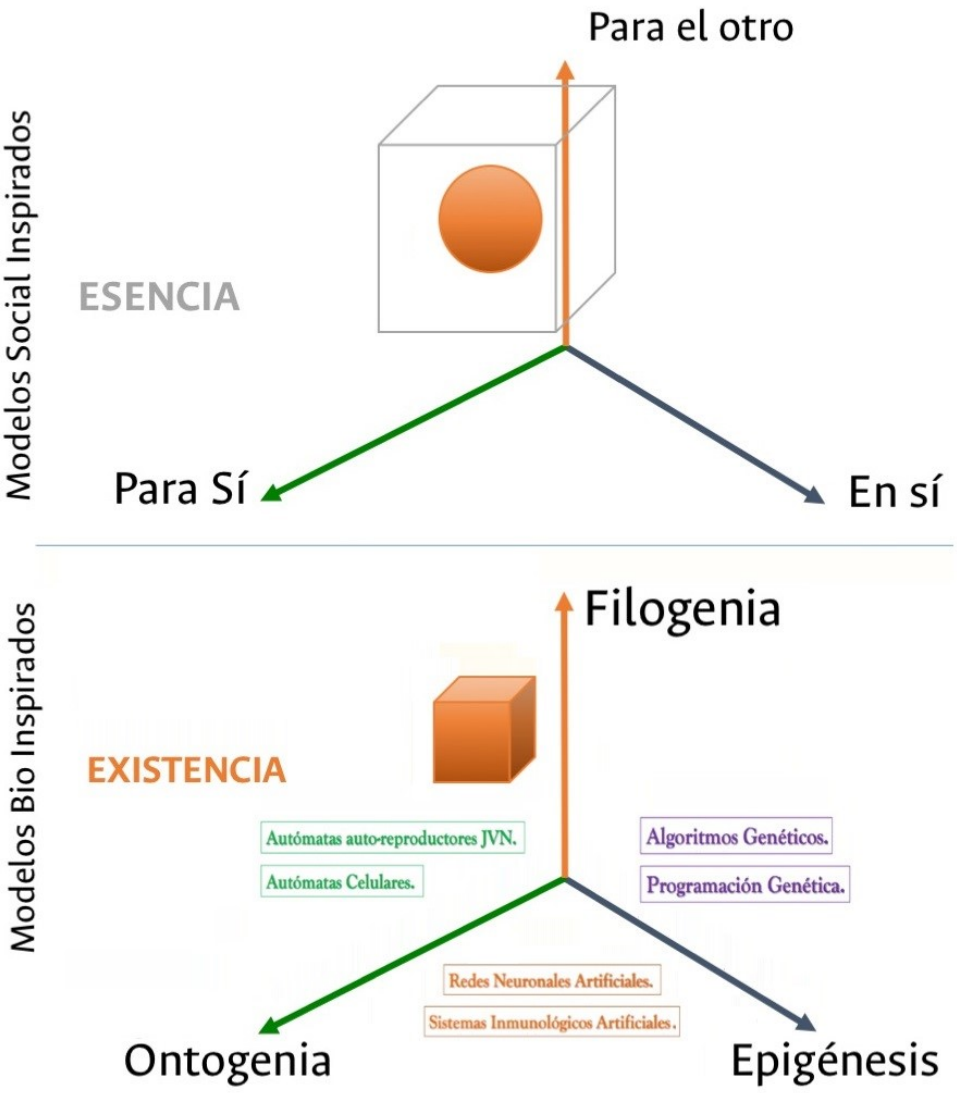

Figura 2-5.: Ejes que conforman el concepto de agente sartreano. Tomada de: http://www.tlon.unal.edu.co/proyecto-tlon/modelo-social

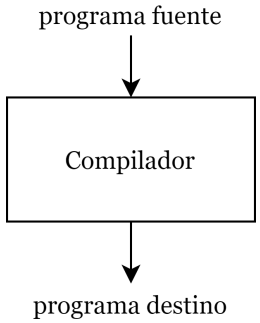

Figura 2-6.: Un compilador. Adaptada de [3].

A diferencia del compilador, está el *intérprete* (ver Figura **2-7**), el cual, en vez de realizar una traducción, lo que haces es directamente ejecutar las operaciones especificadas en una entrada escrita en el lenguaje fuente.

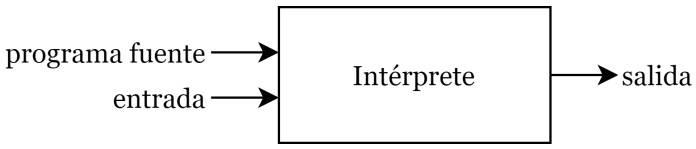

Figura 2-7.: Un intérprete. Adaptada de [3].

Junto con el compilador, otro grupo de programas pueden ser requeridos para crear un programa destino ejecutable (ver Figura **2-8**). Para lograr esto, se deben seguir una serie de pasos, los cuales realizan una serie de transformaciones a la entrada inicial:

1. El código fuente entra en un *preprocesador*, el cual sirve para expandir formas abreviadas en código fuente.

2. Luego, este código entra en el *compilador*, el cual puede producir un programa escrito en código ensamblador como salida.

3. A continuación, este código ensamblador se procesa en un programa llamado *ensamblador*, el cual produce código máquina reubicable.

4. Los programas de gran tamaño usualmente son compilados en múltiples piezas, por lo que el código puede que sea enlazado con otro archivo reubicable y con archivos de librerías dentro del código que se va a ejecutar en la máquina. Por lo que el *enlazador* se encarga de resolver direcciones de memoria externas donde el código en un archivo puede referirse a la ubicación en otro archivo.

5. El *cargador* luego junta todos los archivos ejecutables en la memoria para su ejecución.

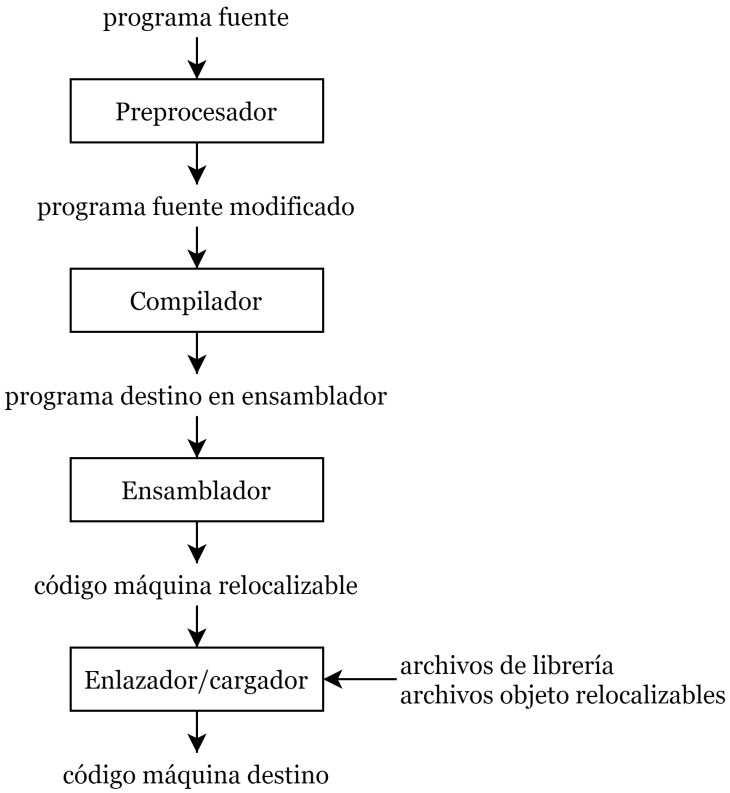

Figura 2-8.: Sistema de procesamiento de lenguajes. Adaptada de [3].

2.3.1. Estructura del compilador

Dentro de un compilador se ejecutan dos procesos, el de *análisis* y el de *síntesis*. El proceso de análisis segmenta el código fuente en piezas a las cuales les impone una estructura gramatical. Si en este paso se detectan sentencias mal estructuradas se deben mostrar mensajes indicando información del error. En esta fase también se recolecta información sobre el código fuente y lo almacena en una estructura llamada *tabla de símbolos*, la cual se entrega junto con la representación intermedia a la siguiente fase, la de síntesis. El proceso de síntesis construye el programa destino deseado desde la representación intermedia y con la información almacenada en la tabla de símbolos [3].

Con más detalle, el proceso de compilación opera con una secuencia de fases, cada una encargada en transformar de una representación a otra el código fuente. Una descomposición típica del compilador se muestra en la Figura **2-9**. La tabla de símbolos es usualmente usada

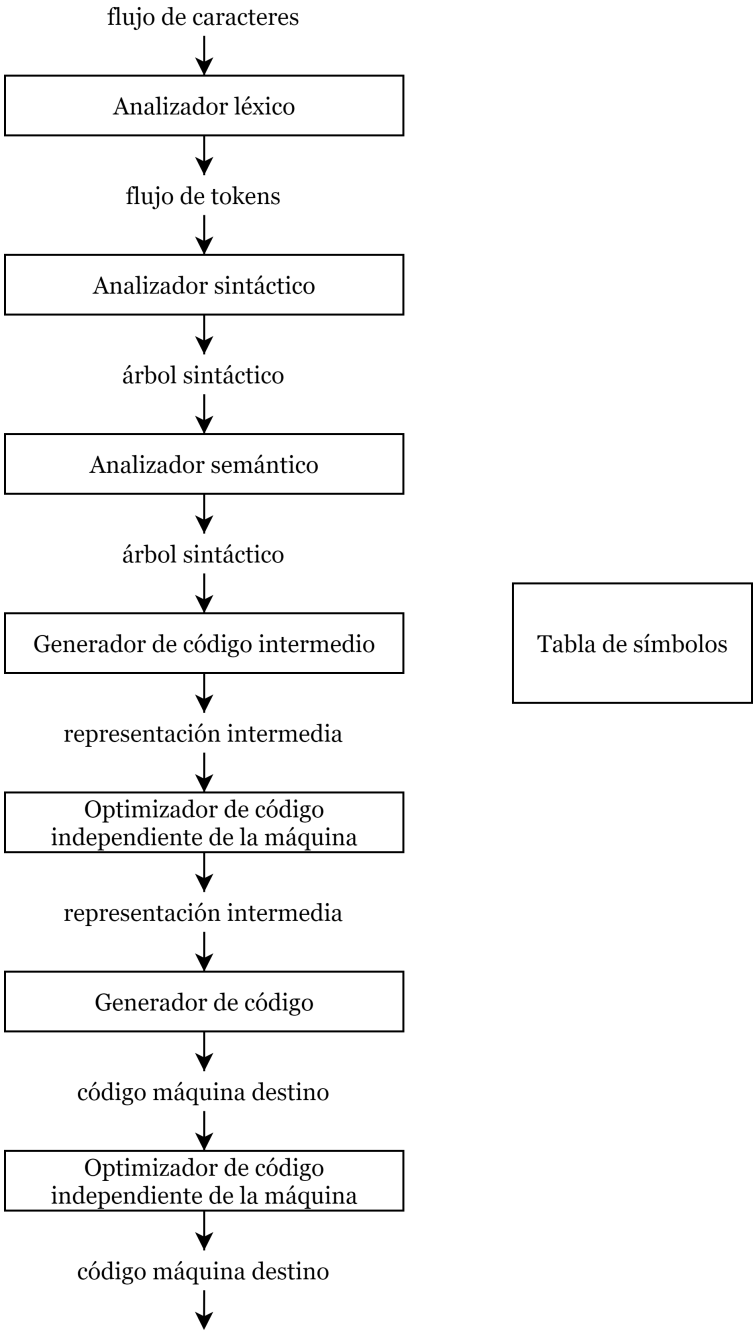

Figura 2-9.: Fases de un compilador. Adaptada de [3].

por todas las fases del compilador.

- **Fase 1: Análisis léxico.** El analizador léxico se encarga de leer el flujo de caracteres de la entrada agrupándolos en secuencias con sentido llamadas *lexemas*. Para cada lexema, el analizador produce una salida llamada *token*, el cual es enviado a la siguiente fase.
- **Fase 2: Análisis sintáctico.** El analizador sintáctico usa los tokens producidos en la fase anterior para crear una representación intermedia en un *árbol*, donde los nodos internos representan las operaciones y las hojas (o hijos) representan los argumentos de esa operación, el cual estructura de manera gramática el flujo de tokens.
- **Fase 3: Análisis semántico.** El analizador semántico usa el árbol de sintaxis y la información de la tabla de símbolos para comprobar consistencia semántica en el código fuente con la definición del lenguaje. También almacena información ya sea en el árbol de sintaxis o en la tabla de símbolos que podría ser usada en las siguientes fases.
- **Fase 4: Generación de código intermedio.** En el proceso de construcción de un código fuente a código destino, un compilador puede construir una o más *representaciones intermedias*, las cuales pueden tener variedad de formas. Muchos compiladores pueden generar un código de bajo nivel o representación intermedia tipo máquina, el cual debe ser fácil de producir y fácil de traducir en código máquina destino.
- **Fase 5: Optimización del código.** La fase de optimización de código independiente de la máquina intenta mejorar el código intermedio para que un mejor código destino sea generado, ya sea en el sentido de tiempo de ejecución, en el tamaño del código o que consuma menos energía.
- **Fase 6: Generación de código.** El generador de código toma como entrada una representación intermedia del código fuente y lo representa en código del lenguaje destino. Si el lenguaje destino es *código máquina*, las ubicaciones de los registros o de memoria son escogidos para cada una de las variables usadas por el programa. Luego, las instrucciones intermedias son traducidas en secuencias de instrucciones de máquina que realizan la misma tarea.

2.3.2. Gestión de la tabla de símbolos

Una función esencial de un compilador es almacenar nombres de variables usados en el programa fuente y recolectar información sobre varios atributos de cada nombre. Estos atributos pueden proveer información sobre el espacio reservado de un nombre, su tipo y su alcance

(en el cual su valor puede ser usado en el programa); y, en el caso de nombres de procedimiento, propiedades como: los números y los tipos de argumentos, la forma en la que se entregan los argumentos (por ejemplo, si es por valor o por referencia) y el tipo de retorno [3].

La *tabla de símbolos* es una estructura de datos que contiene un registro por cada nombre de variable, con campos como los atributos del nombre. La estructura de datos debería ser diseñada para permitirle al compilador encontrar el registro de cada nombre y también poder almacenar o encontrar datos de un registro rápidamente.

2.4. Modelos de computación

Los modelos teóricos de computación son importantes para poder razonar sobre propiedades del software, incluyendo los lenguajes de programación usados para desarrollar software. Ejemplos de propiedades del software serían: poder de expresividad, correctitud, rendimiento y confiabilidad. Los modelos de computación también permiten a los diseñadores de lenguajes de programación y desarrolladores de aplicaciones organizar artefactos de software para manejar la complejidad y promover la reusabilidad y componibilidad [42]. A continuación, se presentan brevemente estos modelos.

2.4.1. Cálculo lambda

Church y Kleene crearon el cálculo λ en los años 30 [11]. El corazón de los lenguajes de programación funcional. Es Turing completo, lo que quiere decir que, cualquier función computable puede ser expresada y evaluada usando este cálculo. Es útil para estudiar conceptos de los lenguajes de programación conceptual debido a su alto nivel de abstracción [42].

La sintaxis de este cálculo está descrita en la Figura **2-10**:

$$
\begin{array}{lll}
e ::= & v & \text{- variable} \\
| & \lambda v.e & \text{- abstracción funcional} \\
| & (e\ e) & \text{- aplicación de la función}
\end{array}
$$

Figura 2-10.: Sintaxis del cálculo λ. Adaptada de [42].

Por ejemplo, dada la función $f(x) = x^2, f : Z \to Z$, evaluando $f(2) = 4$, se escribe así:

$$(\lambda x.x^2\ 2) \to 2^2 \to 4, \tag{2-1}$$

donde la variable es x y el parámetro es 2. Su visualización se muestra en la Figura **2-11**.

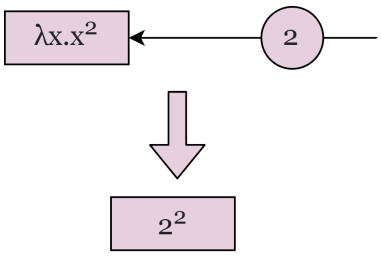

Figura 2-11.: Representación pictográfica del cálculo λ. Adaptada de [42].

Modelos teóricos de concurrencia, distribución y movilidad

A continuación, se presentan conceptos que se usarán con los siguientes modelos de computación:

- La **concurrencia** se puede considerar como un paralelismo potencial. Cuando múltiples procesadores son usados para ejecutar una aplicación concurrente, decimos entonces que es una *ejecución paralela* del programa concurrente. Si los procesadores están separados geográficamente, entonces decimos que es una *ejecución distribuida* del programa concurrente. Si la ubicación de las actividades puede cambiar dinámicamente, entonces decimos que es una *ejecución móvil* del programa concurrente. A diferencia de los programas secuenciales, los programas concurrentes son usualmente no determinísticos: diferentes ejecuciones del programa pueden producir resultados diferentes [42].

- La **computación distribuida** se refiere a la computación que se propaga sobre el espacio y el tiempo. Los sistemas distribuidos son más difíciles de razonar y de desarrollar que los sistemas centralizados. Los estándares son usualmente requeridos para permitir la interacción de componentes desarrollados en orígenes diferentes.

- La **movilidad** se refiere a los sistemas móviles distribuidos, los cuales presentan código móvil, datos móviles, dispositivos móviles y/o usuarios móviles. Típicamente los dispositivos móviles son altamente heterogéneos, y pueden tener restricciones importantes en términos de velocidad de procesamiento, acceso inalámbrico a la red, tamaño de la memoria y disponibilidad (debido al uso de baterías con tiempo limitado de uso).

2.4.2. Cálculo pi

Milner et al. crearon el cálculo π en el año 1992 [29], el cual modela computación concurrente en términos de comunicación de procesos, los cuales pueden ser móviles, ya que se puede cambiar dinámicamente su topología de comunicación. También desarrollaron una teoría de equivalencia de procesos que usa una técnica de bisimulación, donde dos o más procesos son considerados equivalentes si ellos pueden indefinidamente mimetizar cada una de las interacciones del otro dentro de cualquier entorno [42].

Una simple interacción entre dos procesos sobre un canal de comunicación a puede ser modelada como en la ecuación 2-2.

$$a(x).P|\overline{a}b.Q \tag{2-2}$$

En esta expresión, hay dos procesos concurrentes en ejecución (separados por |). El primer proceso denotado por $a(x).P$ está bloqueado, esperando a otro proceso que escriba un valor sobre el canal a, el cual va a reemplazar el valor a en el proceso P en la comunicación. El segundo proceso denotado por $\overline{a}b.Q$ está similarmente bloqueado, esperando a otro proceso para leer el valor b sobre el canal a, antes de continuar como Q. Este ejemplo ilustra un proceso de comunicación. Una forma visual de la representación de esta comunicación está representada en la Figura **2-12**.

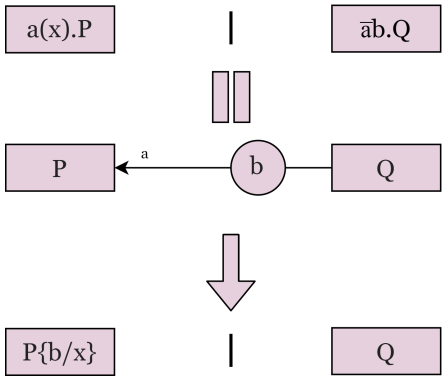

Figura 2-12.: Representación pictográfica del cálculo π. Adaptada de [42].

La sintaxis de este cálculo se muestra en la Figura **2-13**.

α	::=	$\bar{c}x$	Escribir x en el canal c
		$\mid c(x)$	Leer x en el canal c
		$\mid \tau$	No hay interacción
P, Q	::=	o	Proceso vacío
		$\alpha.P$	Proceso con prefijo
		$P \mid Q$	Composición concurrente de P y Q
		$P + Q$	Selección no determinística de P o Q
		$(\nu c)P$	Nuevo canal c, alcance restringido a P
		si $x = y \rightarrow P$	Ejecución condicional (emparejamiento)
		si $x \neq y \rightarrow P$	Ejecución condicional (discordancia)
		$A(y_1, ..., y_n)$	Invocación de un proceso
Δ	::=	$A(x_1, ..., x_n) \triangleq P$	Declaración de un proceso

Figura 2-13.: Sintaxis del cálculo π. Adaptada de [42].

2.4.3. Cálculo join

Fournet y Gonthier crearon el cálculo join para modelar computación concurrente como reacciones químicas, donde expresiones complejas pueden aparecer como el resultado de combinar expresiones simples de acuerdo con un conjunto dinámico de reglas de reescritura bidireccional [18]. La transformación de expresiones simples en expresiones más complejas y viceversa está inspirada en el calentar y enfriar moléculas en las reacciones químicas [42].

El cálculo join modela la computación como un conjunto de sitios de reacción química, donde los procesos, modelados como moléculas (los cuales se componen de átomos) interactúan de acuerdo con reglas locales, las cuales pueden variar en el tiempo. Las reglas de reacción definen patrones que deben ser emparejados por las moléculas en el sitio en el que las reacciones toman lugar. En respuesta a una reacción química, las moléculas desaparecen y nuevas moléculas pueden aparecer en el sitio de reacción. También en respuesta a una reacción atómica, nuevas reglas de reacción pueden ser creadas, las cuales hacen que el modelo sea reflectivo. Los patrones de las reglas de reacción definen nombres con un alcance estático, afectando únicamente las moléculas locales. La distribución y movilidad de procesos son modeladas a medida que se calienten o se enfríen las reglas de reacción para permitir una interacción entre moléculas [42]. Su sintaxis se muestra en la Figura **2-14**.

En el ejemplo de la Figura **2-15** se tiene un conjunto de tres reglas de reacción que como resultado producen lo que está a la derecha.

P	::=		Procesos
		$x\langle\bar{v}\rangle$	*Átomo*
		def D in P	*Definición*
		$P \mid P$	*Molécula*
D	::=		Definiciones
		$J \triangleright P$	*Regla de reacción*
		$D \wedge D$	*Constructor múltiple*
J	::=		Patrones join
		$x\langle\bar{v}\rangle$	*Patrón de un átomo*
		$J \mid J$	*Patrón de una molécula*

Figura 2-14.: Sintaxis del cálculo join. Adaptada de [42].

$$D = ready\langle printer\rangle \mid job\langle file\rangle \triangleright printer\langle file\rangle$$
$$D \vdash ready\langle laser\rangle \mid job\langle f1\rangle$$
$$D \vdash laser\langle f1\rangle$$

Figura 2-15.: Ejemplo escrito con la sintaxis del cálculo join. Adaptada de [42].

Y una representación pictórica de una molécula que cumpla con el *patrón 2* (de la Figura **2-15**) se muestra en la Figura **2-16**.

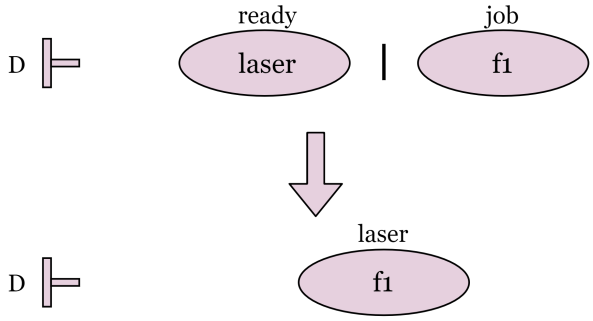

Figura 2-16.: Representación pictórica cálculo join. Adaptada de [42].

2.4.4. Modelo de actores

Hewitt et al. crearon el modelo de actores de un sistema concurrente en los años 70 [21]. Agha refinó el modelo para sistemas distribuidos abiertos en 1986 [2]. Los actores modelan unidades de computación concurrente, los cuales se comunican de manera asíncrona y ga-

rantizan el paso de mensajes [42].

El modelo de actores restringe explícitamente la memoria compartida, y la única forma de que un actor se comunique con otro actor es mediante el paso de mensajes, los cuales requieren conocer el nombre único de ese actor. Los mensajes son recibidos y procesados asíncronamente, lo cual hace que los actores modelen de una forma más natural los sistemas móviles y distribuidos. El modelo permite concurrencia sin fronteras al permitir a los actores crear nuevos actores en respuesta a mensajes. La creación de actores es síncrona, retornando el nombre del actor, el cual puede ser usado para comunicaciones futuras con el actor, o puede ser enviado a través de mensajes a otros actores, generando una topología de comunicación dinámica. La *justicia* es una propiedad clave de los sistemas de actores, lo que permite razonamiento composicional, donde la entrega de los mensajes está garantizada, y un actor está infinitamente disponible para procesar un mensaje eventualmente procesa ese mensaje [42].

Los mensajes son estructuras de datos no complejas, que son inmutables después de haberse creado. Un actor es similar a una cola de mensajes (ver Figura **2-17**), el cual convierte cada mensaje como un dato de entrada, por lo tanto, el actor debe saber cómo procesar cada mensaje.

Figura 2-17.: Cola de mensajes. Adaptada de: https://medium.com/techwomenc/modelo-de-actores-con-akka-225c28a4dff4

Las reacciones de un actor ante un mensaje son las siguientes:

- **Enviar mensajes:** siempre se hace de manera asíncrona. El orden de los mensajes se mantiene entre un actor emisor y uno receptor. Ver Figura **2-18**.

Figura 2-18.: Envío de mensajes. Adaptada de: https://medium.com/techwomenc/modelo-de-actores-con-akka-225c28a4dff4

- **Crear nuevos actores:** esto genera una jerarquía, donde los nuevos actores son hijos del actor generador. Ver Figura **2-19**.

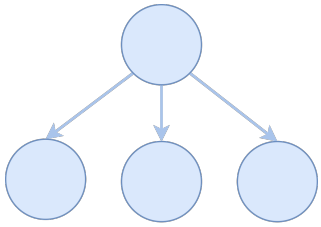

Figura 2-19.: Creación de nuevos actores. Adaptada de: https://medium.com/techwomenc/modelo-de-actores-con-akka-225c28a4dff4

- **Cambiar sus estados internos y su comportamiento:** un actor funciona como una máquina de estados, lo que hace que cuando reciba un mensaje pueda hacer la transición a un estado nuevo. Ver Figura **2-20**.

Figura 2-20.: Cambio de estados internos. Adaptada de: https://medium.com/techwomenc/modelo-de-actores-con-akka-225c28a4dff4

- **Gestionar los actores que ha creado:** al supervisar sus actores hijo, él puede decidir qué hacer cuando estos fallen. Recibirá una notificación con mensajes que contengan información sobre qué actor hijo ha fallado y su razón, entonces decidirá si lo reinicia o lo destruye. Ver Figura **2-21**.

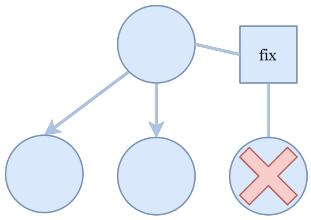

Figura 2-21.: Gestión de actores hijo. Adaptada de: https://medium.com/techwomenc/modelo-de-actores-con-akka-225c28a4dff4

2.4.5. Cálculo de ambientes

Cardelli y Gordon crearon la teoría de ambientes móviles para modelar código, dispositivos y usuarios móviles [10]. El cálculo de ambientes define una estructura jerárquica de ambientes

que contienen procesos y otros ambientes. Los procesos son restringidos a solo comunicarse dentro del mismo ambiente. Las primitivas en este cálculo permiten el movimiento de ambientes hacia adentro o hacia afuera de otros ambientes. También como la creación y destrucción de fronteras de ambiente [42]. Su sintaxis se muestra en Figura **2-22**.

M ::=	x	Variable x
	$in\ \mathbf{n}$	Entrar al ambiente n
	$out\ \mathbf{n}$	Salir del ambiente n
	$open\ \mathbf{n}$	Abrir ambiente n
	$M.M$	Camino de capacidades
P,Q ::=	o	Proceso vacío
	$P \mid Q$	Composición concurrente de P y Q
	$!P$	Replicación de P
	$(\upsilon n)P$	Nuevo nombre n, alcance restringido a P
	$M[P]$	Creación del ambiente
	$M.P$	Acción síncrona
	$<M>$	Salida asíncrona: escribir M
	$(x).P$	Entrada síncrona: leer x

Figura 2-22.: Sintaxis del cálculo de ambientes. Adaptada de [42].

Una interacción simple entre dos procesos puede ser escrita como la ecuación 2-3.

$$m[p[\text{out } m.\text{in } n.\langle M\rangle]]|n[\text{open } p.(x).Q]. \tag{2-3}$$

En esta expresión, los ambientes m y n pueden ser considerados como el modelamiento de dos máquinas en una red, el ambiente p puede considerarse como un paquete moviéndose desde la máquina m a la máquina n. Continuando con el movimiento y apertura del ambiente p, la comunicación interproceso ocurre dentro del ambiente n.

Cabe denotar que los ambientes pueden contener otros ambientes para formar una estructura jerárquica, también que las primitivas de movimiento de ambientes y destrucción se realiza de manera síncrona. En este ejemplo, el proceso out m.in $n.\langle M\rangle$ (dentro del ambiente p) no puede proceder hasta que out m ha sido ejecutado. Una representación gráfica de este proceso se se muestra en Figura **2-23**.

2.5. Bigrafos

Los bigrafos, propuestos por Robin Milner, son un marco de referencia que puede ser utilizado para el modelamiento de la computación ubicua [30]. Se fundamentan en el cálculo

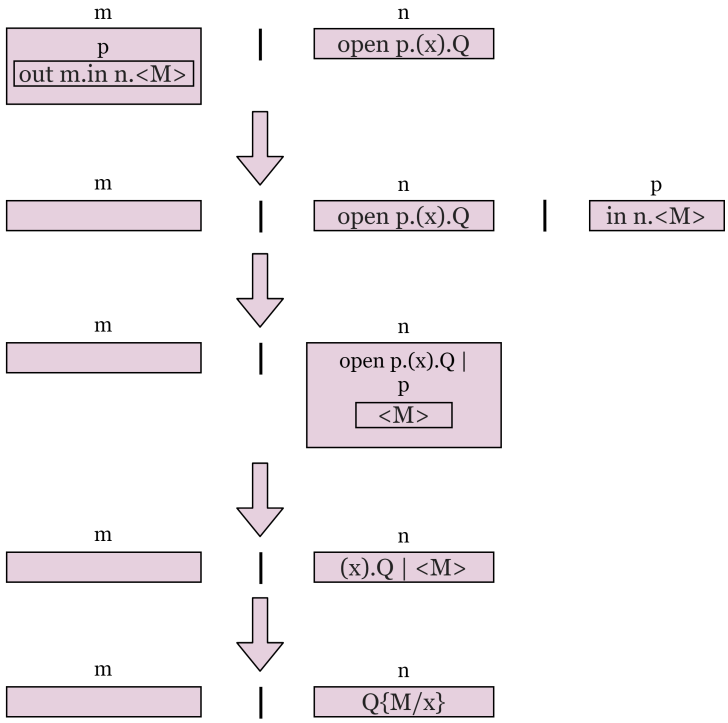

Figura 2-23.: Representación gráfica del cálculo de ambientes. Adaptada de [42].

de acciones, y recurre a múltiples conceptos de los modelos de computación presentados en la Sección 2.4, junto con otras ideas obtenidas de *InteractionNets*: un modelo gráfico de computación usado como una generalización de las pruebas de las estructuras de la lógica lineal [28]; y la *teoría de sistemas reactivos*: que son sistemas responsivos, resilientes, elásticos y dirigidos por mensajes [1]; entre otros. Su objetivo principal es poder modelar sistemas donde la localización y la conectividad son fundamentales junto con la integración de diferentes teorías acerca de computación concurrente, distribuida y móvil. Los bigrafos pueden ser modelados (ver Figura **2-24**) con la superposición de un *grafo de enlace* (un grafo) y un *grafo de lugar* (un conjunto de grafos tipo árbol), de los cuales, en ese mismo orden, el primero sirve para denotar cómo están enlazados los *nodos* del sistema, y el segundo, para representar la jerarquía en la que se contienen estos nodos.

Un bigrafo (ver Figura **2-25**) puede estar asociado a una o más *regiones* las cuales son las raíces del grafo de lugar, y cero o más *sitios libres* en este grafo de lugar, dentro de los cuales otras regiones pueden ser insertadas. Al igual que los nodos, se les pueden asignar *controles* que definen sus identidades y su aridad (el número de *puertos* que posee un nodo en el cual pueden conectarse a otros nodos a través de enlaces). Estos controles son tomados de una *firma*, la cual indica la forma de acceder a los nodos. En el grafo de enlace definimos los

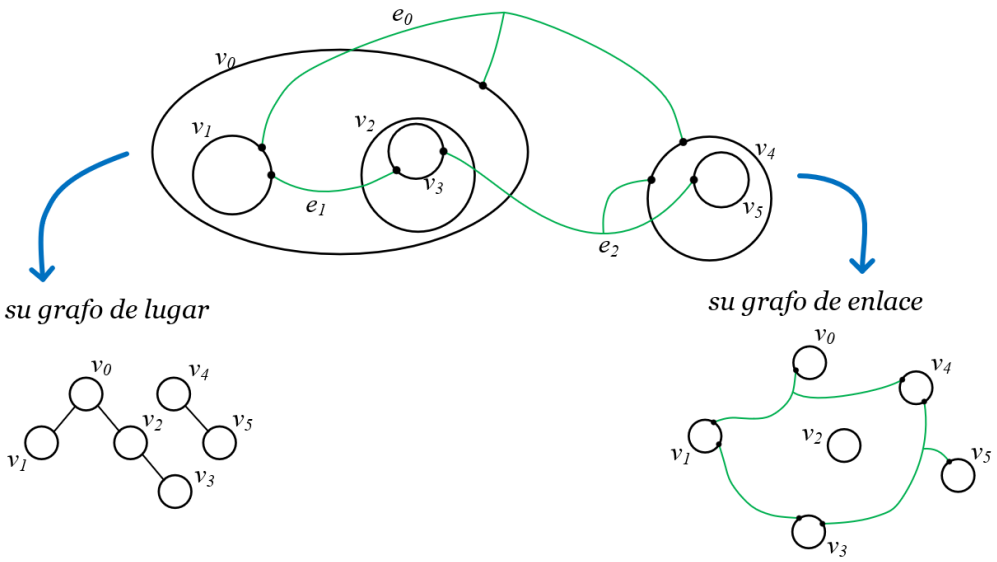

Figura 2-24.: Descomposición de un bigrafo. Adaptada de: https://www.cl.cam.ac.uk/archive/rm135/Bigraphs-Lectures.pdf

nombres internos y *externos*, los cuales determinan los puntos de conexión por los cuales nombres coincidentes pueden ser fusionados por un único enlace [44].

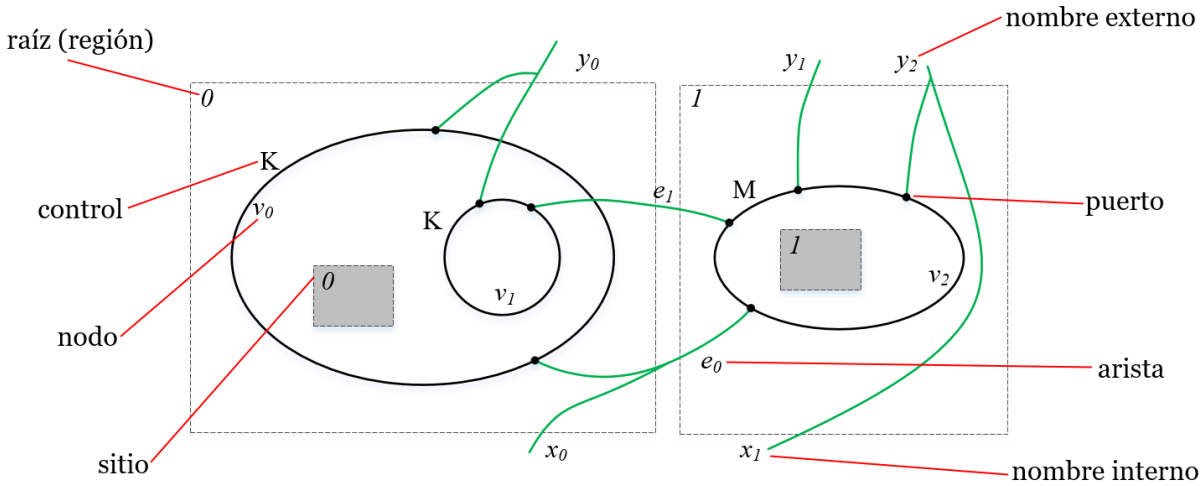

Figura 2-25.: Representación visual de un bigrafo. Adaptada de: https://www.cl.cam.ac.uk/archive/rm135/Bigraphs-Lectures.pdf

Un bigrafo es una tupla: $(V, E, ctrl, prnt, link) : \langle n, X\rangle \to \langle m, Y\rangle$,
Donde V es un conjunto de nodos, E es un conjunto de enlaces, *ctrl* es el mapa de control que asigna controles a los nodos, *prnt* es el mapa que define el anidamiento de los nodos, y *link*, es el mapa de enlaces que define la estructura de los enlaces del bigrafo.

La notación $\langle n, X\rangle \to \langle m, Y\rangle$ indica que el bigrafo tiene n sitios libres y un conjunto de nombres internos X, y m regiones con un conjunto Y de nombres externos. Los cuales son conocidos respectivamente como *interfaces internas* y *externas* del bigrafo [30].

Los bigrafos pueden cambiar de un estado a otro a través de la aplicación de *reglas de reacción*. Las cuales consisten en la transición desde un patrón inicial llamado *redex* hacia un estado nuevo dado por un patrón de cambio. Los enlaces se representan con la notación usada en la Figura **2-26**. Nuevos bigrafos pueden generarse a partir de otros bigrafos mediante operaciones (ver Figura **2-27**) de *composición* (o *anidamiento*), y el *producto paralelo* y el *producto de unión*, los cuales se definen en los puertos del bigrafo. En la composición de bigrafos se conectan los sitios libres del primer bigrafo con las raíces del segundo bigrafo. Para el producto paralelo de dos bigrafos, todos los puntos de los nombres externos en común del primer bigrafo se conectan con los del segundo bigrafo. En el producto de unión, ambos bigrafos quedan contenidos en la misma región, así como el enlazamiento por los nombres que tengan en común.

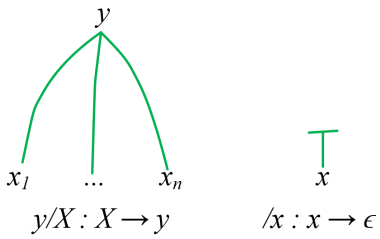

Figura 2-26.: Notación de enlaces de un bigrafo. Adaptada de: https://www.cl.cam.ac.uk/archive/rm135/Bigraphs-Lectures.pdf

2.6. Trabajos relacionados

A continuación se presenta una serie de trabajos relacionados al desarrollado en este documento. Comparten en común ideas y conceptos como la computación paralela y concurrente. Pero en este caso representan su aplicabilidad en otros ámbitos, como lo son, la robótica y la computación P2P móvil, entre otros.

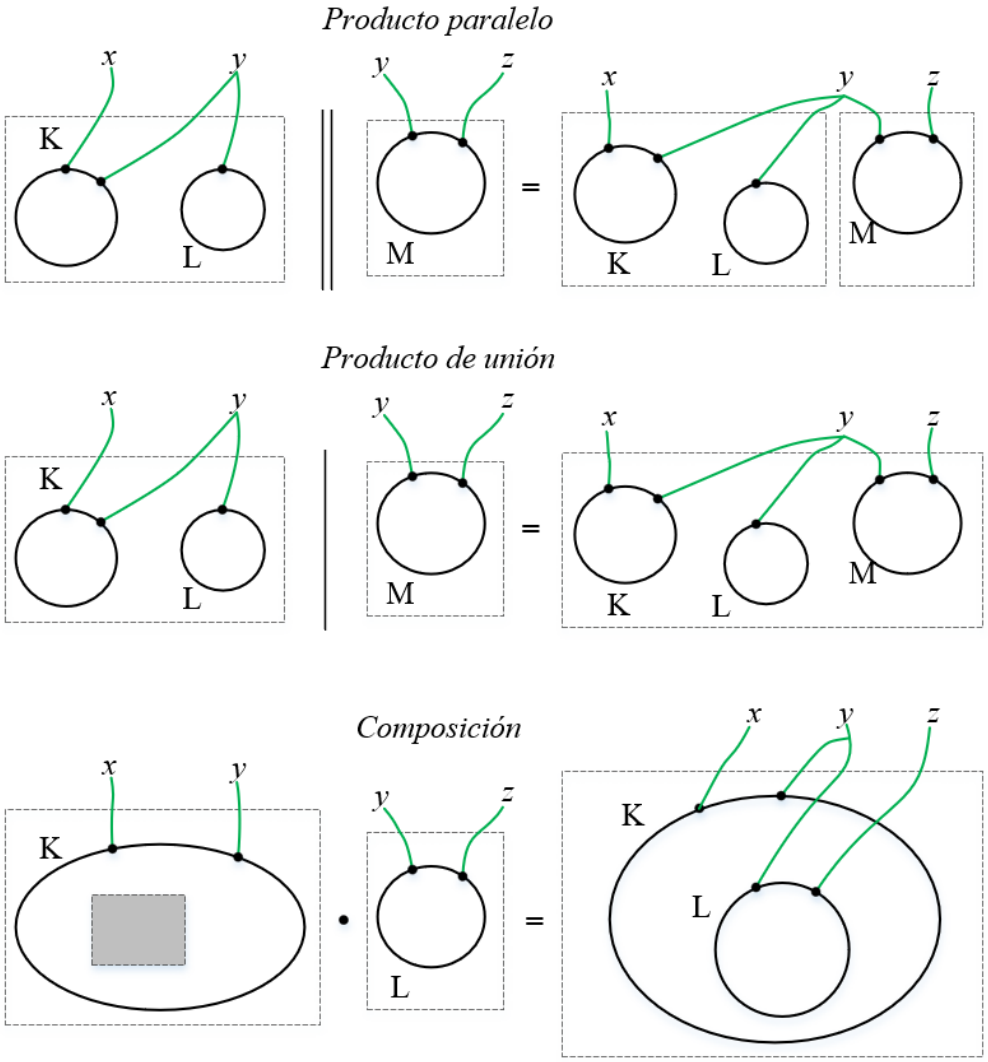

Figura 2-27.: Representación visual de las operaciones sobre dos bigrafos. Adaptada de: https://www.cl.cam.ac.uk/archive/rm135/Bigraphs-Lectures.pdf

2.6.1. Lenguajes de programación distribuidos

Los modelos actuales de programación, en general, solo proveen abstracciones para compartir datos bajo un modelo de consistencia homogéneo. Por lo tanto, puede que una aplicación distribuida provea una fuerte consistencia para una parte de los datos compartidos, y otro grado de consistencia para las otras partes. Mezclar modelos de consistencia no es soportado por los modelos de programación actuales, escribir aplicaciones con esta característica es extremadamente complicado [13]. Por ejemplo, encontramos CScript, un lenguaje de programación distribuida que fue diseñado para construir aplicaciones de consistencia mixta, que ofrece también soporte para replicación de datos nativa. Este consiste de objetos replicados disponibles y consistentes. CScript regula las interacciones entre esos objetos para evitar inconsistencias subsecuentes que surjan cuando se mezclen modelos de consistencia; y como resultado, se presenta que CScript es flexible y más eficiente respecto al uso de memoria [13].

Por otro lado, el auge de las plataformas de computación móvil ha dado origen a una nueva clase de aplicaciones: aplicaciones móviles que interactúan con aplicaciones P2P ejecutándose en teléfonos móviles cercanos [41]. Desarrollar tales aplicaciones es un reto porque el problema es inherente a la programación distribuida y concurrente, y también porque los problemas son inherentes a las redes móviles, tales como el hecho de que la conectividad de las redes móviles es frecuentemente intermitente, y se presenta la ausencia de una infraestructura centralizada para coordinar los móviles participantes de la red P2P [41]. Para afrontar esta problemática se presenta AmbientTalk: un lenguaje de programación distribuido diseñado específicamente para desarrollar aplicaciones móviles P2P. AmbientTalk pretende hacer más fácil el desarrollo de aplicaciones móviles que sean resilientes a las fallas de la red. Este lenguaje está basado en el modelo de actores [41].

Es válido afirmar que un lenguaje de programación puede proveer mayor soporte a la intercomunicación de procesos respecto a una librería. Ya que la mayoría de lenguajes de paso de mensajes limitan el soporte de la comunicación entre las partes de un solo programa [36]. Por eso encontramos el lenguaje de programación distribuido Lynx, que facilita el paso de mensajes de manera conveniente y segura frente a los tipos de datos, no solo dentro de aplicaciones, sino también entre aplicaciones y entre colecciones de servidores distribuidos [36]. También encontramos el lenguaje de programación distribuido SR, que es un lenguaje que contiene múltiples procesos que se ejecutan en paralelo. Este lenguaje contiene un sistema de software completo que controla una potencial colección de procesadores para ser programados como un conjunto de módulos integrados de software. Los mecanismos clave de este lenguaje son las sentencias para la gestión de recursos, de sus operaciones y el manejo de las entradas. Este segundo lenguaje soporta compilación separada, abstracción de tipos y enlaces de comunicación dinámicos [6].

2.6.2. Lenguajes intermedios

Dentro de la aplicabilidad de los lenguajes intermedios específicos podemos encontrar múltiples entornos donde estos son requeridos. Por ejemplo, encontramos el lenguaje LEAN [7]: un lenguaje intermedio experimental basado en la reescritura de grafos. Está basado en una alternativa para los Sistemas de Reescritura de Términos (TRS, en inglés), en el cual los términos son reemplazados por grafos. Así como un Sistema de Reescritura de Grafos (GRS, en inglés), que consiste en un conjunto de reglas de reescritura de grafos el cual especifica cómo un grafo puede ser reescrito. A parte de soportar programación funcional, LEAN también describe construcciones imperativas y permite la manipulación de grafos cíclicos. Los programas pueden presentarse como deterministas, así también como no deterministas. En particular, LEAN puede servir como un lenguaje intermedio entre lenguajes declarativos y arquitecturas de computación, ambos de forma secuencial y de forma paralela [7].

Otro ámbito para el uso de los lenguajes intermedios puede encontrarse en la robótica. Por ejemplo, con el lenguaje VML (Virtual Machine Language), que es un lenguaje de programación de nivel intermedio para manipuladores, los cuales permiten programación estructurada, concurrencia, tipos de dato específicos para robótica y un cierto grado de independencia. VML puede ser usado como un lenguaje interactivo de comandos tanto como un lenguaje objetivo para sistemas de software de robótica de alto nivel. Su definición involucra un número de decisiones las cuales presentan una visión abstracta del robot como uno de muchos componentes en una fábrica automatizada [8]. Y para otros ámbitos, como el de computación general, encontramos el lenguaje FLIC, que es un lenguaje intermedio de tipo funcional, que provee un lenguaje intermedio común entre diversas implementaciones de lenguajes funcionales, incluyendo también los paralelos [26].

3. Diseño del lenguaje de programación DiUNisio 2.0

Para poder diseñar el lenguaje de programación social-inspirado que se requiere para el modelo TLÖN (presentado en la Sección 2.2), se tuvieron en cuenta múltiples consideraciones, las cuales se listan a continuación:

- El sistema operará sobre una red ad hoc, lo que implica que debe tenerse en cuenta la necesidad de realizar computación concurrente, distribuida y móvil.
- El computador virtual será la representación del conjunto de todos los recursos disponibles (y otorgados para la red) de los nodos que lo conforman.
- Existe una metáfora social que fundamenta la filosofía de este modelo. Donde los agentes sartreanos son los engranajes o partes fundamentales que hacen funcionar este sistema.
- El lenguaje de programación debe poder acceder a cada una de las capas del modelo TLÖN, tener la capacidad de poder manipular y definir las interacciones entre cada uno de sus componentes.
- Al igual que un lenguaje de programación general, este lenguaje debía ser capaz de realizar computación.

Pero además de estas necesidades, también se consideraron los siguientes conceptos naturales y sociales:

- La definición de fronteras dadas por el universo y los países.
- La noción de temporalidad, la cual permite el registro de eventos, la existencia de una cronología y la evolución de los componentes que son afectados por ella.
- El concepto de leyes naturales, las cuales rigen todos los elementos inmersos en su naturaleza.

- El gobierno y la distribución de poderes mediante la jerarquía de instituciones y la potestad de decisión de sus grupos logísticos.
- Elementos como la justicia [31], el consenso [5] y la cooperación [20].
- Los agentes sociales tienen la capacidad de participar en esta sociedad a través de distintos roles, los cuales determinarán qué función cumplirá ese agente en un momento y espacio dado.
- La concepción de idioma, la cual permite la correcta comunicación entre dos entes.
- La cualidad de identificarse, de ser único y de ser distinguido de entre un grupo colectivo de similares [37].
- La capacidad de un agente artificial de evaluar su entorno para tomar decisiones y cambiar sus estados internos.
- La idea de conocimiento, como un medio que permite resolver problemas ya sea mediante habilidades o razonamiento.

A partir de las necesidades y conceptos presentados anteriormente, el diseño de este lenguaje de programación pudo ser modelado a través de las relaciones entre las entidades mostradas en la Figura **3-1**. Para diseñar estas entidades y sus relaciones del lenguaje de programación a proponer, se tomaron en cuenta pensamientos del filósofo alemán Schopenhauer, que dice que las comunidades de personas están ejecutando una obra de teatro donde hay una única puesta en escena, y además, dice que la vida es como estar en una obra de teatro, donde cada uno es el protagonista de su propia vida [35].

Tomando los conceptos presentados en el *cálculo de ambientes* y el *modelo de actores*, se pudieron modelar diferentes entidades espaciales y sociales que interactúan unas con otras de manera muy similar a las descritas por estos modelos de computación. Al igual que como se presentan en la obra de Russell y Norvig ([32]) soluciones a problemas mediante el uso de comunidades de agentes inteligentes, en este caso, los *agentes sartreanos* representan una comunidad de agentes inteligentes pero con comportamiento sociales.

Las entidades tipo **actor** presentan un comportamiento reactivo ante el paso de mensajes, los cuales son enviados y recibidos entre actores. Estos mensajes poseen un contenido el cual será el encargado de activar la respuesta indicada gracias al patrón con el que el mensaje esté estructurado. Lo último se inspiró en el cálculo join, en el cual un patrón de átomos (o

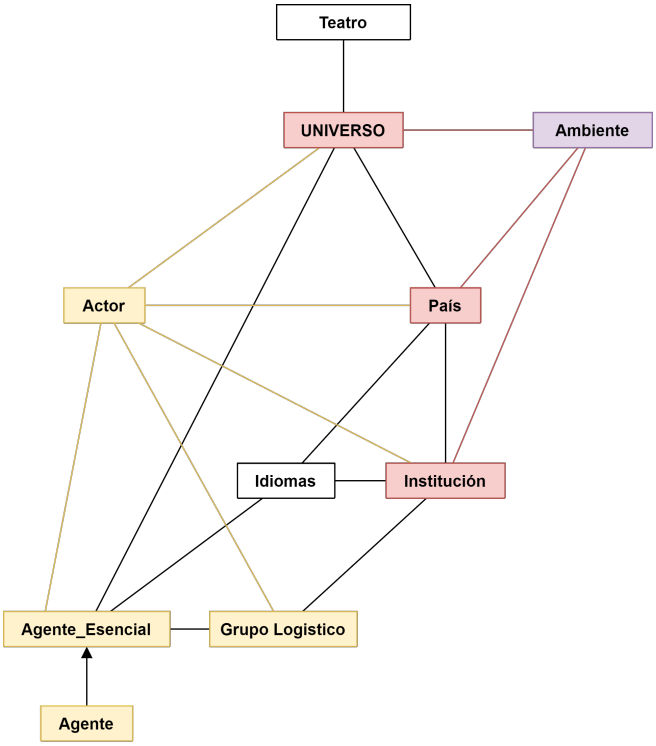

Figura 3-1.: Relaciones entre las entidades de la propuesta del modelo social. Creación propia.

molécula), es considerado para su reacción.

En este modelo, los actores también son capaces de desplazarse entre **ambientes**, los cuales están contenidos jerárquicamente unos en otros. Lo que permite la capacidad de computación móvil. Asimismo, un ambiente puede moverse con las operaciones descritas en la la Figura **2-22**. A continuación, se describen las entidades consideradas en la propuesta de este lenguaje de programación social-inspirado:

1. **Teatro:** En este caso, en términos teatrales, representa el lugar donde las **obras** son efectuadas. Contiene una **escenografía**, la cual en el modelo TLÖN, representa los recursos disponibles para la realización de estas obras. Cada obra (o aplicación) está compuesta por un **elenco**[1] de actores, los cuales serán los encargados de hacerla realidad. Mediante un **diálogo**[2] se definen las conversaciones entre los actores. Y que finalmente concluye con **mutis**[3], donde los recursos son liberados. Finalmente, se da la puesta en escena, en la cual las obras se hacen realidad (o función principal del programa, en términos de programación). En este modelo de computación social, el **Teatro** representa al más grande de los ambientes, el que contiene a todos ellos.

2. **Universo:** Este ambiente representa el **espacio-tiempo** y las **leyes de la naturaleza**, en el cual su contenido es afectado por ambas cosas. Desde él nacerían en el sistema las demás entidades, sería el encargado de propagar mensajes a todas las entidades que contiene, informándoles sobre el paso del tiempo (mediante una multidifusión). En este modelo, el **Universo** representa un ambiente que puede contener otros ambientes (ya sean **Instituciones** o **Países**).

3. **País:** Representa un territorio delimitado, en él se crearían las entidades institucionales, poseería diferentes atributos que conformen su **identidad**; un medio de registro de eventos históricos; una **cultura**, la cual podría definir, por ejemplo, el idioma oficial de su población; y, por último, una interfaz que permite la recepción de mensajes para su **redirección**. Dentro de este modelo, el **País** representa un ambiente que puede contener otros ambientes tipo **Institución**.

4. **Instituciones:** Son organizaciones encargadas de ofrecer servicios, están contenidas jerárquicamente unas en otras, siendo la de arriba dominante sobre las inferiores. Pueden representar diferentes modelos de gobierno, de empresas, de entidades nacionales,

[1] Conjunto de actores que constituyen una compañía teatral o que actúan en una obra. RAE.
[2] Obra literaria, en prosa o en verso, en que se finge una plática o controversia entre dos o más personajes. RAE.
[3] En el teatro, acción de salir de la escena un actor. RAE.

entre otros. Presentan su propia **identidad**, una **historia**, la definición de las actividades de sus **miembros**, **canales** de comunicación interna, el horario de **disponibilidad** en el que ofrecerían un repertorio de **servicios**. También presentan una interfaz reactiva ante mensajes entrantes como su medio de **percepción** del entorno.

5. **Grupos logísticos:** Son comunidades de agentes, los cuales se encargan en la toma de decisiones de una institución, mediante mecanismos de decisión como la **votación** y el **consenso**. Se representan a sí mismas con su propia **identidad**, un conjunto de **estados** internos que determinan sus decisiones, una serie de **participantes**, los cuales pueden ser miembros de la institución contenedora o instituciones subordinadas. En este grupo logístico también se implementa la noción de ejecución no determinística del operador (+) del cálculo π, donde dadas una serie de procesos, el grupo logístico tomaría la **decisión** de ejercer una acción de entre las posibilidades.

6. **Agente esencial:** Representa el agente estándar, el comportamiento común entre todos los agentes del sistema. Contienen su propia **identidad**; un recuento de su propia **historia**; un conjunto de variables que determinan su **estado** actual; un **inventario**, en el cual almacenarían elementos que les permitan cumplir con sus deseos u obligaciones dentro de la sociedad por la que se mueven. También un medio de reacción ante la percepción del ambiente en el que se encuentra a través de los **mensajes** que recibe. También con un conocimiento en común con todos los agentes del sistema.

7. **Otros agentes:** Son la caracterización especial de diferentes roles específicos que la sociedad necesita. Serían los que contendrían conocimiento extra y la modificación y/o extensión de todos los aspectos de un agente esencial.

8. **Idiomas:** Por último, los idiomas, serían en este caso, una serie de **patrones** que definen cómo un actor reacciona ante un mensaje, lo que metafóricamente representaría un idioma dentro de este modelo. Y así, los actores que hablen el mismo idioma podrán entenderse y reaccionar de manera adecuada. La realización de computación con este modelo social fue modelada con la gramática del lenguaje de programación DiUNisio 2.0 (mostrada en el Anexo A), donde DiUNisio es un lenguaje de programación que originalmente estaba definido bajo el paradigma imperativo, el cual, en este trabajo fue modificado para introducir en él la sintaxis requerida para estructurar las entidades del modelo social presentado anteriormente junto con otras ideas tomadas del cálculo pi, con las cuales de pueden describir procesos concurrentes (o paralelos).

Para escribir código en este lenguaje de programación, el programador sería considerado como el *director de la obra de teatro*. Este director podría empezar a determinar quiénes podrían

ser los actores principales, por ejemplo, los que permiten la comunicación; pero también podría determinar quiénes son los actores secundarios, por ejemplo, los que mantienen las comunicaciones activas. Igualmente este director tiene que considerar cuáles son los elementos de la escenografía disponibles para que los actores logren desenvolverse correctamente en sus papeles. Se requiere igualmente un guión, que le diga a los actores cuándo entrar, cuándo actuar y cuándo salir de escena. En otras palabras, el *director de la obra*, metafóricamente hablando, lo que debe hacer es escribir una obra de teatro para crear aplicaciones. La lógica de programación sería la siguiente:

1. Dentro de la definición del Teatro:
   - Primero se establecería la escenografía, la cual debería definir la disposición de los nodos de la red ad hoc disponibles para la ejecución de las obras, por ejemplo, mediante clústeres. Esto representa analógicamente cuál es la inicialización de las variables y entorno de cómputo requerido y disponible para la ejecución de las obras en el Teatro.
   - Luego se modelan las obras, donde se tienen que definir sus actores (elenco) y dónde estos estarían contenidos (universos y países). Estas obras simulan la definición de aplicaciones y métodos que están definidos que ofrecerá el sistema para ejecutarse en cualquier momento.
   - A continuación, se describiría el diálogo detonador del sistema, el cual podría ser un mensaje inicial dirigido a un actor del sistema. Es el diálogo una metáfora a decir cómo será el algoritmo principal del sistema, representa la serie de comunicaciones o interacciones entre las entidades de computación definidas.
   - Opcionalmente, definir qué debería suceder al final de la obra o mutis. Representa cómo se debe "liberar memoriaz/o terminar procesos en ejecución.
   - Finalmente definir la puesta en escena de las obras sobre la escenografía, en otras palabras, definir la asignación de recursos e interfaces de interacción entre los componentes del sistema para la ejecución de sus aplicaciones. La puesta en escena representa una *función principal*, la cual es el punto de entrada al sistema programado.
2. Simultáneamente, deberían definirse las entidades: Universo, País con sus Instituciones y Grupos Logísticos, el Agente Esencial y los Agentes específicos.

Otras consideraciones y sugerencias:

- Para una vista técnica y funcional de este lenguaje se recomienda la lectura de los manuales de DiUNisio 2.0 (ver el Apéndice A y el Apéndice B).

- Para consultar ejemplos escritos con este lenguaje se recomienda la lectura del manual de usuario de DiUNisio 2.0 dentro del Apéndice A y también en el Capítulo 7).
- La seguridad [14] de acceso se podría modelar con el uso de llaves de entrada y de salida de ambientes.
- Se pueden implementar modelos de economía [25], donde los agentes trabajan buscando el empleo ideal para suplir sus necesidades, o buscando un comportamiento cooperativo basado en teoría de juegos.
- También se podría integrar el uso de las emociones [19], las cuales determinen el comportamiento irracional de los agentes.
- Los agentes podrían tener sus propios objetivos [43], los cuales los motivarían a seguir una serie de acciones para cumplirlos.
- Se podrían representar diferentes profesiones, debido a la posibilidad de generar diferentes tipos de agentes con conocimiento específico.
- Se podrían representar múltiples universos, los cuales permitan el flujo del espacio tiempo a diferentes tasas. Lo que permitiría realizar simulaciones o evaluación de parámetros.

4. Diseño del lenguaje intermedio LIST

Dentro de las fases de un compilador encontramos la fase de generación de código intermedio, en la cual se produce una salida mediante la transformación de la entrada que recibe. De esta salida se requiere que sea un código más adecuado para generar código objeto o código máquina para una máquina objetivo en las fases posteriores. El diseñar un lenguaje intermedio difiere de diseñar un lenguaje de máquina principalmente por lo siguiente: cada instrucción representa exactamente una operación fundamental, y el número de registros del procesador disponibles puede ser muy grande, a veces sin límites.

A la hora de escoger un lenguaje intermedio podemos encontrar gran variedad de ellos, ya sean orientados a objetos, lenguajes de alto nivel, o más reducidos como el código de tres direcciones. Dependiendo de las propiedades que se busquen en él, podremos encontrar ventajas y desventajas en cada uno de ellos. Es por esto que para el diseño del lenguaje intermedio para el compilador del computador virtual del sistema TLÖN se estudiaron diferentes lenguajes intermedios, como:

1. Código de tres direcciones (3AC) [3]: es un código intermedio usado para optimizar compiladores. Cada instrucción contiene a lo mucho 3 operandos. Su nombre se deriva del uso de estos 3 operandos (aún así cuando pueden aparecer instrucciones con menos).

2. Parrot intermediate representation (PIR)[1]: es uno de los dos lenguajes ensambladores para la máquina virtual Parrot. PIR provee un conjunto de abstacciones que permiten al programador ignorar ciertas redundancias en el *bytecode* de Parrot y permite escribir rápidamente código que se adhiere a las complejidades de Parrot, tal como las convenciones del llamado a métodos.

3. LLVM[2] IR: es el núcleo de LLVM, es similar a assembly. Esta representación intermedia usa un conjunto de instrucciones RISC fuertemente tipado. Soporta 3 modos equivalentes: el formato de ensamblado legible por humanos, un formato de memoria adecuado para interfaces, y un formato denso de *bitcode* para serialización.

[1]Consultar en: http://www.parrot.org/
[2]Consultar en: https://llvm.org/docs/index.html

4. CIL [17]: el Common Intermediate Language es un lenguaje intermedio orientado a objetos y basado en pila. Desarrollado por Microsoft para ejecución en entornos compatibles con la especificación Common Language Infrastructure (CLI).

Un ejemplo de código escrito en 3AC se presenta a continuación:

```
v1 := a + a
v2 := 5 / a
v3 := v2 - c
x := v3
```

En el que evidenciamos un código simple pero restrictivo a la hora de escribir algo más complejo.

Un ejemplo de código escrito en PIR se presenta a continuación:

```
.local int x
.local num y
x = 2
y = 5.5
.local num z
z = x + y
```

Con el que se aprecia que se declaran las variables previamente, pero sigue siendo muy similar al 3AC.

Un ejemplo de código escrito en LLVM IR se presenta a continuación:

```
@.str = internal constant [14 x i8] c"hello, world\0A\00"

declare i32 @printf(i8*, ...)

define i32 @main(i32 %argc, i8** %argv) nounwind {
entry:
    %tmp1 = getelementptr [14 x i8], [14 x i8]* @.str, i32 0, i32 0
    %tmp2 = call i32 (i8*, ...) @printf( i8* %tmp1 ) nounwind
    ret i32 0
}
```

En este ejemplo podemos ver cómo se declaran todos los tipos de variables y métodos. Igualmente se evidencia cierta complejidad para su comprensión a simple vista.

Un ejemplo de código escrito en CIL se presenta a continuación:

```
.class private auto ansi beforefieldinit HolaMundo
       extends [mscorlib]System.Object
{
 .method private hidebysig static void  Main(string[] args)
 cil managed
  {
    .entrypoint
    .maxstack  8
    IL_0000:  nop
    IL_0001:  ldstr "Hola Mundo!"
    IL_0006:  call  void [mscorlib]System.Console::WriteLine(string)
    IL_000b:  nop
    IL_000c:  ret
  }

  .method public hidebysig specialname rtspecialname
          instance void  .ctor() cil managed
  {
    .maxstack  8
    IL_0000:  ldarg.0
    IL_0001:  call  instance void [mscorlib]System.Object::.ctor()
    IL_0006:  ret
  }
}
```

En este ejemplo podemos ver un código más sencillo de analizar (ya que es orientado a objetos), a pesar de tener gran cantidad de elementos como etiquetas y ciertas palabras clave propias de este lenguaje y su compilador.

Respecto a los ejemplos presentados anteriormente, el diseño del lenguaje intermedio para este proyecto se inspiró en CIL (Common Intermediate Language de Microsoft) [17], por plena preferencia a la hora de entender el código escrito. Aún así se evidencia que CIL muestra un nivel de expresividad mayor respecto a los lenguajes 3AC y PIR; pero menor confusión a la hora de compararlo con LLVM IR. CIL es un lenguaje intermedio que trabaja

con operaciones que van cargando y descargando datos de una pila. Esto permite que el compilador pueda ir ejecutando instrucción por instrucción y pueda hacer uso del valor que está en la cima de esta. Esto es representable con una máquina de pila abstracta. También CIL utiliza una representación basada en el paradigma orientado a objetos, donde existen clases, métodos y encapsulamiento.

En este caso, el lenguaje intermedio propuesto para este trabajo es LIST (Lenguaje Intermedio para el Sistema TLÖN), una versión simplificada y adaptada de CIL, el cual fue escogido por su expresividad y su flexibilidad como primera fase de generación de código intermedio. Con el que se renombran algunos de sus elementos y se cambia gran parte de su sintaxis. Con el objetivo de proponer una versión inicial de código intermedio que a futuro sirva para generar código máquina para los nodos que son parte fundamental del computador virtual de TLÖN. Este lenguaje presenta una estructura muy simple:

1. Para importar librerías se usa la palabra clave *namespace*.
2. Se pueden definir clases, las cuales contienen una serie de atributos y métodos. Estas clases pueden extender de otras, en donde se presenta el concepto de *herencia*.
3. Los atributos de una clase se definen con su modificador de acceso, su nombre y su tipo de dato, al igual que su posible inicialización.
4. Los métodos de una clase se definen de manera similar a los atributos, solo que en este caso definen el tipo de retorno, un conjunto de parámetros, y dentro de ellos, un conjunto de instrucciones.
5. Las instrucciones atómicas de este lenguaje se pueden consultar en el Apéndice D, donde se muestra la instrucción junto con sus posibles parámetros. Existen operaciones simples (sin parámetro), unarias (un parámetro) y binarias (dos parámetros).
6. Existen diferentes tipos de dato: entero, real, cadena de caracteres, nulo. Junto con estructuras como los arreglos.

Esta nueva gramática (que se encuentra dentro del Apéndice C) representa los elementos del modelo social escritos en DiUNisio 2.0 como clases que extienden de una clase que debería ser implementada en futuras versiones. Las cuales en un trabajo futuro deberían tener ya implementados los métodos y atributos requeridos para cumplir con esta propuesta. Esto se debe a que dentro del alcance de este proyecto no se consideraron pruebas de ejecución de los algoritmos generados como código intermedio producto del proceso de traducción.

Para esta versión de lenguaje intermedio, LIST también hace uso de una serie de anotaciones (palabras clave precedidas de una arroba) que simbolizan qué métodos de la clase representan comportamientos reactivos ante mensajes (con la anotación @receptor), o, por ejemplo, si un atributo de una institución es usado como valor de referencia para ser encontrada dentro de una jerarquía de instituciones (con @reference).

También se hace uso de llamados hipotéticos a la capacidad de realizar reflexión computacional (mediante [sovoraib]System.Reflection), donde un programa puede observar y/o modificar su estructura de alto nivel. Los cuales, en este caso, permiten llamar funciones que no fueron definidas en DiUNisio 2.0 ya que se supone que son parte de librerías o microservicios del sistema.

Nota: Para una vista técnica de este lenguaje se recomienda la lectura del manual técnico de LIST (ver Apéndice C). Para la consulta de ejemplos ir al Capítulo 7, donde se muestran códigos intermedios escritos en LIST producto del proceso de traducción de la herramienta desarrollada.

5. Microinstrucciones y microservicios

Para aportar al modelamiento del computador virtual del sistema TLÖN, se consideraron una serie de microinstrucciones y microservicios que permitirían aprovechar las capacidades de su computador virtual, que también pueden ofrecerle al programador una gran variedad de herramientas con el fin de escribir programas y desarrollar aplicaciones. Donde las microinstrucciones determinan las operaciones de pila, que en este caso deberían ser aceptadas e implementadas en el lenguaje intermedio LIST, y los microservicios, los cuales serían llamados al sistema, que ofrecen un sinfín de facilidades: de gestión de recursos virtuales, de ejecución paralela, para transferencia de datos, entre otros.

Las **microinstrucciones** se definen como:

- Instrucciones **atómicas**.
- Acciones mínimas realizables dentro del entorno de computación virtual del sistema TLÖN.
- Conjunto de **instrucciones** que se ejecutan en el computador virtual del sistema TLÖN.
- Son **locales** (nodo ad hoc) o **distribuidas** (red ad hoc).

Los **microservicios** se proponen con las siguientes categorías:

1. Operaciones sobre **colecciones de datos** (o la posibilidad de usar estructuras de datos).
2. Al igual que el anterior, pero estructuras de datos diseñadas para operaciones **concurrentes**.
3. Operaciones de **diagnóstico** sobre la ejecución de aplicaciones y programas.
4. Operaciones sobre los **dispositivos de entrada** y **de salida** del sistema.
5. Operaciones sobre la **red** y las comunicaciones de los nodos de la red ad hoc del sistema.

6. Operaciones de **serialización** y **deserialización** de datos (indispensable para la representación de movilidad de los actores y ambientes del modelo social).

7. Servicios de **control de acceso**, **permisos** y **criptografía**.

8. Operaciones de **codificación** y **decodificación** de texto.

9. Operaciones y herramientas para computación **multihilo**, como los semáforos.

10. Gestión de **tareas en paralelo**.

11. Operaciones de **reflexión** para objetos de programación.

12. Una serie de servicios para operación sobre **tipos de dato**, el uso de la **consola**, generación de números **aleatorios**, entre otros.

13. Servicios sobre el **sistema de archivos virtual** del computador de TLÖN.

14. Operaciones sobre **Actores** y **Ambientes** del modelo social del sistema.

15. Definición de las operaciones y clases del modelo social para la creación de **Universos**, **Países**, **Instituciones**, **Idiomas**, **Grupos Logísticos** y **Agentes**.

Nota: En este caso se hizo la revisión, selección y adaptación de algunas de las operaciones que define CIL, las cuales son Turing-completas y representan gran variedad de operaciones sobre la máquina de pila abstracta del compilador. Y para la selección de microservicios, se investigaron las funcionalidades que ofrece la librería principal de Microsoft (mscorlib), agregando nuevos microservicios indispensables para el modelo social desarrollado en este documento. Para revisar la propuesta de microinstrucciones y microservicios, proceder a su lectura en el Anexo C y D.

5.1. Modelamiento mediante bigrafos

A continuación se representan mediante bigrafos, un pequeño grupo de microinstrucciones y microservicios; los cuales permiten visualizar cómo interactúan los elementos virtuales del compilador y, para los microservicios, la interacción de los nodos que se presentan contenidos en un clúster, cuyos recursos son representados como elementos virtuales que están conectados entre nodos a través del orquestador del computador virtual de TLÖN. Son importantes porque permiten modelar problemas y soluciones más complejos de una manera gráfica.

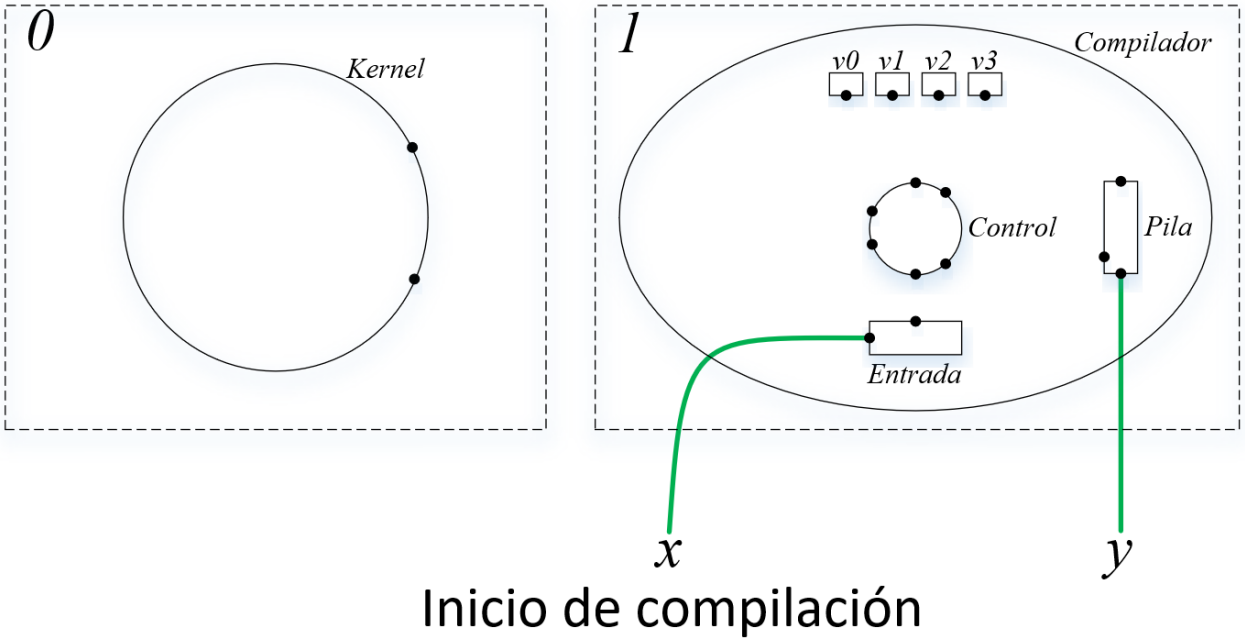

Figura 5-1.: Modelamiento del compilador en el estado inicial.

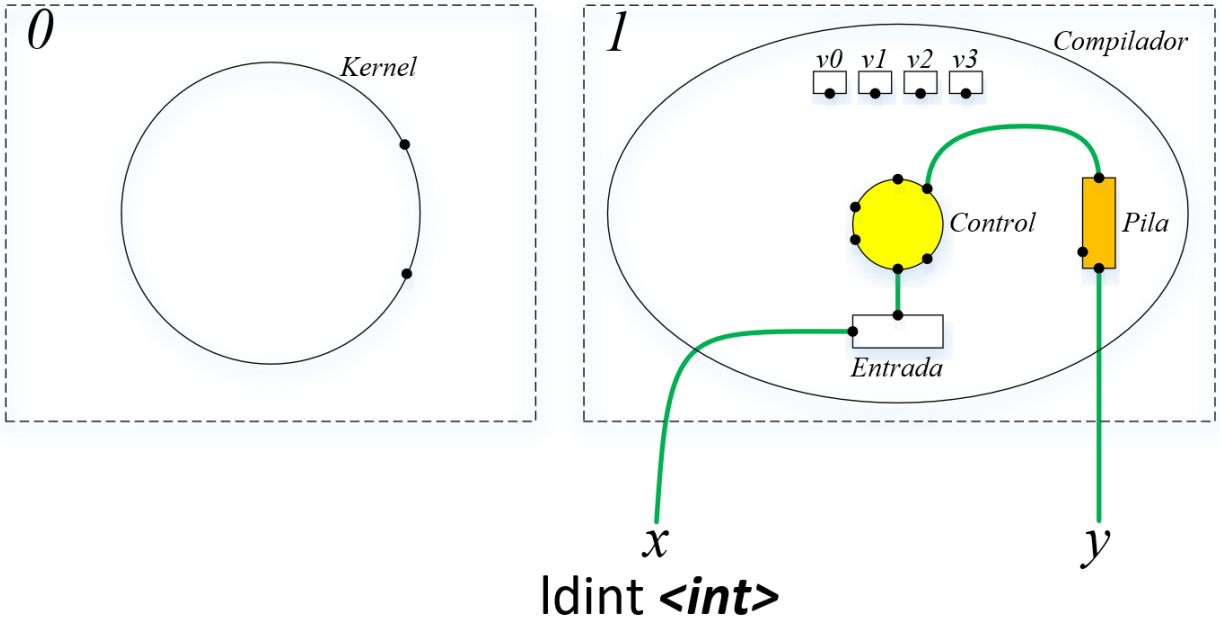

Figura 5-2.: Modelamiento del compilador cuando lee la operación *ldint* junto con su parámetro. En este caso, el control se activa leyendo el parámetro de la entrada y enviándolo a la cima de la pila.

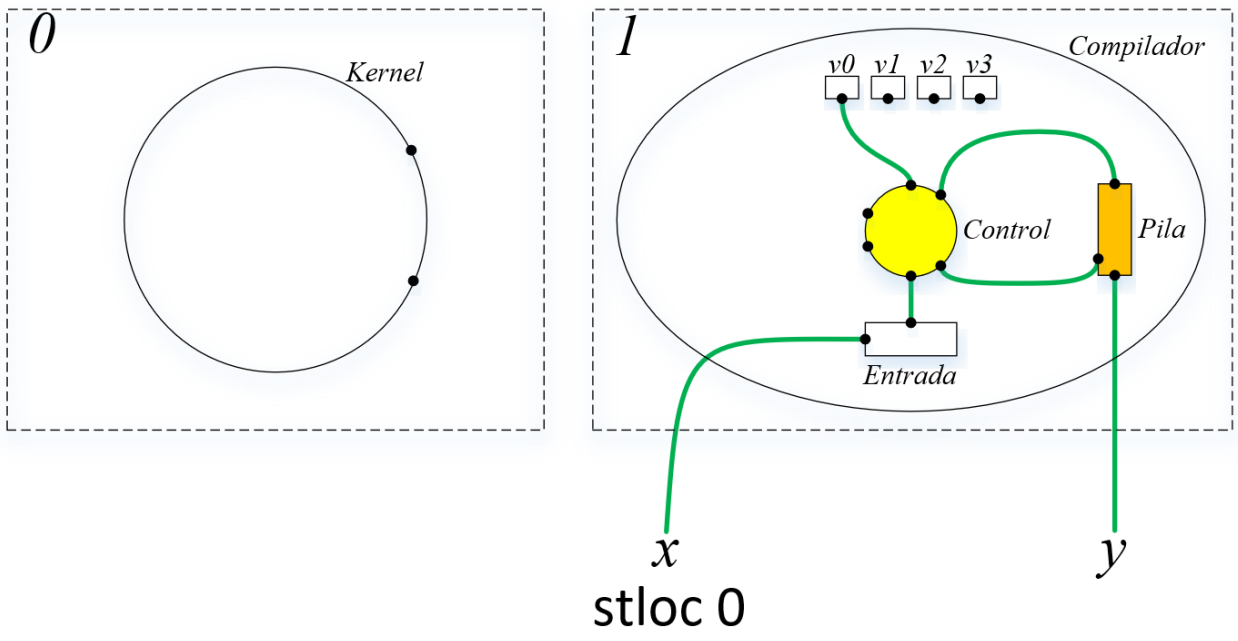

Figura 5-3.: Modelamiento del compilador cuando lee la operación *stloc* junto con su parámetro. En este caso, el control se activa leyendo el parámetro de la entrada, consultando la variable indicada en el parámetro y luego enviando su valor a la cima de la pila.

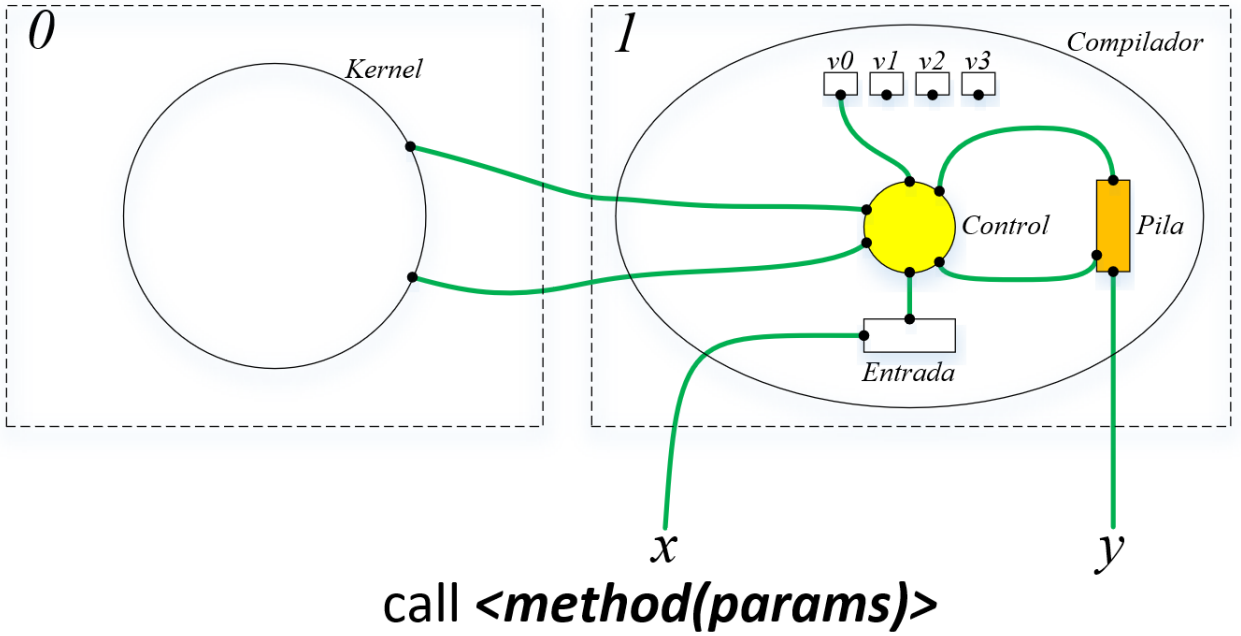

Figura 5-4.: Modelamiento del compilador cuando lee la operación *call* que contiene un método junto con sus parámetros. En este caso, el control se activa leyendo la referencia de la función junto con sus parámetros, consultando posiblemente la variable local indicada en alguno de sus parámetros. A continuación hace un llamado al kernel del sistema, el cual deberá retornar un dato que será apilado encima de la pila.

El primer modelo de bigrafo que se presentará es el que representa las microinstrucciones, y se logra a través de una representación como una máquina de pila; donde se tiene una entrada, una pila, un control y un conjunto dado de registros o variables locales.
El segundo modelo de bigrafo es el que representa los microservicios, y se logra a través de una representación de virtualidad, donde dentro del computador virtual se contienen sus recursos virtuales otorgados por múltiples clústeres que contienen múltiples nodos.

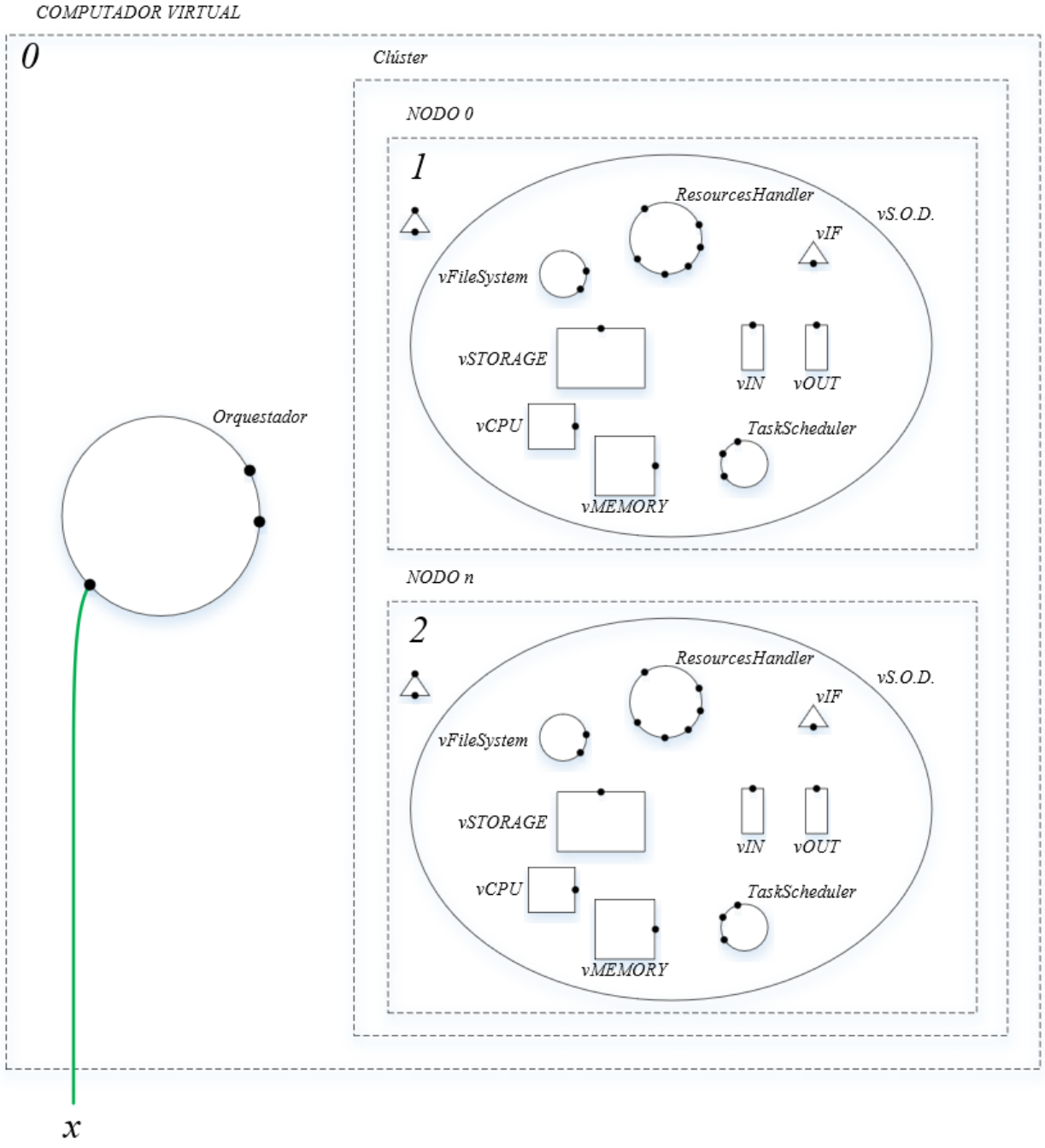

Figura 5-5.: Modelamiento de la orquestación en el estado inicial.

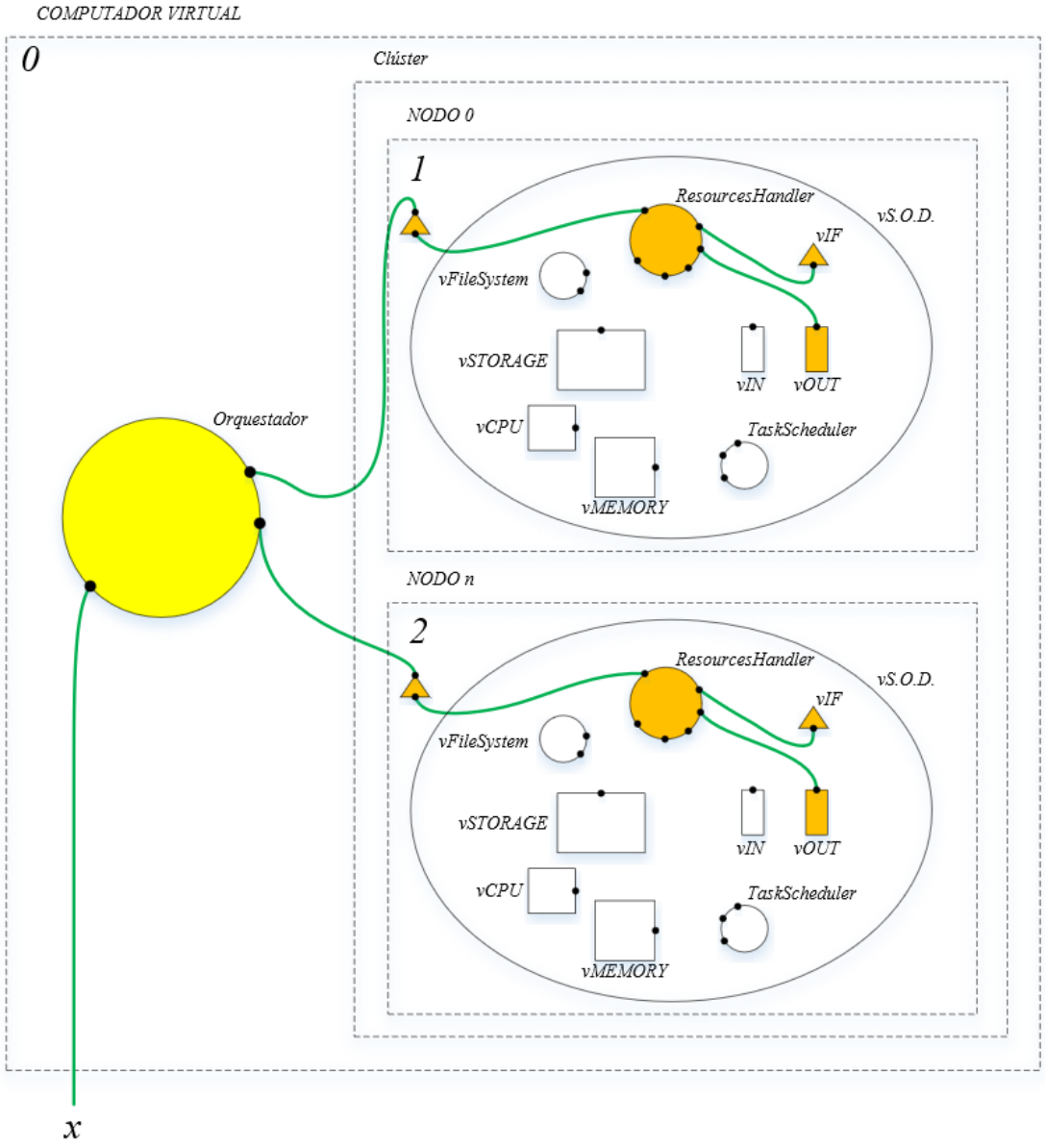

Figura 5-6.: Modelamiento de la orquestación en el momento en que pretende ejecutar una impresión en pantalla, lo cual, según los parámetros indicados, enviará al o a los nodos lo que les corresponde mostrar en sus respectivas consolas.

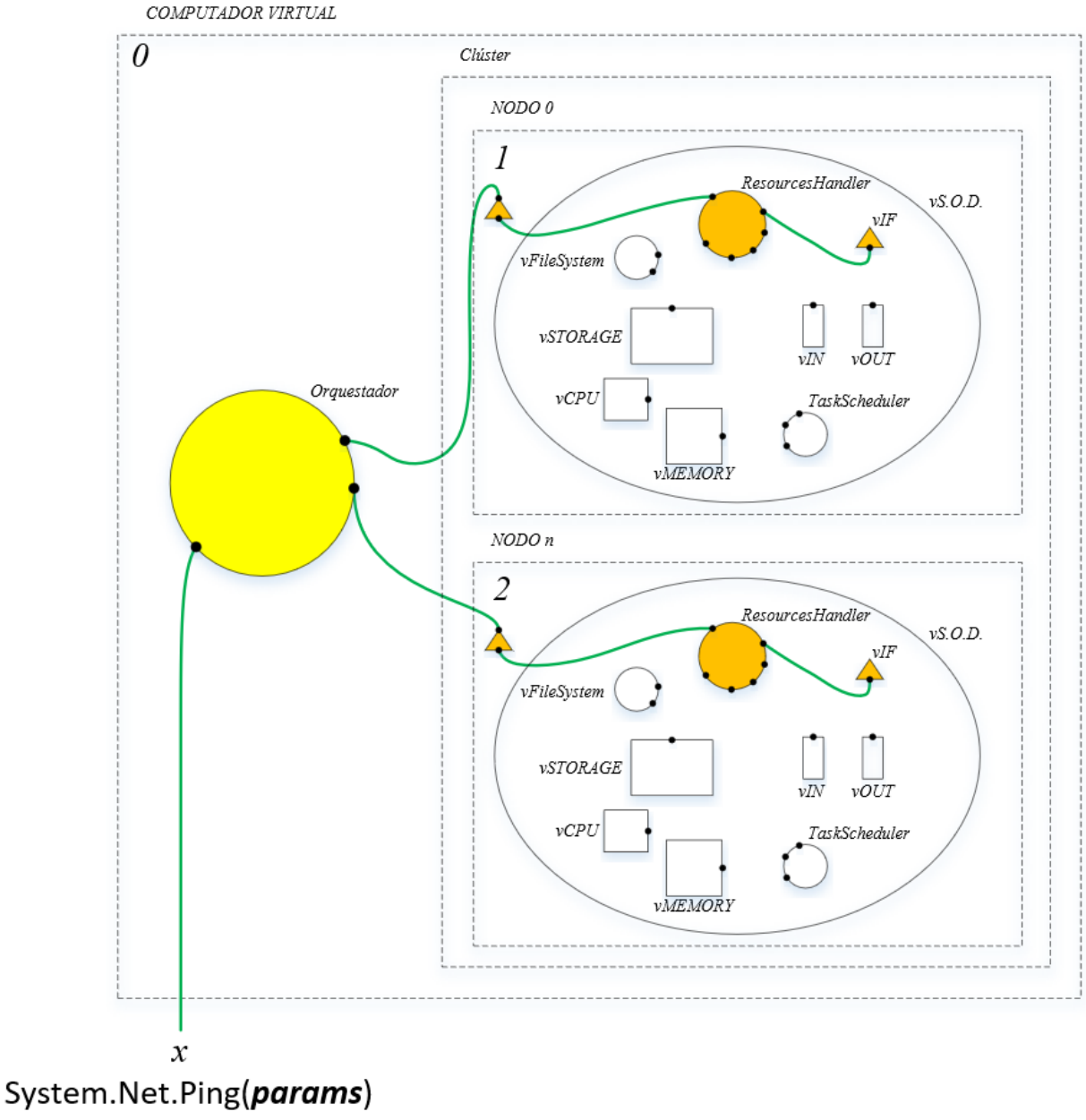

Figura 5-7.: Modelamiento de la orquestación en el momento en que requiere realizar una prueba de conexión entre dos nodos del computador virtual.

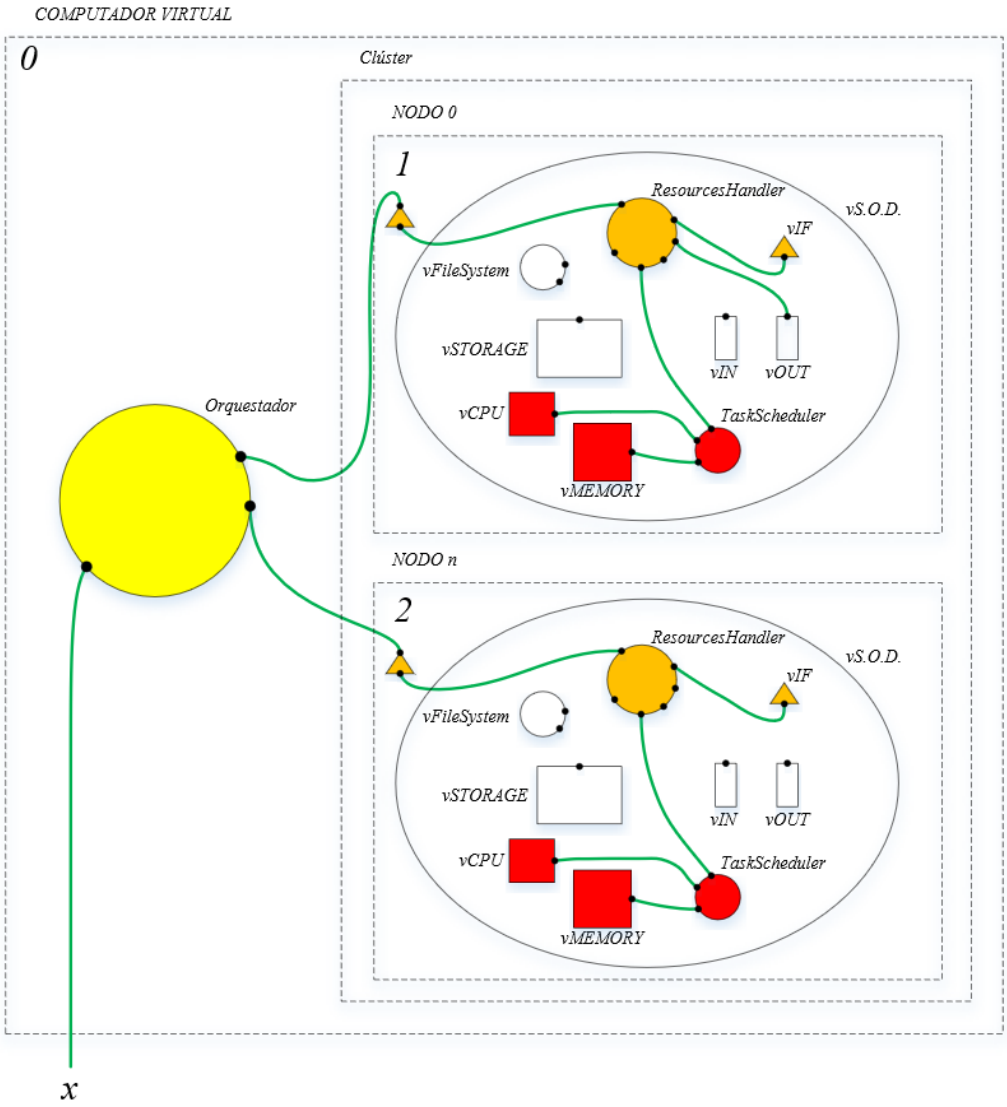

Figura 5-8.: Modelamiento de la orquestación en el momento en el que se ordena la ejecución paralela de un proceso a través de múltiples nodos del computador virtual.

6. Construcción del Traductor

Para la construcción e implementación del traductor de DiUNisio 2.0 se utilizó la herramienta ANTLR, la cual permite la generación de analizadores léxicos, sintácticos y semánticos con gran facilidad; los cuales fueron indispensables para la generación de código intermedio escrito en LIST. En la Figura **6-1** se pueden apreciar las representaciones intermedias que genera ANTLR, en las primeras tres fases del proceso de compilación, a lo que se le suma la generación de código intermedio dada por el prototipo de traductor desarrollado.

El proyecto está basado en la implementación de la interface *ParseTreeVisitor* (definida en la librería de ANTLR), que es la que estandariza las operaciones sobre los nodos al recorrer el árbol de sintaxis generado por el analizador sintáctico, y permite retornar valores desde las hojas, o nodos terminales, hacia arriba.

Para el almacenamiento de las variables, parámetros de funciones y atributos de clase, se hizo uso de una tabla de símbolos; esta permite almacenar los valores generados por las diferentes operaciones semánticas. La tabla de símbolos fue implementada con un diccionario (Hashmap), en la que se guardaban unidades atómicas de tipo *Symbol*, usándose el nombre del símbolo como llave y su valor era el símbolo en sí mismo.

La clase *Symbol* definía los atributos de nombre, tipo de dato, valor, su accesibilidad (pública o privada), número de argumento o de atributo de clase. La cual, de manera flexible, podía ser utilizada para almacenar cualquier tipo de símbolo del lenguaje, como variables, atributos de clase, argumentos de métodos, variables locales y resultados.

El proceso de traducción se basó en recorrer el árbol de sintaxis empezando con la regla sintáctica *estructura*, la cual contenía en ella la definición de *Universo*, de los *Idiomas*, de las *Constituciones* (la definición de los países), del *Agente Esencial*, de *Agentes* específicos y la definición del *Teatro*. Donde dentro de cada una de las producciones de las reglas sintácticas de estos elementos fueron recorridas y visitadas.

Al igual que DiUNisio, en su versión inicial, esta gramática realiza operaciones esenciales en la programación imperativa, donde se tiene la definición de secuencia, selección e iteración.

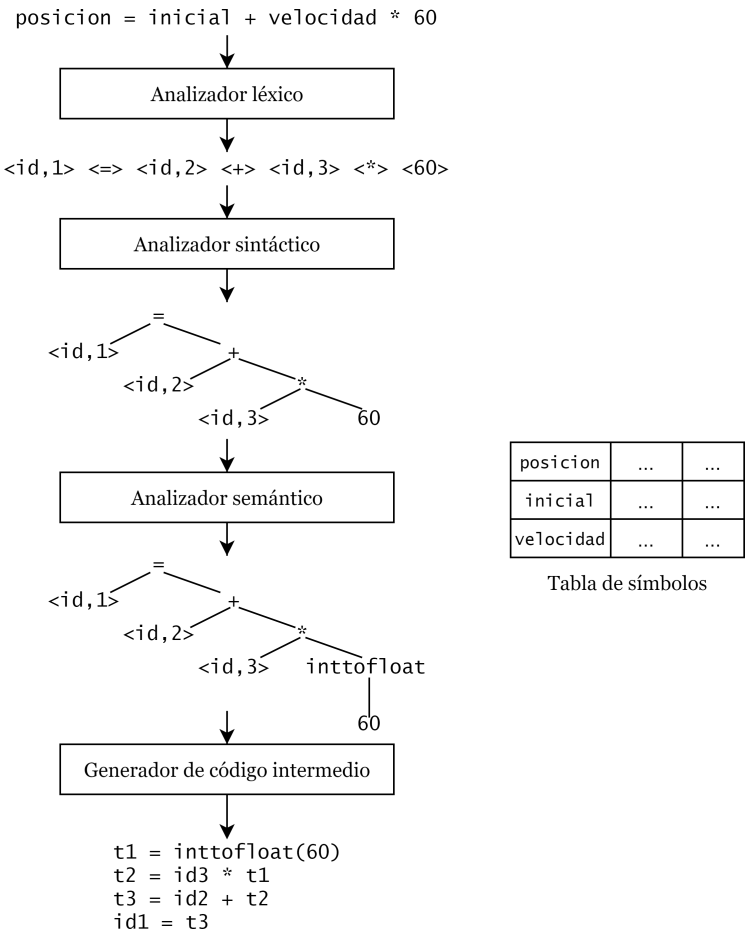

Figura 6-1.: Representaciones intermedias generadas por cada fase del proceso de compilación hasta la fase de generación de código intermedio. Adaptada de [42].

Junto con una serie de operadores lógicos, matemáticos y semánticos, que determinan el poder de computación de este lenguaje. En este caso, el traductor a medida que va realizando operaciones, las va almacenando de forma inmediata en nuevas variables locales, lo que puede generar redundancia a la hora de asignar una operación a una variable. Ya que, por ejemplo, si se quiere realizar la asignación $def\ a : entero = 1 + 2$, primero el resultado de $1 + 2$ será almacenado en una nueva variable, se cargará nuevamente esa variable y a continuación sí se asignará a la variable a.

Para la gestión de ámbitos y las variables que se definen en cada uno de ellos, se hizo uso de una estructura jerárquica, donde cada ambiente podría o no tener un ambiente hijo, partiendo de un ambiente global como referencia. Lo que permitía que cuando se accediera a secciones de código donde se crea un nuevo ambiente, un ambiente hijo era creado sobre el ambiente actual (el ambiente padre). Y al momento de salir de este ambiente, se reemplazaba el apuntador de ambiente actual desde el ambiente hijo, hasta su ambiente padre. Esto permitió llevar una correcta definición y uso de variables independientemente de la cantidad de ambientes anidados que existieran. Asimismo, con la definición de parámetros de una función o método, que se agregaban en la definicón del ambiente actual para establecer con prioridad la referencia a estos, en vez de variables externas a la función que tuvieran el mismo nombre.

El traductor generaba clases a partir de los elementos del modelo social. Se definen sus atributos y métodos partiendo de elementos como la identidad, la instanciación y los mecanismos de respuesta ante los mensajes de otros actores. Dependiendo de la entidad el modelo social a traducir, se transformaban esos elementos en sentencias estandarizadas basadas en un paradigma orientado a objetos, pero aplicado a los lenguajes intermedios. Igualmente, se hizo uso de diferentes anotaciones (palabras clave precedidas de arroba) los cuales indican al compilador qué elementos representan los mecanismos de respuesta y cuáles representan por ejemplo, métodos de instanciación o asignación de roles.

Debido a que en este trabajo solamente se está modelando lo que sería un lenguaje de programación para la computación social-inspirada, la implementación que haga realidad la computación con estas entidades aún es requerida, ya que dentro del alcance de este proyecto no se designaron objetivos de ejecución de pruebas funcionales del código intermedio generado (esto corresponde a trabajo futuro). Estas entidades se esbozan en el Apéndice E junto con una posible mínima colección de métodos que requieran para su correcta interacción (dada por el modelo propuesto). Por ejemplo, en esta aproximación de generación de código intermedio existen llamados hipotéticos a microservicios que debería ofrecer este sistema (TLÖN). Lo cual se simboliza con llamados a $System.Reflection.call(...)$ o

$System.Reflection.resolve(...)$, de los cuales, el primero recibe un arreglo de atributos y el nombre del método del objeto cargado en la pila sobre el cual se intentará ejecutar el llamado; y el segundo, encargado de obtener el campo de atributo de un objeto dado por el nombre que fue entregado como argumento de esta función.

Por lo anterior, es necesario un paso siguiente a la generación de código intermedio producido por este traductor, bien sea para ayudar a eliminar redundancias, ya que, en el traductor desarrollado, cada operación almacena un valor en una variable local, la cual puede ser inmediatamente usada para que su valor sea utilizado. La versión siguiente debería ser capaz de implementar (o recibir de una librería) todas las clases del modelo social propuesto y combinarlas con el código generado por DiUNisio 2.0.

7. Pruebas de generación de código intermedio

Para la presentación de la funcionalidad de la herramienta de Traducción desarrollada, se presenta a continuación un segmento obtenido de los resultados de traducción generados a partir de un programa escrito en DiUNisio 2.0.

Un ejemplo de traducción se realizó a través de un programa sencillo, el cual contiene las mínimas entidades necesarias para poder realizar el proceso de mostrar un mensaje en consola. En este caso se quiere poder mostrar el mensaje *HolaMundo* en la consola de un nodo de un clúster. Para ello, el código se estructuró de la siguiente manera:

```
UNIVERSO: {
    HISTORIA:{...};
    ESPACIO_TIEMPO: {...}
    INSTANCIADOR: {...}
    LEYES: {...}
}

CONSTITUCION_DEL_PAIS TLON: {
    IDENTIDAD: {...}
    INSTANCIADOR: {...}
→ORGANIGRAMA: {
        1. MENSAJERIA: {
            IDENTIDAD: {...}
            CANALES: [...];
            MIEMBROS: {{...}}
            PERCIBIR: {...}
            SERVICIOS:{...}
        }
    }
    REDIRIGIR_MENSAJES: {...}
```

```
}

AGENTE_ESENCIAL: {
    IDENTIDAD: {...}
    INSTANCIADOR: {...}
    PERCIBIR: {...}
}

TEATRO: {
    ESCENOGRAFIA: {...}
    OBRAS:{
        appHolaMundo(destino:cadena) {
            ELENCO: {...}
            DIALOGO: {...}
            MUTIS: {...}
        }
    }
    PUESTA_EN_ESCENA() {...}
}
```

Donde se tiene que, el Teatro, en este caso contiene al obra $appHolaMundo(cadena)$, en la cual se definen internamente qué actores serán partícipes de esta obra, siendo estos un Universo, un País de tipo $TLON$ con una institución llamada Mensajería y dentro de este ambiente también se crea un agente. La aplicación representa en este caso la funcionalidad de una institución que se encarga de imprimir en consola un mensaje, el cual será procesado por uno de sus miembros, el mensajero, el cual se encargará de procesar los datos que ingresen en el canal interno de comunicación $cola\_mensajes$. De esta manera cuando la institución de mensajería inicie su servicio $holaMundo()$, este va a enviar al canal un parámetro de tipo cadena con el contenido "$HolaMundo$". A continuación se muestra cómo la definición de institución que fue traducida a código intermedio:

```
1. MENSAJERIA:{
    IDENTIDAD:{
        nombre:cadena = "Mensajeria Express";
    }
    CANALES: [cola_mensajes*:cadena];
        MIEMBROS:{
        1:MENSAJERO {
```

```
            cola_mensajes*? {
                ==> def mensaje:cadena;
                RECURSOS.console.println(mensaje);
            }
        }
    }
    PERCIBIR: {
        "imprimir hola mundo" {
            holaMundo();
        }
    }
    SERVICIOS:{
        holaMundo() {
            cola_mensajes* <== ["Hola Mundo!"];
        }
    }
}
```

Y su respectiva representación en código intermedio es la siguiente:

```
@instantiate
class public MENSAJERIA extends [sovoralib]Social.Institution
{
    @reference
    attribute private int32 reference = int32(1)
    attribute private string nombre = string("Mensajeria Express")
    attribute private [sovoralib]Social.Channel<string> cola_mensajes = [
       sovoralib]Social.Channel<string>()

    @role 1:MENSAJERO
    @consumes cola_mensajes
    method private void MENSAJEROConsumesFromcola_mensajes()
    {
            ldarg 0
            call object MENSAJERIA::cola_mensajes.consume()
            stloc 0
    ldloc 0
    newarr [sovoralib]System.Collections.Array
    ldloc 0
    stelem 0
    ldstr "RECURSOS.console.println"
    call object [sovoralib]System.Reflection.call(array, string)
```

```
    stloc 1
    ret
}

@receptor
method private void Join0(string cond0)
{
    ldarg 1
    ldstr "imprimir hola mundo"
    call bool [sovoralib]System.String.equality(string, string)
    brtrue L0
    ret
 L0: nop
    ldstr "holaMundo"
    call object [sovoralib]System.Reflection.call(string)
    stloc 0
    ret
}

method private void holaMundo()
{
    newarr [sovoralib]System.Collections.Array
    ldstr "Hola Mundo!"
    stelem 0
    stloc 1
    call object MENSAJERIA::cola_mensajes.send(object)
    ret
}
```

En este caso, se modeló una clase de tipo Institución, donde sus atributos representan su identidad y sus canales de comunicación interna. Dentro de ella se presentan diferentes métodos, los cuales representan los mecanismos de reacción ante mensajes, la definición de sus servicios, y los roles que cumplen cada uno de sus miembros.

Los microservicios generados en la traducción fueron: *ldarg*0, el cual permite obtener la referencia local del objeto. *call*, que permite realizar llamados al sistema mediante microservicios. *stloc* y *ldloc*, que se utilizan para almacenar valores en variables locales. *newarr*, que permite la creación de un arreglo, junto con sus operaciones de *stelem* (y *ldelem*) que permite poder almacenar un elemento en la posición dada en el arreglo. *ldstr*, la cual es usada para cargar una cadena en la pila, *brtrue* para determinar si el valor en la pila es verdadero y saltar a la operación etiquetada. *ret* como indicador de retorno de un método.

También se generaron anotaciones, como @*instantiate*, la cual le indica al compilador, que esa clase debe instanciarse cuando su superclase (bien sea un país, u otra institución) sea

instanciada. @*reference* que indica que el atributo a continuación debe ser usado para encontrar la institución dentro de un organigrama de un país. @*role* y @*consumes* que indican que el siguiente método será asignado al miembro determinado por la primera se encargue de su ejecución, y el segundo, que indica cuál es el canal institucional del cuál ese miembro se encargará de recibir las solicitudes.

7.1. Otros ejemplos

7.1.1. Hola Mundo

Este código representa una impresión de Hola Mundo completo para su ejecución en un nodo.

```
# Definición del ACTOR UNIVERSO
UNIVERSO: {
    # Bitácora de sucesos en el Universo
    HISTORIA:{cadena, cadena};

    # Conjunto de ESTADOS internos
    ESPACIO_TIEMPO: {
        paises:PAIS[];
        agentes:AGENTE[];
    }

    # Parametrización inicial
    INSTANCIADOR: {
        "crear" {
            HISTORIA.agregar("EVENTO:", "Creación del universo");
        }
    }

    # Interacción con mensajes entrantes
    LEYES: {
        PAIS "construir pais" & nombre:cadena & recs:RECURSOS {
             def pais:PAIS = TLON <- [nombre, recs];
             ESPACIO_TIEMPO.paises.agregar(pais);
             retornar pais;
        }
```

```
        # Instanciación de un AGENTE_ESENCIAL
        AGENTE "crear agente" & nombre:cadena & pais_natal:PAIS &
        ↪ institucion:REFERENCIA {
            def agente:AGENTE = AGENTE_ESENCIAL <- [nombre, pais_natal, id];
            EPACIOTIEMPO.agentes.agregar(agente);
            rol:ROL = [agente, "trabajara en", institucion] -> pais_natal;
            [rol] -> agente;
            retornar agente;
        }
    }
}

CONSTITUCION_DEL_PAIS TLON: {
    IDENTIDAD: {
        nombre:cadena;
    }

    #Parametrización del país
    INSTANCIADOR: {
        nombre:cadena & recs:RECURSOS {
            IDENTIDAD.nombre = nombre;
            # Asignación de recursos exclusivos
            [recs] -> ESTE_PAIS.1;
        }
    }

→ORGANIGRAMA: {
        1. MENSAJERIA: {
            IDENTIDAD: {
                nombre:cadena = "Mensajeria Express";
            }
            #Canales de comunicación internos
            CANALES: [cola_mensajes*:cadena];
            MIEMBROS: {
                1:MENSAJERO {
                    # Consume del canal cola_mensajes cuando llega una
                    ↪ cadena
```

```
                cola_mensajes*? { #cuando recibe una cadena
                    ==> def mensaje:cadena;
                    RECURSOS.console.println(mensaje);
                }
            }
        }
        PERCIBIR: {
            "imprimir hola mundo" {
                holaMundo();
            }
        }
        SERVICIOS: {
            # Servicio en el que se envía
            holaMundo() {
                cola_mensajes* <== ["¡Hola Mundo¡"];
            }
        }
    }
}

    # Interfaz de redirección de mensajes entrantes al país desde una
    ↪ aplicación
    REDIRIGIR_MENSAJES: {
        "¡Hola Mundo!" {
            ["imprimir hola mundo"] -> ESTE_PAIS.1;
        }
    }
}

# Definición del ACTOR AGENTE_ESENCIAL
AGENTE_ESENCIAL: {
    # Conjunto de atributos que definen la IDENTIDAD
    IDENTIDAD: {
        nombre:cadena;
        rol:ROL;
    }

    # Parametrización inicial
```

```
    INSTANCIADOR: {
        nombre:cadena {
            IDENTIDAD.nombre = nombre;
        }
    }

    # PERCIBIR, DECIDIR y ACTUAR con los mensajes entrantes
    PERCIBIR: {
        rol:ROL {
            IDENTIDAD.rol = rol;
        }
    }
}

##########
# TEATRO #
##########

TEATRO: {
    ESCENOGRAFIA: {
        # Crea un clúster
        def cluster_A:cluster = system.net.ad_hoc.cluster("CLUSTER TLON_A");

        # Agrega el NODO_1 y sus recursos: CPU, RAM, STORAGE, INTERFACES,
        ↪  INPUTS, OUTPUTS
        cluster_A.addNode("n1", system.net.host("NODO_1"));
    }

    OBRAS: {
        appHolaMundo(destino:cadena) {
            ELENCO: {
                # Se instancia el UNIVERSO
                def universo:UNIVERSO = UNIVERSO <- ["crear"];

                # Se instancia el país tlon_A de tipo TLON_A con los
                ↪  recursos del nodo origen
                def tlon_A:PAIS =
```

```
            ["construir pais", "TLON_A",
            ↪  cluster_A[origen].resources.STANDARD] -> universo;

            # Se instancian el agente de tlon_A
            def alpha:AGENTE =
            ["crear agente", "Agente Alpha", tlon_A, tlon_A.1] ->
            ↪  universo;
        }

        DIALOGO: {
            # Inicio de la interacción de los elementos sociales
            ["¡Hola Mundo!"] -> tlon_A;
        }

        MUTIS: {
            # Se liberan los recursos utilizados, llama al garbage
            ↪  collector
            universo.destruir();
        }
    }
}

PUESTA_EN_ESCENA() {
    # Define el nodo de donde se mostrará el mensaje de ¡Hola Mundo!
    def origen:cadena = "n1";

    appHolaMundo(destino); # Ejecuta la aplicación que muestra ¡Hola
    ↪  Mundo! en el destino
}
}
```

A continuación se muestra su representación en código intermedio LIST.

```
namespace import sovoralib
namespace app
class public TLON extends [sovoralib]Social.Country
{
 attribute private string nombre

 method public instance void constructor(string nombre, [sovoralib]System.
    Cluster.Resources recs)
```

```
{
    ldarg 0
    call instance [sovoralib]Social.Country()
    ldarg 1
    statt string TLON::nombre
    newarr [sovoralib]System.Collections.Array
    ldarg 2
    stelem 0
    stloc 0
    ldloc 0
    ldstr "ESTE_PAIS.1"
    call instance [sovoralib]Social.Institution.reference(string)
    call object [sovoralib]Social.sendMessage(array, instance)
    ret
}


@receptor
method private void Join0(string cond0)
{
    ldarg 1
    ldstr " Hola  Mundo!"
    call bool [sovoralib]System.String.equality(string, string)
    brtrue L0
    ret
 L0: nop
    newarr [sovoralib]System.Collections.Array
    ldstr "imprimir hola mundo"
    stelem 0
    stloc 0
    ldloc 0
    ldstr "ESTE_PAIS.1"
    call instance [sovoralib]Social.Institution.reference(string)
    call object [sovoralib]Social.sendMessage(array, instance)
    ret
}

@instantiate
class public MENSAJERIA extends [sovoralib]Social.Institution
{
 @reference
 attribute private int32 reference = int32(1)
 attribute private string nombre = string("Mensajeria Express")
 attribute private [sovoralib]Social.Channel<string> cola_mensajes = [
     sovoralib]Social.Channel<string>()
```

```
  @role 1:MENSAJERO
  @consumes cola_mensajes
  method private void MENSAJEROConsumesFromcola_mensajes()
  {
     ldarg 0
     call object MENSAJERIA::cola_mensajes.consume()
     stloc 0
     ldloc 0
     newarr [sovoralib]System.Collections.Array
     ldloc 0
     stelem 0
     ldstr "RECURSOS.console.println"
     call object [sovoralib]System.Reflection.call(array, string)
     stloc 1
     ret
  }

  @receptor
  method private void Join0(string cond0)
  {
     ldarg 1
     ldstr "imprimir hola mundo"
     call bool [sovoralib]System.String.equality(string, string)
     brtrue L0
     ret
   L0: nop
     ldstr "holaMundo"
     call object [sovoralib]System.Reflection.call(string)
     stloc 0
     ret
  }

  method private void holaMundo()
  {
     newarr [sovoralib]System.Collections.Array
     ldstr " Hola  Mundo "
     stelem 0
     stloc 1
     call object MENSAJERIA::cola_mensajes.send(object)
     ret
  }

 }
}

class public AgenteEsencial extends [sovoralib]Social.Agent
```

```
{
 attribute private string nombre
 attribute private [sovoralib]Social.Role rol


 method public instance void constructor(string nombre)
 {
    ldarg 0
    call instance [sovoralib]Social.Agent()
    ldarg 1
    statt string AgenteEsencial::nombre
    ret
 }


 @receptor
 method private void Join0([sovoralib]Social.Role rol)
 {
    ldarg 1
    statt [sovoralib]Social.Role AgenteEsencial::rol
    ret
 }
}


class public Universo extends [sovoralib]Social.Universe
{
 attribute private [sovoralib]System.Collections.Hashtable<string, string>
     historia = [sovoralib]System.Collections.Hashtable<string, string>()
 attribute private [sovoralib]Social.Country[] paises
 attribute private [sovoralib]Social.Agent[] agentes

 method public instance void constructor(string cond0)
 {
    ldarg 0
    call instance [sovoralib]Social.Universe()
    ldarg 1
    ldstr "crear"
    call bool [sovoralib]System.String.equality(string, string)
    brtrue L0
    ret
  L0: nop
    newarr [sovoralib]System.Collections.Array
    ldstr "EVENTO:"
    stelem 0
    ldstr "Creaci n del universo"
```

```
    stelem 1
    ldstr "HISTORIA.agregar"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 0
    ret
}

@receptor
method private [sovoralib]Social.Country Join0(string cond0, string nombre, [
    sovoralib]System.Cluster.Resources recs)
{
    ldarg 1
    ldstr "construir pais"
    call bool [sovoralib]System.String.equality(string, string)
    brtrue L0
    ret
 L0: nop
    newarr [sovoralib]System.Collections.Array
    ldarg 2
    stelem 0
    ldarg 3
    stelem 1
    ldstr "TLON"
    call instance [sovoralib]Social.instance(array, string)
    stloc 0
    ldloc 0
    stloc 1
    newarr [sovoralib]System.Collections.Array
    ldloc 1
    stelem 0
    ldstr "ESPACIO_TIEMPO.paises.agregar"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 2
    ldloc 1
    ret
    ret
}

@receptor
method private [sovoralib]Social.Agent Join1(string cond0, string nombre, [
    sovoralib]Social.Country pais_natal, [sovoralib]Social.Institution.
    Reference institucion)
{
    ldarg 1
    ldstr "crear agente"
    call bool [sovoralib]System.String.equality(string, string)
```

```
    brtrue L0
    ret
  L0: nop
    newarr [sovoralib]System.Collections.Array
    ldarg 2
    stelem 0
    ldarg 3
    stelem 1
    ldarg 0
    ldstr "id"
    ldatt object [sovoralib]System.Reflection.resolver(instance, string)
    stelem 2
    ldstr "AGENTE_ESENCIAL"
    call instance [sovoralib]Social.instance(array, string)
    stloc 0
    ldloc 0
    stloc 1
    newarr [sovoralib]System.Collections.Array
    ldloc 1
    stelem 0
    ldstr "EPACIOTIEMPO.agentes.agregar"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 2
    newarr [sovoralib]System.Collections.Array
    ldloc 1
    stelem 0
    ldstr "trabajara en"
    stelem 1
    ldarg 4
    stelem 2
    stloc 3
    ldloc 3
    ldarg 3
    call object [sovoralib]Social.sendMessage(array, instance)
    stloc 4
    ldloc 4
    stloc 5
    newarr [sovoralib]System.Collections.Array
    ldloc 5
    stelem 0
    stloc 6
    ldloc 6
    ldstr "agente"
    ldarg 0
    ldstr "agente"
    ldatt object [sovoralib]System.Reflection.resolver(instance, string)
```

```
    call object [sovoralib]Social.sendMessage(array, instance)
    ldloc 1
    ret
    ret
 }
}

class public Teatro extends [sovoralib]Social.Theater
{
 method private void appHolaMundo(string destino)
 {
    newarr [sovoralib]System.Collections.Array
    ldstr "crear"
    stelem 0
    ldstr "UNIVERSO"
    call instance [sovoralib]Social.instance(array, string)
    stloc 1
    ldloc 1
    stloc 2
    ldloc 2
    newarr [sovoralib]System.Collections.Array
    ldstr "construir pais"
    stelem 0
    ldstr "TLON_A"
    stelem 1
    ldarg 0
    ldstr "cluster_A[origen].resources.STANDARD"
    ldatt object [sovoralib]System.Reflection.resolver(instance, string)
    stelem 2
    stloc 3
    ldloc 3
    ldloc 2
    call object [sovoralib]Social.sendMessage(array, instance)
    stloc 4
    ldloc 4
    stloc 5
    ldloc 5
    newarr [sovoralib]System.Collections.Array
    ldstr "crear agente"
    stelem 0
    ldstr "Agente Alpha"
    stelem 1
    ldloc 5
    stelem 2
    stelem 3
    stloc 6
```

```
    ldloc 6
    ldloc 2
    call object [sovoralib]Social.sendMessage(array, instance)
    stloc 7
    ldloc 7
    stloc 8
    ldloc 8
    newarr [sovoralib]System.Collections.Array
    ldstr " Hola  Mundo!"
    stelem 0
    stloc 0
    ldloc 0
    ldstr "tlon_A"
    ldarg 0
    ldstr "tlon_A"
    ldatt object [sovoralib]System.Reflection.resolver(instance, string)
    call object [sovoralib]Social.sendMessage(array, instance)
    ldstr "universo.destruir"
    call object [sovoralib]System.Reflection.call(string)
    stloc 0
    ret
}

@entrypoint
method public static void Main()
{
    newarr [sovoralib]System.Collections.Array
    ldstr "CLUSTER TLON_A"
    stelem 0
    ldstr "system.net.ad_hoc.cluster"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 0
    ldloc 0
    stloc 1
    newarr [sovoralib]System.Collections.Array
    ldstr "n1"
    stelem 0
    newarr [sovoralib]System.Collections.Array
    ldstr "NODO_1"
    stelem 0
    ldstr "system.net.host"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 2
    ldloc 2
    stelem 1
    ldstr "cluster_A.addNode"
```

```
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 3
    ldstr "n1"
    stloc 4
    newarr [sovoralib]System.Collections.Array
    ldarg 0
    ldstr "destino"
    ldatt object [sovoralib]System.Reflection.resolver(instance, string)
    stelem 0
    ldstr "appHolaMundo"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 5
    ret
 }
}
```

7.1.2. Envío de mensaje de un nodo a otro:

Este código representa la interacción de dos países contenidos en el mismo universo, en el cual el país A envía un mensaje al país B. Donde cada país está asociado a un nodo específico.

```
# Definición del ACTOR UNIVERSO
UNIVERSO:
{
    # Bitácora de sucesos en el Universo
    HISTORIA:{cadena, cadena};
    # Conjunto de ESTADOS internos
    ESPACIO_TIEMPO: {
        duracion_dia:entero;
        hora_actual:entero = 0;
        dias_transcurridos:entero = 0;
        avance_tiempo:entero = 1000; # Por defecto el avance de tiempo se
        ↪  realiza cada 1000ms
        paises:PAIS[];
        agentes:AGENTE[];
    }
    # Parametrización inicial
    INSTANCIADOR: {
        horas:entero {
            ESTADOS.duracion_dia = horas;
            #Inicia el avance del tiempo
```

```
            ["avanzar tiempo"] -> SI_MISMO;
        }
    }
    # Interacción con mensajes entrantes
    LEYES: {
        "avanzar tiempo" {
            ESTADOS.hora_actual = (ESTADOS.hora_actual + 1) %
            ↪  ESTADOS.duracion_dia;
            si (ESTADOS.hora_actual == 0) {
                ESTADOS.dias_transcurridos = ESTADOS.dias_transcurridos + 1;
            }
            system.social.actors.broadcast([ESTADOS.paises,
            ↪  ESTADOS.agentes],
            ["hora actual", hora_actual]);
            system.timeout(ESTADOS.avance_tiempo); # Se esperan 1000ms
            ["avanzar tiempo"] -> SI_MISMO;
        }
        PAIS "construir pais" & nombre:cadena & recs:RECURSOS &
        ↪  idioma_oficial:IDIOMA {
             def pais:PAIS = TLON <- [nombre, recs, idioma_oficial];
             ESPACIO_TIEMPO.paises.agregar(pais);
             HISTORIA.agregar("EVENTO:" + ESTADOS.hora_actual, "El dia "
             + ESTADOS.dias_transcurridos + " a las " + ESTADOS.hora_actual
             + " se creo el pais " + nombre);
             retornar pais;
        }
        # Instanciación de un AGENTE_ESENCIAL
        AGENTE "crear agente" & nombre:cadena & pais_natal:PAIS &
        ↪  institucion:REFERENCIA {
            # Generación de número de identificación
            def id:entero = ["registrar agente", nombre] -> pais_natal;

            def agente:AGENTE = AGENTE_ESENCIAL <- [nombre, pais_natal, id];
            EPACIOTIEMPO.agentes.agregar(agente);

            rol:ROL = [agente, "trabajara en", institucion] -> pais_natal;
            [rol] -> agente;
```

```
            retornar agente;
        }
        # Instanciación de un agente MENSAJERO
        AGENTE "crear agente mensajero" & nombre:cadena
        & pais_natal:PAIS & institucion:REFERENCIA {
            # Generación de número de identificación
            def id:entero = ["registrar agente", nombre] -> pais_natal;

            def agente:AGENTE = MENSAJERO <- [nombre, pais_natal, id];
            ESTADOS.agentes.agregar(agente);

            rol:ROL = [agente, "trabajara en", institucion] -> pais_natal;
            [rol] -> agente;

            retornar agente;
        }
    }
}

# Definición del conjunto de IDIOMAS
IDIOMAS:
{
    # Percepciones comunes ante mensajes para agentes que hablan el idioma
    TLONES_A: {
        "hola, soy " & agente:AGENTE & nombre:cadena & id:entero {
            [
               || agregarConocido(nombre, id);
               || ["hola, soy ", SI_MISMO, IDENTIDAD.nombre, INVENTARIO.id]
                ↪  -> agente;
            ]
        }
    }

    TLONES_B: {
        "hello, my name is " & agente:AGENTE & name:cadena & id:entero {
            [
               || agregarConocido(nombre, id);
```

```
                || ["hello, my name is ", SI_MISMO, IDENTIDAD.nombre,
                ↪  INVENTARIO.id] -> SI_MISMO;
            ]
        }
    }
}

CONSTITUCION_DEL_PAIS TLON: {
    IDENTIDAD: {
        nombre:cadena;
    }
    # Bitácora de sucesos en el país
    HISTORIA: {entero, cadena};
    ESTADOS: {}
    #Parametrización del país
    INSTANCIADOR: {
        nombre:cadena & recs:RECURSOS & idioma_oficial:IDIOMA {
            IDENTIDAD.nombre = nombre;
            RECURSOS.agregar(recs);
            CULTURA.idioma_oficial = idioma_oficial;

            # Asignación de recursos exclusivos
            [recs.OUTPUTS] -> ESTE_PAIS.1.1;
        }
    }
    CULTURA: {
        idioma_oficial:IDIOMA;
    }
→ORGANIGRAMA: {
        1. GOBIERNO: {
            IDENTIDAD: {
                nombre:cadena = "Gobierno del país";
            }
            # Bitácora de sucesos en el Gobierno
            HISTORIA: {entero, cadena};
            DISPONIBILIDAD = [de 8 a 12, de 12 a 17];
            MIEMBROS: {
                1:PRESIDENTE {}
```

```
    2:GOBERNANTE {}
}
PERCIBIR: {
    ROL agente:AGENTE {
        retornar empleo_disponible();
    }
    mensaje:cadena & remitente:PAIS {
        ["mostrar mensaje", mensaje, remitente] -> DIRIGENTES;
    }
}

#Grupo de miembros del gobierno que toman decisiones
GRUPO_LOGISTICO DIRIGENTES: {
    IDENTIDAD: {
        nombre:cadena = "Dirigentes del país";
    }
    ESTADOS: {}
    PARTICIPANTES = [PRESIDENTE, GOBERNANTE, REGISTRADOR];
    #DEFINE UN CONJUNTO DE REGLAS, y DECIDE SI SON TODAS O
    ↪  SOLO UNA
    #A Múltiples niveles
    DECIDIR: {
        "mostrar mensaje" & mensaje:cadena & remitente:cadena {
            # Decisión que se somete a votación por los
            ↪  miembros
            # del grupo para escoger la forma de mostrar un
            ↪  mensaje
            [
                ++ RECURSOS.console.println("Mensaje de " +
                ↪  remitente + ": " + mensaje);
                ++ RECURSOS.console.println(mensaje);
            ] usando votacion(2);
        }
    }
}

2. REGISTRADURIA: {
    IDENTIDAD: {
```

```
    nombre:cadena = "Registraduria de la poblacion";
}
# Bitácora de sucesos en la Registraduría
HISTORIA: {entero, cadena};
ESTADOS: {
    contador:entero = 0;
    poblacion:{entero, cadena};
}
CANALES: [cola_registros*:cadena, cola_ids*:entero];
DISPONIBILIDAD = [de 8 a 12, de 12 a 17];
MIEMBROS: {
    1:REGISTRADOR {
        # Consume del canal cola_mensajes cuando llega una
        ↪ tupla
        cola_registros*? {
            ==> def nombre:cadena; #cuando recibe una
            ↪ cadena

            ESTADOS.contador = ESTADOS.contador + 1;
            ESTADOS.poblacion.agregar(ESTADOS.contador,
            ↪ nombre);

            cola_ids* <== ESTADOS.contador;
        }
    }
}
PERCIBIR: {
    ROL agente:AGENTE {
        retornar empleo_disponible();
    }
    entero nombre:cadena {
        retornar registrarAgente(nombre);
    }
}
SERVICIOS: {
    # Servicio en el que se registra un agente
    # y se le retorna su número de identificación
    entero registrarAgente(nombre:cadena) {
```

```
                [
                    || cola_registros* <== nombre;
                    || cola_ids* ==> def id:entero; retornar id;
                ]
            }
        }
    }
}
2. OTROS: {
    1. MENSAJERIA: {
        IDENTIDAD: {
            nombre:cadena = "Mensajeria Express";
        }
        # Bitácora de sucesos en la Mensajería Express
        HISTORIA: {entero, cadena};
        #canales de comunicación internos
        CANALES: [cola_mensajes*:[]];
        DISPONIBILIDAD = [de 8 a 17];
        MIEMBROS: {
            1:MENSAJERO {
                # Consume del canal cola_mensajes cuando llega una
                ↪  tupla
                cola_mensajes*? { #cuando recibe una tupla
                    ==>[0] INVENTARIO.mensaje;
                    ==>[1] INVENTARIO.pais_destino;
                    ==>[2] INVENTARIO.destinatario;

                    INVENTARIO.pais_origen = ESTE_PAIS;

                    desplazarseA(INVENTARIO.pais_destino,
                    ↪  INVENTARIO.destinatario);
                    entregarMensaje();
                    desplazarseA(ESTE_PAIS, ESTA_INSTITUCION);
                }
            }
        }
        PERCIBIR: {
            ROL agente:AGENTE {
```

```
                    retornar empleo_disponible();
                }
                origen:PAIS & destino:REFERENCIA & mensaje:cadena {
                    enviarMensaje(mensaje, origen, destino);
                }
            }
            SERVICIOS: {
                # Servicio en el que se envía
                enviarMensaje(mensaje:cadena, pais_origen:PAIS,
                institucion_destino:REFERENCIA) {
                    cola_mensajes* <== [mensaje, pais_origen,
                    ↪  institucion_destino];
                }
            }
        }
    }
}

# Interfaz de redirección de mensajes entrantes al país desde una
↪  aplicación
REDIRIGIR_MENSAJES:
{
    # Redirección del mensaje a la Registraduría con
    # retorno del número de identificación generado
    entero "registrar agente" & nombre:cadena {
        retornar [nombre] -> ESTE_PAIS.1.2;
    }

    # Redirección del mensaje a Mensajería Express
    "enviar mensaje" & origen:PAIS & destino:REFERENCIA & mensaje:cadena
    ↪  {
        [origen, destino, mensaje] -> ESTE_PAIS.2.1;
    }

    # Redirección de la solicitud de un candidato buscando empleo
    ROL candidato:AGENTE & "trabajara en" & institucion:REFERENCIA {
        retornar [candidato] -> institucion;
    }
```

```
    }
}

# Definición del ACTOR AGENTE_ESENCIAL
AGENTE_ESENCIAL:
{
    # Conjunto de atributos que definen la IDENTIDAD
    IDENTIDAD:
    {
        nombre:cadena;
        pais_natal:PAIS;
        rol:ROL;
    }

    # Bitácora de sucesos del agente
    HISTORIA: {entero, cadena};

    # Conjunto de ESTADOS internos
    ESTADOS: {
        ubicacion_actual:UBICACION; # Ubicación del agente
    }

    # Definición de su INVENTARIO como un AMBIENTE
    INVENTARIO: {
        id:entero;
        nombres_de_conocidos:{entero,cadena};
    }

    # Parametrización inicial
    INSTANCIADOR: {
        nombre:cadena & pais_natal:PAIS & id:entero {
            IDENTIDAD.nombre = nombre;
            IDENTIDAD.pais_natal = pais_natal;
            PERCIBIR.agregar(pais_natal.CULTURA.idioma_oficial);
            INVENTARIO.id = id;
        }

        nombre:cadena & idioma_natal:IDIOMA {
```

```
            IDENTIDAD.nombre = nombre;
            PERCIBIR.agregar(idioma_natal);
        }
    }

    # PERCIBIR, DECIDIR y ACTUAR con los mensajes entrantes
    PERCIBIR: {
        rol:ROL {
            IDENTIDAD.rol = rol;
        }

        "hora actual" & hora_actual:entero {
            si hora_actual == ROL.hora_libre {
                system.social.actors.disperse(ESTE_AGENTE, ROL.institucion);
            }
            si_no {
                system.social.actors.recopile(ESTE_AGENTE, ROL.institucion);
            }
        }
    }

    #Conjunto de funciones que conoce
    CONOCIMIENTO:
    {
        # Función que permite agregar otros agentes a la lista de
        ↪ conocidos
        agregarConocido(nombre:cadena, id:entero) {
            INVENTARIO.nombres_de_conocidos.agregar(id, nombre);
        }

        # Función que permite a un agente desplazarse a otra institución
        ↪ del país actual
        desplazarseA(institucion:REFERENCIA) {
            system.ambients.moveFrom(ESTADOS.ubicacion_actual,
            ↪ system.actor.serialize(SI_MISMO)).moveTo(institucion);
        }

        entero votacion(opciones:entero) {
```

```
            retornar system.random() * opciones;
        }
    }
}

AGENTE MENSAJERO: {
    INVENTARIO: {
        mensaje:cadena;
        pais_origen:PAIS;
        pais_destino:PAIS;
        destinatario:REFERENCIA;
    }

    CONOCIMIENTO: {
        # Función que ejecuta la acción de entregar un mensaje
        entregarMensaje() {
            [INVENTARIO.mensaje, INVENTARIO.pais_origen] ->
            ↪  INVENTARIO.destinatario;

            # Reinicio de variables del inventario
            INVENTARIO.mensaje = nulo;
            INVENTARIO.pais_origen = nulo;
            INVENTARIO.pais_destino = nulo;
            INVENTARIO.destinatario = nulo;
        }

        # Función que permite desplazarse a una INSTITUCION de otro PAIS
        desplazarseA(pais:PAIS, institucion:REFERENCIA) {
            system.ambients.moveFrom(ESTADOS.ubicacion,
            ↪  system.actor.serialize(SI_MISMO)).moveTo(pais, institucion);
        }
    }
}

##########
# TEATRO #
##########
```

```
TEATRO: {
    ESCENOGRAFIA: {
        # Crea un clúster
        def cluster_A:cluster = system.net.ad_hoc.cluster("CLUSTER TLON_A");

        # Agrega el NODO_1 y sus recursos: CPU, RAM, STORAGE, INTERFACES,
        ↪  INPUTS, OUTPUTS
        cluster_A.addNode("n1", system.net.host("NODO_1"));

        # Agrega el NODO_2 y sus recursos: CPU, RAM, STORAGE, INTERFACES,
        ↪  INPUTS, OUTPUTS
        cluster_A.addNode("n2", system.net.host("NODO_2"));
    }

    OBRAS: {
        appEnviarMensaje(origen:cadena, destino:cadena, mensaje:cadena) {
            ELENCO: {
                # Se instancia el UNIVERSO con días que duran "24 horas"
                def universo:UNIVERSO = UNIVERSO <- [24];

                # Se instancia el país tlon_A de tipo TLON_A con los
                ↪  recursos del nodo origen
                def tlon_A:PAIS =
                ["construir pais", "TLON_A",
                ↪  cluster_A[origen].resources.STANDARD, IDIOMAS.TLONES_A]
                ↪  -> universo;

                # Se instancian los agentes de tlon_A
                def alpha:AGENTE =
                ["crear agente", "Agente Alpha", tlon_A, tlon_A.1.2] ->
                ↪  universo;
                def beta:AGENTE =
                ["crear agente mensajero", "Agente beta", tlon_A,
                ↪  tlon_A.2.2] -> universo;

                # Se instancia el país tlon_B de tipo TLON_A con los
                ↪  recursos del nodo destino
```

```
            def tlon_B:PAIS = ["construir pais", "TLON_B",
            ↪  cluster_A[destino].resources.STANDARD_OUTPUTS,
            ↪  IDIOMAS.TLONES_B] -> universo;

            # Se instancian los agentes de tlon_B
            def gamma:AGENTE =
            ["crear agente", "Agente Gama", tlon_B, tlon_B.1.1] ->
            ↪  universo;
            def delta:AGENTE =
            ["crear agente", "Agente Delta", tlon_B, tlon_B.1.1] ->
            ↪  universo;
            def epsilon:AGENTE =
            ["crear agente", "Agente Epsilon", tlon_B, tlon_B.1.2] ->
            ↪  universo;
        }
        DIALOGO: {
            # Inicio de la interacción de los elementos sociales
            ["enviar mensaje", tlon_A, tlon_B.1, mensaje] -> tlon_A;
        }

        MUTIS: {
            # Se liberan los recursos utilizados, llama al garbage
            ↪  collector
            universo.destruir();
        }
    }
}

PUESTA_EN_ESCENA() {
    # Define el nodo de origen del mensaje como el nodo "n1"
    def origen:cadena = "n1";

    cluster_A[origen].console.println("Ingrese nodo destino: ");

    # Escribir "n2"
    def destino:cadena = cluster_A[origen].console.readln();

    cluster_A[origen].console.println("Ingrese mensaje a enviar: ");
```

```
        # Escribir el mensaje a enviar a "n2"
        def mensaje:cadena = cluster_A[origen].console.readln();

        appEnviarMensaje(origen, destino, mensaje);
        # Ejecuta la aplicación que envía un mensaje desde "n1" hasta "n2"
    }
}
```

A continuación se muestra su representación en código intermedio LIST.

```
namespace import sovoralib
namespace app

class public TLONES_A extends [sovoralib]Social.Language
{

 @receptor
 method private void Join0(string cond0, [sovoralib]Social.Agent agente,
     string nombre, int32 id)
 {
    ldarg 1
    ldstr "hola, soy "
    call bool [sovoralib]System.String.equality(string, string)
    brtrue L0
    ret
  L0: nop
    parallel 2
    thread 0
    newarr [sovoralib]System.Collections.Array
    ldarg 3
    stelem 0
    ldarg 4
    stelem 1
    ldstr "agregarConocido"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 0
    thread 1
    newarr [sovoralib]System.Collections.Array
    ldstr "hola, soy "
    stelem 0
    ldarg 0
    ldstr "SI_MISMO"
    ldatt object [sovoralib]System.Reflection.resolver(instance, string)
    stelem 1
    ldarg 3
```

```
    stelem 2
    ldarg 4
    stelem 3
    stloc 1
    ldloc 1
    ldstr "agente"
    ldarg 0
    ldstr "agente"
    ldatt object [sovoralib]System.Reflection.resolver(instance, string)
    call object [sovoralib]Social.sendMessage(array, instance)
    parallel end
    ret
 }
}

class public TLONES_B extends [sovoralib]Social.Language
{

 @receptor
 method private void Join0(string cond0, [sovoralib]Social.Agent agente,
    string name, int32 id)
 {
    ldarg 1
    ldstr "hello, my name is "
    call bool [sovoralib]System.String.equality(string, string)
    brtrue L0
    ret
  L0: nop
    parallel 2
    thread 0
    newarr [sovoralib]System.Collections.Array
    ldarg 0
    ldstr "nombre"
    ldatt object [sovoralib]System.Reflection.resolver(instance, string)
    stelem 0
    ldarg 4
    stelem 1
    ldstr "agregarConocido"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 0
    thread 1
    newarr [sovoralib]System.Collections.Array
    ldstr "hello, my name is "
    stelem 0
    ldarg 0
    ldstr "SI_MISMO"
```

```
    ldatt object [sovoralib]System.Reflection.resolver(instance, string)
    stelem 1
    ldarg 0
    ldstr "IDENTIDAD.nombre"
    ldatt object [sovoralib]System.Reflection.resolver(instance, string)
    stelem 2
    ldarg 4
    stelem 3
    stloc 1
    ldloc 1
    ldstr "SI_MISMO"
    ldarg 0
    ldstr "SI_MISMO"
    ldatt object [sovoralib]System.Reflection.resolver(instance, string)
    call object [sovoralib]Social.sendMessage(array, instance)
    parallel end
    ret
 }
}

class public TLON extends [sovoralib]Social.Country
{
 attribute private string nombre
 attribute private [sovoralib]System.Collections.Hashtable<int32, string>
     historia = [sovoralib]System.Collections.Hashtable<int32, string>()
 attribute private [sovoralib]Social.Language idioma_oficial

 method public instance void constructor(string nombre, [sovoralib]System.
     Cluster.Resources recs, [sovoralib]Social.Language idioma_oficial)
 {
    ldarg 0
    call instance [sovoralib]Social.Country()
    ldarg 1
    statt string TLON::nombre
    newarr [sovoralib]System.Collections.Array
    ldarg 2
    stelem 0
    ldstr "RECURSOS.agregar"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 0
    ldarg 3
    statt [sovoralib]Social.Language TLON::idioma_oficial
    newarr [sovoralib]System.Collections.Array
    ldarg 0
    ldstr "recs.OUTPUTS"
    ldatt object [sovoralib]System.Reflection.resolver(instance, string)
```

```
    stelem 0
    stloc 1
    ldloc 1
    ldstr "ESTE_PAIS.1.1"
    call instance [sovoralib]Social.Institution.reference(string)
    call object [sovoralib]Social.sendMessage(array, instance)
    ret
}


@receptor
method private int32 Join0(string cond0, string nombre)
{
    ldarg 1
    ldstr "registrar agente"
    call bool [sovoralib]System.String.equality(string, string)
    brtrue L0
    ret
 L0: nop
    newarr [sovoralib]System.Collections.Array
    ldarg 2
    stelem 0
    stloc 0
    ldloc 0
    instancia de [sovoralib]Social.Institution.Reference
    call object [sovoralib]Social.sendMessage(array, instance)
    stloc 1
    ldloc 1
    ret
    ret
}

@receptor
method private void Join1(string cond0, [sovoralib]Social.Country origen, [
    sovoralib]Social.Institution.Reference destino, string mensaje)
{
    ldarg 1
    ldstr "enviar mensaje"
    call bool [sovoralib]System.String.equality(string, string)
    brtrue L0
    ret
 L0: nop
    newarr [sovoralib]System.Collections.Array
    ldarg 2
    stelem 0
    ldarg 3
```

```
    stelem 1
    ldarg 4
    stelem 2
    stloc 0
    ldloc 0
    ldstr "ESTE_PAIS.2.1"
    call instance [sovoralib]Social.Institution.reference(string)
    call object [sovoralib]Social.sendMessage(array, instance)
    ret
}

@receptor
method private [sovoralib]Social.Role Join2([sovoralib]Social.Agent candidato
    , string cond1, [sovoralib]Social.Institution.Reference institucion)
{
    ldarg 2
    ldstr "trabajara en"
    call bool [sovoralib]System.String.equality(string, string)
    brtrue L0
    ret
 L0: nop
    newarr [sovoralib]System.Collections.Array
    ldarg 1
    stelem 0
    stloc 0
    ldloc 0
    ldarg 3
    call object [sovoralib]Social.sendMessage(array, instance)
    stloc 1
    ldloc 1
    ret
    ret
}

@instantiate
class public GOBIERNO extends [sovoralib]Social.Institution
{
 @reference
 attribute private int32 reference = int32(1)
 attribute private string nombre = string("Gobierno del pa s")
 attribute private [sovoralib]System.Collections.Hashtable<int32, string>
     historia = [sovoralib]System.Collections.Hashtable<int32, string>()
 attribute private [sovoralib]Social.Schedule horario0 = [sovoralib]Social.
     Schedule(8, 12)
 attribute private [sovoralib]Social.Schedule horario1 = [sovoralib]Social.
     Schedule(12, 17)
```

```
@role 1:PRESIDENTE
method private void PRESIDENTE()
{
   ret
}

@role 2:GOBERNANTE
method private void GOBERNANTE()
{
   ret
}

@receptor
method private [sovoralib]Social.Role Join0([sovoralib]Social.Agent agente)
{
   ldstr "empleo_disponible"
   call object [sovoralib]System.Reflection.call(string)
   stloc 0
   ldloc 0
   ret
   ret
}

@receptor
method private void Join1(string mensaje, [sovoralib]Social.Country
   remitente)
{
   newarr [sovoralib]System.Collections.Array
   ldstr "mostrar mensaje"
   stelem 0
   ldarg 1
   stelem 1
   ldarg 2
   stelem 2
   stloc 0
   ldloc 0
   ldstr "DIRIGENTES"
   ldarg 0
   ldstr "DIRIGENTES"
   ldatt object [sovoralib]System.Reflection.resolver(instance, string)
   call object [sovoralib]Social.sendMessage(array, instance)
   ret
}

@instantiate
```

```
class public DIRIGENTES extends [sovoralib]Social.LogisticsGroup
{
 attribute private string nombre = string("Dirigentes del pa s")
 attribute private [sovoralib]Social.LogisticsGroup.Member[] participantes =
     [sovoralib]Social.LogisticsGroup.Member[]("PRESIDENTE","GOBERNANTE","
    REGISTRADOR")

 @receptor
 method private void Join0(string cond0, string mensaje, string remitente)
 {
    ldarg 1
    ldstr "mostrar mensaje"
    call bool [sovoralib]System.String.equality(string, string)
    brtrue L0
    ret
  L0: nop
    newarr [sovoralib]System.Collections.Array
    ldstr "Mensaje de "
    ldarg 3
    call string [sovoralib]System.String.concat(string, string)
    stloc 2
    ldloc 2
    ldstr ": "
    call string [sovoralib]System.String.concat(string, string)
    stloc 3
    ldloc 3
    ldarg 2
    call string [sovoralib]System.String.concat(string, string)
    stloc 4
    stelem 0
    ldstr "RECURSOS.console.println"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 5
    newarr [sovoralib]System.Collections.Array
    ldarg 2
    stelem 0
    ldstr "RECURSOS.console.println"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 6
    ldarg 0
    ldstr "votacion"
    ldatt object [sovoralib]System.Reflection.resolver(instance, string)
    stloc 7
    newarr [sovoralib]System.Collections.Array
    ldint 2
    stelem 0
```

```
    ldarg 0
    ldatt [sovoralib]Social.LogisticsGroup.Member[] DIRIGENTES::
        participantes
    ldstr "votacion"
    call int32 [sovoralib]Social.LogisticsGroup.decide(array, string, array)
  L2: switch (L3, L4)
  L3: nop
    newarr [sovoralib]System.Collections.Array
    ldstr "Mensaje de "
    ldarg 3
    call string [sovoralib]System.String.concat(string, string)
    stloc 8
    ldloc 8
    ldstr ": "
    call string [sovoralib]System.String.concat(string, string)
    stloc 9
    ldloc 9
    ldarg 2
    call string [sovoralib]System.String.concat(string, string)
    stloc 10
    stelem 0
    ldstr "RECURSOS.console.println"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 11
    br L1
  L4: nop
    newarr [sovoralib]System.Collections.Array
    ldarg 2
    stelem 0
    ldstr "RECURSOS.console.println"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 12
    br L1
  L1: ret
 }
}

@instantiate
class public REGISTRADURIA extends [sovoralib]Social.Institution
{
 @reference
 attribute private int32 reference = int32(2)
 attribute private string nombre = string("Registraduria de la poblacion")
 attribute private [sovoralib]System.Collections.Hashtable<int32, string>
     historia = [sovoralib]System.Collections.Hashtable<int32, string>()
 attribute private int32 contador = int32(0)
```

```
attribute private [sovoralib]System.Collections.Hashtable<int32, string>
    poblacion
attribute private [sovoralib]Social.Channel<string> cola_registros = [
    sovoralib]Social.Channel<string>()
attribute private [sovoralib]Social.Channel<int32> cola_ids = [sovoralib]
    Social.Channel<int32>()
attribute private [sovoralib]Social.Schedule horario0 = [sovoralib]Social.
    Schedule(8, 12)
attribute private [sovoralib]Social.Schedule horario1 = [sovoralib]Social.
    Schedule(12, 17)

@role 1:REGISTRADOR
@consumes cola_registros
method private void REGISTRADORConsumesFromcola_registros()
{
   ldarg 0
   call object REGISTRADURIA::cola_registros.consume()
   stloc 0
   ldatt string REGISTRADURIA::nombre
   ldatt int32 REGISTRADURIA::contador
   cast float64
   ldint 1
   toflt
   sum
   stloc 1
   ldloc 1
   statt int32 REGISTRADURIA::contador
   newarr [sovoralib]System.Collections.Array
   ldatt int32 REGISTRADURIA::contador
   stelem 0
   ldatt string REGISTRADURIA::nombre
   stelem 1
   ldstr "ESTADOS.poblacion.agregar"
   call object [sovoralib]System.Reflection.call(array, string)
   stloc 2
   call object REGISTRADURIA::cola_ids.send(object)
   ret
}

@receptor
method private [sovoralib]Social.Role Join0([sovoralib]Social.Agent agente)
{
   ldstr "empleo_disponible"
   call object [sovoralib]System.Reflection.call(string)
   stloc 0
   ldloc 0
```

```
      ret
      ret
   }

   @receptor
   method private int32 Join1(string nombre)
   {
      newarr [sovoralib]System.Collections.Array
      ldarg 1
      stelem 0
      ldstr "registrarAgente"
      call object [sovoralib]System.Reflection.call(array, string)
      stloc 0
      ldloc 0
      ret
      ret
   }

   method private int32 registrarAgente(string nombre)
   {
      parallel 2
      thread 0
      call object REGISTRADURIA::cola_registros.send(object)
      thread 1
      call object REGISTRADURIA::cola_ids.consume(object)
      stloc 4
      ldloc 4
      ldloc 4
      ret
      parallel end
      ret
   }

 }
}

@instantiate
class public OTROS extends [sovoralib]Social.Institution
{
 @reference
 attribute private int32 reference = int32(2)


 @instantiate
 class public MENSAJERIA extends [sovoralib]Social.Institution
 {
```

```
@reference
attribute private int32 reference = int32(1)
attribute private string nombre = string("Mensajeria Express")
attribute private [sovoralib]System.Collections.Hashtable<int32, string>
    historia = [sovoralib]System.Collections.Hashtable<int32, string>()
attribute private [sovoralib]Social.Channel<[sovoralib]System.Collections.
    Array> cola_mensajes = [sovoralib]Social.Channel<[sovoralib]System.
    Collections.Array>()
attribute private [sovoralib]Social.Schedule horario0 = [sovoralib]Social.
    Schedule(8, 17)

@role 1:MENSAJERO
@consumes cola_mensajes
method private void MENSAJEROConsumesFromcola_mensajes()
{
    ldarg 0
    call object MENSAJERIA::cola_mensajes.consume()
    ldelem 0
    ldelem 1
    ldelem 2
    ldarg 0
    ldstr "ESTE_PAIS"
    ldatt object [sovoralib]System.Reflection.resolver(instance, string)
    ldarg 0
    ldstr "pais_origen"
    statt object [sovoralib]System.Reflection.resolver(instance, string)
    newarr [sovoralib]System.Collections.Array
    ldarg 0
    ldstr "INVENTARIO.pais_destino"
    ldatt object [sovoralib]System.Reflection.resolver(instance, string)
    stelem 0
    ldarg 0
    ldstr "INVENTARIO.destinatario"
    ldatt object [sovoralib]System.Reflection.resolver(instance, string)
    stelem 1
    ldstr "desplazarseA"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 0
    ldstr "entregarMensaje"
    call object [sovoralib]System.Reflection.call(string)
    stloc 1
    newarr [sovoralib]System.Collections.Array
    ldarg 0
    ldstr "ESTE_PAIS"
    ldatt object [sovoralib]System.Reflection.resolver(instance, string)
    stelem 0
```

```
    ldarg 0
    ldstr "ESTA_INSTITUCION"
    ldatt object [sovoralib]System.Reflection.resolver(instance, string)
    stelem 1
    ldstr "desplazarseA"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 2
    ret
}

@receptor
method private [sovoralib]Social.Role Join0([sovoralib]Social.Agent agente)
{
    ldstr "empleo_disponible"
    call object [sovoralib]System.Reflection.call(string)
    stloc 0
    ldloc 0
    ret
    ret
}

@receptor
method private void Join1([sovoralib]Social.Country origen, [sovoralib]
    Social.Institution.Reference destino, string mensaje)
{
    newarr [sovoralib]System.Collections.Array
    ldarg 3
    stelem 0
    ldarg 1
    stelem 1
    ldarg 2
    stelem 2
    ldstr "enviarMensaje"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 0
    ret
}

method private void enviarMensaje(string mensaje, [sovoralib]Social.Country
     pais_origen, [sovoralib]Social.Institution.Reference
    institucion_destino)
{
    newarr [sovoralib]System.Collections.Array
    ldarg 0
    stelem 0
    ldarg 0
```

```
      stelem 1
      ldarg 0
      stelem 2
      stloc 5
      call object MENSAJERIA::cola_mensajes.send(object)
      ret
   }

  }
 }
}

class public AgenteEsencial extends [sovoralib]Social.Agent
{
 attribute private string nombre
 attribute private [sovoralib]Social.Country pais_natal
 attribute private [sovoralib]Social.Role rol
 attribute private [sovoralib]System.Collections.Hashtable<int32, string>
    historia = [sovoralib]System.Collections.Hashtable<int32, string>()
 attribute private [sovoralib]Social.Location ubicacion_actual
 attribute private int32 id
 attribute private [sovoralib]System.Collections.Hashtable<int32, string>
    nombres_de_conocidos


 method public instance void constructor(string nombre, [sovoralib]Social.
    Country pais_natal, int32 id)
 {
    ldarg 0
    call instance [sovoralib]Social.Agent()
    ldarg 1
    statt string AgenteEsencial::nombre
    ldarg 2
    statt [sovoralib]Social.Country AgenteEsencial::pais_natal
    newarr [sovoralib]System.Collections.Array
    ldarg 0
    ldstr "pais_natal.CULTURA.idioma_oficial"
    ldatt object [sovoralib]System.Reflection.resolver(instance, string)
    stelem 0
    ldstr "PERCIBIR.agregar"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 0
    ldarg 3
    statt int32 AgenteEsencial::id
    ret
 }
```

```
method public instance void constructor(string nombre, [sovoralib]Social.
    Language idioma_natal)
{
    ldarg 0
    call instance [sovoralib]Social.Agent()
    ldarg 1
    statt string AgenteEsencial::nombre
    newarr [sovoralib]System.Collections.Array
    ldarg 2
    stelem 0
    ldstr "PERCIBIR.agregar"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 0
    ret
}


@receptor
method private void Join0([sovoralib]Social.Role rol)
{
    ldarg 1
    statt [sovoralib]Social.Role AgenteEsencial::rol
    ret
}

@receptor
method private void Join1(string cond0, int32 hora_actual)
{
    ldarg 1
    ldstr "hora actual"
    call bool [sovoralib]System.String.equality(string, string)
    brtrue L0
    ret
 L0: nop
 L1: nop
    brtrue L2
    br L3
 L2: nop
    newarr [sovoralib]System.Collections.Array
    ldarg 0
    ldstr "ESTE_AGENTE"
    ldatt object [sovoralib]System.Reflection.resolver(instance, string)
    stelem 0
    ldarg 0
    ldstr "ROL.institucion"
```

```
    ldatt object [sovoralib]System.Reflection.resolver(instance, string)
    stelem 1
    ldstr "system.social.actors.disperse"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 0
    br L5
 L3: nop
    newarr [sovoralib]System.Collections.Array
    ldarg 0
    ldstr "ESTE_AGENTE"
    ldatt object [sovoralib]System.Reflection.resolver(instance, string)
    stelem 0
    ldarg 0
    ldstr "ROL.institucion"
    ldatt object [sovoralib]System.Reflection.resolver(instance, string)
    stelem 1
    ldstr "system.social.actors.recopile"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 0
 L5: nop
    ret
}

method private void agregarConocido(string nombre, int32 id)
{
    newarr [sovoralib]System.Collections.Array
    ldarg 0
    stelem 0
    ldarg 0
    stelem 1
    ldstr "INVENTARIO.nombres_de_conocidos.agregar"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 0
    ret
}
method private void desplazarseA([sovoralib]Social.Institution.Reference
    institucion)
{
    newarr [sovoralib]System.Collections.Array
    ldarg 0
    stelem 0
    ldstr "system.ambients.moveFrom(ESTADOS.ubicacion_actual,system.actor.
        serialize(SI_MISMO)).moveTo"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 0
    ret
```

```
 }
 method private int32 votacion(int32 opciones)
 {
    ldstr "system.random"
    call object [sovoralib]System.Reflection.call(string)
    stloc 0
    ldloc 0
    ldloc 0
    ret
 }
}

class public MENSAJERO extends AgenteEsencial
{
 attribute private string mensaje
 attribute private [sovoralib]Social.Country pais_origen
 attribute private [sovoralib]Social.Country pais_destino
 attribute private [sovoralib]Social.Institution.Reference destinatario

 method private void entregarMensaje()
 {
    newarr [sovoralib]System.Collections.Array
    ldatt string MENSAJERO::mensaje
    stelem 0
    ldatt [sovoralib]Social.Country MENSAJERO::pais_origen
    stelem 1
    stloc 0
    ldloc 0
    ldatt [sovoralib]Social.Institution.Reference MENSAJERO::destinatario
    call object [sovoralib]Social.sendMessage(array, instance)
    ldnull
    statt string MENSAJERO::mensaje
    ldnull
    statt [sovoralib]Social.Country MENSAJERO::pais_origen
    ldnull
    statt [sovoralib]Social.Country MENSAJERO::pais_destino
    ldnull
    statt [sovoralib]Social.Institution.Reference MENSAJERO::destinatario
    ret
 }
 method private void desplazarseA([sovoralib]Social.Country pais, [sovoralib]
     Social.Institution.Reference institucion)
 {
    newarr [sovoralib]System.Collections.Array
    ldarg 0
    stelem 0
```

```
    ldarg 0
    stelem 1
    ldstr "system.ambients.moveFrom(ESTADOS.ubicacion,system.actor.serialize(
        SI_MISMO)).moveTo"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 0
    ret
 }
}


class public Universo extends [sovoralib]Social.Universe
{
 attribute private [sovoralib]System.Collections.Hashtable<string, string>
     historia = [sovoralib]System.Collections.Hashtable<string, string>()
 attribute private int32 duracion_dia
 attribute private int32 hora_actual = int32(0)
 attribute private int32 dias_transcurridos = int32(0)
 attribute private int32 avance_tiempo = int32(1000)
 attribute private [sovoralib]Social.Country[] paises
 attribute private [sovoralib]Social.Agent[] agentes

 method public instance void constructor(int32 horas)
 {
    ldarg 0
    call instance [sovoralib]Social.Universe()
    ldarg 1
    statt int32 Universo::duracion_dia
    newarr [sovoralib]System.Collections.Array
    ldstr "avanzar tiempo"
    stelem 0
    stloc 0
    ldloc 0
    ldstr "SI_MISMO"
    ldarg 0
    ldstr "SI_MISMO"
    ldatt object [sovoralib]System.Reflection.resolver(instance, string)
    call object [sovoralib]Social.sendMessage(array, instance)
    ret
 }

 @receptor
 method private void Join0(string cond0)
 {
    ldarg 1
    ldstr "avanzar tiempo"
```

```
  call bool [sovoralib]System.String.equality(string, string)
  brtrue L0
  ret
L0:  nop
  ldatt int32 Universo::hora_actual
  cast float64
  ldint 1
  toflt
  sum
  stloc 0
  ldloc 0
  ldloc 0
  ldatt int32 Universo::duracion_dia
  cast float64
  rem
  stloc 1
  ldloc 1
  statt int32 Universo::hora_actual
L1:  nop
  brtrue L2
  br L3
L2:  nop
  ldatt int32 Universo::dias_transcurridos
  cast float64
  ldint 1
  toflt
  sum
  stloc 0
  ldloc 0
  statt int32 Universo::dias_transcurridos
  br L3
L3:  nop
  newarr [sovoralib]System.Collections.Array
  newarr [sovoralib]System.Collections.Array
  ldatt [sovoralib]Social.Country[] Universo::paises
  stelem 0
  ldatt [sovoralib]Social.Agent[] Universo::agentes
  stelem 1
  stloc 2
  ldloc 2
  stelem 0
  newarr [sovoralib]System.Collections.Array
  ldstr "hora actual"
  stelem 0
  ldatt int32 Universo::hora_actual
  stelem 1
```

```
    stloc 3
    ldloc 3
    stelem 1
    ldstr "system.social.actors.broadcast"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 4
    newarr [sovoralib]System.Collections.Array
    ldatt int32 Universo::avance_tiempo
    stelem 0
    ldstr "system.timeout"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 5
    newarr [sovoralib]System.Collections.Array
    ldstr "avanzar tiempo"
    stelem 0
    stloc 6
    ldloc 6
    ldstr "SI_MISMO"
    ldarg 0
    ldstr "SI_MISMO"
    ldatt object [sovoralib]System.Reflection.resolver(instance, string)
    call object [sovoralib]Social.sendMessage(array, instance)
    ret
}

@receptor
method private [sovoralib]Social.Country Join1(string cond0, string nombre, [
    sovoralib]System.Cluster.Resources recs, [sovoralib]Social.Language
    idioma_oficial)
{
    ldarg 1
    ldstr "construir pais"
    call bool [sovoralib]System.String.equality(string, string)
    brtrue L0
    ret
 L0: nop
    newarr [sovoralib]System.Collections.Array
    ldarg 2
    stelem 0
    ldarg 3
    stelem 1
    ldarg 4
    stelem 2
    ldstr "TLON"
    call instance [sovoralib]Social.instance(array, string)
    stloc 0
```

```
    ldloc 0
    stloc 1
    newarr [sovoralib]System.Collections.Array
    ldloc 1
    stelem 0
    ldstr "ESPACIO_TIEMPO.paises.agregar"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 2
    newarr [sovoralib]System.Collections.Array
    ldstr "EVENTO:"
    ldatt int32 Universo::hora_actual
    call instance [sovoralib]System.String(object)
    call string [sovoralib]System.String.concat(string, string)
    stloc 3
    stelem 0
    ldstr "El dia "
    ldatt int32 Universo::dias_transcurridos
    call instance [sovoralib]System.String(object)
    call string [sovoralib]System.String.concat(string, string)
    stloc 4
    ldloc 4
    ldstr " a las "
    call string [sovoralib]System.String.concat(string, string)
    stloc 5
    ldloc 5
    ldatt int32 Universo::hora_actual
    call instance [sovoralib]System.String(object)
    call string [sovoralib]System.String.concat(string, string)
    stloc 6
    ldloc 6
    ldstr " se creo el pais "
    call string [sovoralib]System.String.concat(string, string)
    stloc 7
    ldloc 7
    ldarg 2
    call string [sovoralib]System.String.concat(string, string)
    stloc 8
    stelem 1
    ldstr "HISTORIA.agregar"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 9
    ldloc 1
    ret
    ret
}
```

```
@receptor
method private [sovoralib]Social.Agent Join2(string cond0, string nombre, [
    sovoralib]Social.Country pais_natal, [sovoralib]Social.Institution.
    Reference institucion)
{
    ldarg 1
    ldstr "crear agente"
    call bool [sovoralib]System.String.equality(string, string)
    brtrue L0
    ret
 L0: nop
    newarr [sovoralib]System.Collections.Array
    ldstr "registrar agente"
    stelem 0
    ldarg 2
    stelem 1
    stloc 0
    ldloc 0
    ldarg 3
    call object [sovoralib]Social.sendMessage(array, instance)
    stloc 1
    ldloc 1
    stloc 2
    newarr [sovoralib]System.Collections.Array
    ldarg 2
    stelem 0
    ldarg 3
    stelem 1
    ldloc 2
    stelem 2
    ldstr "AGENTE_ESENCIAL"
    call instance [sovoralib]Social.instance(array, string)
    stloc 3
    ldloc 3
    stloc 4
    newarr [sovoralib]System.Collections.Array
    ldloc 4
    stelem 0
    ldstr "EPACIOTIEMPO.agentes.agregar"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 5
    newarr [sovoralib]System.Collections.Array
    ldloc 4
    stelem 0
    ldstr "trabajara en"
    stelem 1
```

```
    ldarg 4
    stelem 2
    stloc 6
    ldloc 6
    ldarg 3
    call object [sovoralib]Social.sendMessage(array, instance)
    stloc 7
    ldloc 7
    stloc 8
    newarr [sovoralib]System.Collections.Array
    ldloc 8
    stelem 0
    stloc 9
    ldloc 9
    ldstr "agente"
    ldarg 0
    ldstr "agente"
    ldatt object [sovoralib]System.Reflection.resolver(instance, string)
    call object [sovoralib]Social.sendMessage(array, instance)
    ldloc 4
    ret
    ret
}

@receptor
method private [sovoralib]Social.Agent Join3(string cond0, string nombre, [
    sovoralib]Social.Country pais_natal, [sovoralib]Social.Institution.
    Reference institucion)
{
    ldarg 1
    ldstr "crear agente mensajero"
    call bool [sovoralib]System.String.equality(string, string)
    brtrue L0
    ret
 L0: nop
    newarr [sovoralib]System.Collections.Array
    ldstr "registrar agente"
    stelem 0
    ldarg 2
    stelem 1
    stloc 0
    ldloc 0
    ldarg 3
    call object [sovoralib]Social.sendMessage(array, instance)
    stloc 1
    ldloc 1
```

```
stloc 2
newarr [sovoralib]System.Collections.Array
ldarg 2
stelem 0
ldarg 3
stelem 1
ldloc 2
stelem 2
ldstr "MENSAJERO"
call instance [sovoralib]Social.instance(array, string)
stloc 3
ldloc 3
stloc 4
newarr [sovoralib]System.Collections.Array
ldloc 4
stelem 0
ldstr "ESTADOS.agentes.agregar"
call object [sovoralib]System.Reflection.call(array, string)
stloc 5
newarr [sovoralib]System.Collections.Array
ldloc 4
stelem 0
ldstr "trabajara en"
stelem 1
ldarg 4
stelem 2
stloc 6
ldloc 6
ldarg 3
call object [sovoralib]Social.sendMessage(array, instance)
stloc 7
ldloc 7
stloc 8
newarr [sovoralib]System.Collections.Array
ldloc 8
stelem 0
stloc 9
ldloc 9
ldstr "agente"
ldarg 0
ldstr "agente"
ldatt object [sovoralib]System.Reflection.resolver(instance, string)
call object [sovoralib]Social.sendMessage(array, instance)
ldloc 4
ret
ret
```

```
 }
}

class public Teatro extends [sovoralib]Social.Theater
{
 method private void appEnviarMensaje(string origen, string destino, string
     mensaje)
 {
    newarr [sovoralib]System.Collections.Array
    ldint 24
    stelem 0
    ldstr "UNIVERSO"
    call instance [sovoralib]Social.instance(array, string)
    stloc 3
    ldloc 3
    stloc 4
    ldloc 4
    newarr [sovoralib]System.Collections.Array
    ldstr "construir pais"
    stelem 0
    ldstr "TLON_A"
    stelem 1
    ldarg 0
    ldstr "cluster_A[origen].resources.STANDARD"
    ldatt object [sovoralib]System.Reflection.resolver(instance, string)
    stelem 2
    ldarg 0
    ldstr "IDIOMAS.TLONES_A"
    ldatt object [sovoralib]System.Reflection.resolver(instance, string)
    stelem 3
    stloc 5
    ldloc 5
    ldloc 4
    call object [sovoralib]Social.sendMessage(array, instance)
    stloc 6
    ldloc 6
    stloc 7
    ldloc 7
    newarr [sovoralib]System.Collections.Array
    ldstr "crear agente"
    stelem 0
    ldstr "Agente Alpha"
    stelem 1
    ldloc 7
    stelem 2
    stelem 3
```

```
stloc 8
ldloc 8
ldloc 4
call object [sovoralib]Social.sendMessage(array, instance)
stloc 9
ldloc 9
stloc 10
ldloc 10
newarr [sovoralib]System.Collections.Array
ldstr "crear agente mensajero"
stelem 0
ldstr "Agente beta"
stelem 1
ldloc 7
stelem 2
stelem 3
stloc 11
ldloc 11
ldloc 4
call object [sovoralib]Social.sendMessage(array, instance)
stloc 12
ldloc 12
stloc 13
ldloc 13
newarr [sovoralib]System.Collections.Array
ldstr "construir pais"
stelem 0
ldstr "TLON_B"
stelem 1
ldarg 0
ldstr "cluster_A[destino].resources.STANDARD_OUTPUTS"
ldatt object [sovoralib]System.Reflection.resolver(instance, string)
stelem 2
ldarg 0
ldstr "IDIOMAS.TLONES_B"
ldatt object [sovoralib]System.Reflection.resolver(instance, string)
stelem 3
stloc 14
ldloc 14
ldloc 4
call object [sovoralib]Social.sendMessage(array, instance)
stloc 15
ldloc 15
stloc 16
ldloc 16
newarr [sovoralib]System.Collections.Array
```

```
ldstr "crear agente"
stelem 0
ldstr "Agente Gama"
stelem 1
ldloc 16
stelem 2
stelem 3
stloc 17
ldloc 17
ldloc 4
call object [sovoralib]Social.sendMessage(array, instance)
stloc 18
ldloc 18
stloc 19
ldloc 19
newarr [sovoralib]System.Collections.Array
ldstr "crear agente"
stelem 0
ldstr "Agente Delta"
stelem 1
ldloc 16
stelem 2
stelem 3
stloc 20
ldloc 20
ldloc 4
call object [sovoralib]Social.sendMessage(array, instance)
stloc 21
ldloc 21
stloc 22
ldloc 22
newarr [sovoralib]System.Collections.Array
ldstr "crear agente"
stelem 0
ldstr "Agente Epsilon"
stelem 1
ldloc 16
stelem 2
stelem 3
stloc 23
ldloc 23
ldloc 4
call object [sovoralib]Social.sendMessage(array, instance)
stloc 24
ldloc 24
stloc 25
```

```
    ldloc 25
    newarr [sovoralib]System.Collections.Array
    ldstr "enviar mensaje"
    stelem 0
    ldloc 7
    stelem 1
    stelem 2
    ldarg 0
    stelem 3
    stloc 0
    ldloc 0
    ldstr "tlon_A"
    ldarg 0
    ldstr "tlon_A"
    ldatt object [sovoralib]System.Reflection.resolver(instance, string)
    call object [sovoralib]Social.sendMessage(array, instance)
    ldstr "universo.destruir"
    call object [sovoralib]System.Reflection.call(string)
    stloc 0
    ret
}

@entrypoint
method public static void Main()
{
    newarr [sovoralib]System.Collections.Array
    ldstr "CLUSTER TLON_A"
    stelem 0
    ldstr "system.net.ad_hoc.cluster"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 0
    ldloc 0
    stloc 1
    newarr [sovoralib]System.Collections.Array
    ldstr "n1"
    stelem 0
    newarr [sovoralib]System.Collections.Array
    ldstr "NODO_1"
    stelem 0
    ldstr "system.net.host"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 2
    ldloc 2
    stelem 1
    ldstr "cluster_A.addNode"
    call object [sovoralib]System.Reflection.call(array, string)
```

```
stloc 3
newarr [sovoralib]System.Collections.Array
ldstr "n2"
stelem 0
newarr [sovoralib]System.Collections.Array
ldstr "NODO_2"
stelem 0
ldstr "system.net.host"
call object [sovoralib]System.Reflection.call(array, string)
stloc 4
ldloc 4
stelem 1
ldstr "cluster_A.addNode"
call object [sovoralib]System.Reflection.call(array, string)
stloc 5
ldstr "n1"
stloc 6
newarr [sovoralib]System.Collections.Array
ldstr "Ingrese nodo destino: "
stelem 0
ldstr "cluster_A[origen].console.println"
call object [sovoralib]System.Reflection.call(array, string)
stloc 7
ldstr "cluster_A[origen].console.readln"
call object [sovoralib]System.Reflection.call(string)
stloc 8
ldloc 8
stloc 9
newarr [sovoralib]System.Collections.Array
ldstr "Ingrese mensaje a enviar: "
stelem 0
ldstr "cluster_A[origen].console.println"
call object [sovoralib]System.Reflection.call(array, string)
stloc 10
ldstr "cluster_A[origen].console.readln"
call object [sovoralib]System.Reflection.call(string)
stloc 11
ldloc 11
stloc 12
newarr [sovoralib]System.Collections.Array
ldloc 6
stelem 0
ldloc 9
stelem 1
ldloc 12
stelem 2
```

```
    ldstr "appEnviarMensaje"
    call object [sovoralib]System.Reflection.call(array, string)
    stloc 13
    ret
 }
}
```

8. Conclusiones y recomendaciones

8.1. Conclusiones

1. La propuesta del lenguaje de programación social-inspirado desarrollado en este trabajo es un paso más para continuar con la creación de un lenguaje para un sistema de computación distribuida basado en redes ad hoc, como lo propone el grupo TLÖN.

2. El conocimiento de diferentes modelos de computación como los cálculos π, join, ambientes y el modelo de actores, permiten introducir patrones sintácticos y semánticos que otorgan beneficios a la hora de diseñar un lenguaje de programación concurrente, distribuido y móvil.

3. Un compilador, el que se encarga de traducir un código fuente en un código destino, presenta diferentes fases, las cuales van estructurando y transformando la entrada, la cual va atravesando estas fases, siendo transformada en representaciones intermedias. Esto permite que existan diferentes formas de llegar a generar código máquina, ya sea a través de diferentes lenguajes intermedios, o simplemente una traducción semidirecta; a lo que finalmente se le pueden aplicar diferentes formas de optimización.

4. El paso de mensajes del modelo de actores, al haberse implementado en la sintaxis de DiUNisio 2.0, aportó la capacidad de comunicación asíncrona entre las entidades del modelo social. Lo que permitió que realizaran múltiples tareas sin tener que esperar por una respuesta inmediata. Asimismo, la sintaxis del cálculo π para determinar dos procesos concurrentes (con el operador $|$), o también, la ejecución no determinística de estos (con el operador $+$).

5. La utilización del diseño basado en analogías fue útil a la hora de proponer una forma de construir aplicaciones bajo el enfoque de computación social-inspirada en la manera en que un director de obras de teatro construye composiciones teatrales, desde la concepción de las mismas hasta la puesta en escena frente a un auditorio. En el caso de la elaboración de aplicaciones de software social-inspiradas, el desarrollador se asemeja a un director de obra quien gestiona los recursos computacionales para dar solución al problema planteado, desde su origen hasta la obtención de la solución mediante la

ejecución sobre el sistema de cómputo.

8.2. Recomendaciones

Aunque dentro del alcance de este proyecto no se pudo profundizar la idea de la combinación de computadores virtuales interconectado, obtienen recursos un mismo nodo, igualmente se considera importante para futuras versiones.

Para el trabajo futuro de este prototipo de traductor, se sugieren:

- La creación de un Entorno de Desarrollo que facilite el desarrollo de aplicaciones en este lenguaje.
- Continuar con las siguientes fases del Compilador.
- La implementación de complementos en Seguridad, Disponibilidad y Tolerancia a Fallos.

Por otro lado, para la optimización del código intermedio generado por este traductor, se podrían:

- Eliminar redundancias, simplificar el almacenamiento y el cargue inmediato de variables locales.
- Declarar la reserva de memoria.
- Resolver los llamados dinámicos por los microservicios de System.Reflection.
- Reenumerar etiquetas.
- Resolver anotaciones.
- Eliminar retornos redundantes.
- Reducir repeticiones load store.
- Conversión de datos más declarativa.

A. Manual de usuario de DiUNisio 2.0

Diunisio 2.0: Manual de usuario

Andrés Felipe De Orcajo Vélez
Jorge Eduardo Ortiz Triviño

Universidad Nacional de Colombia
Facultad de Ingeniería, Departamento de Ingeniería de Sistemas e Industrial
Bogotá, Colombia
2020

A.1. Introducción

A.1.1. Programación social-inspirada

La programación social-inspirada, para el sistema TLÖN, es un paradigma que describe la programación en términos de interacción entre entidades mediante el paso de mensajes.

Se basa en dos conceptos:

- **Actores:** entidades comunicativas, envían mensajes unas a otras.
- **Ambientes:** espacios donde un actor se puede mover e interactuar.

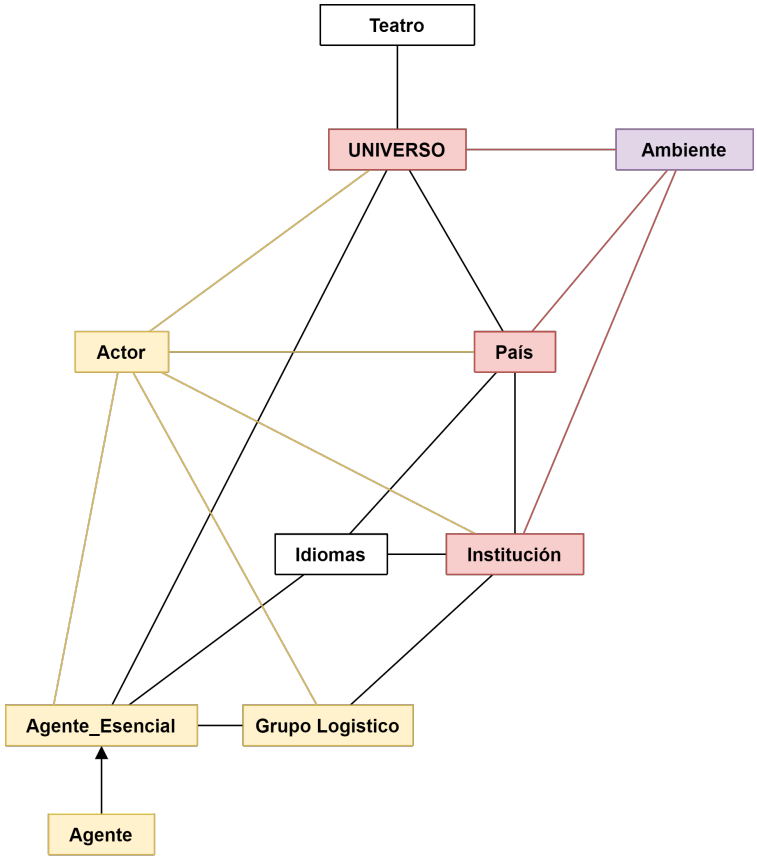

Figura A-1.: Modelo social de DiUNisio 2.0.

A.1.2. Descripción general

DiUNisio 2.0 es un lenguaje de programación social-inspirado en español desarrollado con la librería ANTLR; que permite construir algoritmos basado en la interacción de diferentes entidades que conforman un modelo social.

A.2. Descripción del modelo de programación

El algoritmo es programado usando caracteres en formato de texto plano usando los caracteres del conjunto ASCII. Los caracteres válidos en el algoritmo son los siguientes:

- caracteres alfabéticos:

  ```
  A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
  a b c d e f g h i j k l m n o p q r s t u v w x y z _
  ```

- caracteres numéricos:

  ```
  0 1 2 3 4 5 6 7 8 9
  ```

- caracteres especiales:

  ```
  ! & ( ) * , - . : ; < = > / % + [ ] ^ { | }
  ```

- caracteres de espacios en blanco:

  ```
  espacio nuevaLinea tabulación
  ```

Dentro de los literales y comentarios cualquier caracter ASCII (excepto caracteres de control) son válidos.
Los caracteres de espacios en blanco no son significantes. Pueden ser usados libremente entre unidades léxicas para mejorar la legibilidad del algoritmo. También son usadas para separar unidades léxicas unas de otras si no hay otra forma de hacerlo.
Sintácticamente el algoritmo es una secuencia de unidades léxicas en las siguientes categorías:

- nombres de variables;
- números;
- cadenas de caracteres;
- palabras reservadas;
- delimitadores y operadores;
- comentarios.

Las unidades léxicas del lenguaje son presentadas a continuación.

A.2.1. Nombres de variables

Un nombre de variable consiste en caracteres alfanuméricos, el primero debe ser alfabético. Todos los nombres de variables son distintos (distingue mayúsculas y minúsculas).

Ejemplos:

```
variable123
Estatura_promedio
_VALOR
```

Los nombres de variables son usados para identificar variables (números, parámetros, conjuntos).
Todos los nombres de variables deben ser únicos dentro del alcance, donde sean válidos.

A.2.2. Números

Un número tiene la forma *nnEsxx*, donde *nn* es un número con punto decimal opcional, *s* es el signo + o -, *xx* es un exponente decimal. La letra *E* también puede ser *e*.

Ejemplos:

```
123
3.14159
56.E+5
.78
123.456e-7
```

Los números son usados para representar cantidades numéricas.

A.2.3. Números complejos

Un número complejo tiene la forma *nnEsxxEsnnEsxxi*, donde *nn* es un número con punto decimal opcional, *s* puede ser el signo - o + (excepto cuando es el signo del exponente), *xx* es un exponente decimal. La letra *E* también puede ser *e*.

Ejemplos:

```
123+i
3.14159-2i
56.E5+2.2i
.78-.9E5i
123.456e-7+i
```

Los números complejos son usados para representar cantidades numéricas en el plano complejo.

A.2.4. Cadena de caracteres

Una cadena de caracteres es una secuencia de caracteres arbitrarios encerrados por comillas dobles.

Ejemplos:

```
"Esta es una cadena de caracteres"
"Hola "
```

Las cadenas de caracteres son usadas para representar cantidades simbólicas.

A.2.5. Palabras Reservadas

Una palabra reservada es una secuencia de caracteres alfabéticos y posiblemente un caracter especial.
Todas las palabras reservadas corresponden a una categoría: palabras reservadas, las cuales pueden ser usadas como nombres de variables, y palabras no reservadas, las cuales son reconocidas por el contexto y por eso pueden ser usadas como nombres de variables.
Las palabras reservadas son las siguientes:

AGENTE	AGENTE_ESENCIAL	CONOCIMIENTO	CONSTITUCION_DEL_PAIS
CULTURA	DECIDIR	DIALOGO	DISPONIBILIDAD
ELENCO	ESCENOGRAFIA	ESPACIO_TIEMPO	ESTADOS
GRUPO_LOGISTICO	HISTORIA	IDENTIDAD	IDIOMAS
IDIOMA	INSTANCIADOR	INSTITUCION	INVENTARIO
LEYES	MIEMBROS	MUTIS	OBRAS
ORGANIGRAMA	PARTICIPANTES	PERCIBIR	PUESTA_EN_ESCENA
REDIRIGIR_MENSAJES	SERVICIOS	TEATRO	RECURSOS
REFERENCIA	UBICACION	UNIVERSO	booleano
caso	cluster	def	defecto
entonces	falso	decimal	funcion
hacer	entero	mientras	nulo
PAIS	para	para_cada	retornar
ROL	romper	seleccionar	si
si_no	cadena	usando	vacio
verdadero			

Las palabras no reservadas son presentadas en las siguientes secciones.
Todas las palabras reservadas tienen un significado específico, el cual será explicado en las construcciones sintácticas correspondientes, donde las palabras reservadas son usadas.

A.2.6. Delimitadores y operadores

Un delimitador es o un caracter especial simple o una secuencia de dos caracteres especiales como se muestran a continuación:

```
-> <- ? ==> <== || && & == != > < >= <=
++ + - * / % ^ ! ' ; = ( ) { } [] [ ] , :
```

Si el delimitador consiste de dos caracteres, no debe haber espacio entre los caracteres.
Todos los delimitadores tienen un significado específico, el cual será explicado en las correspondientes construcciones sintácticas, donde los delimitadores son usados.

A.2.7. Comentarios

Para propósitos de documentación del algoritmo puede contener comentarios, que tienen dos formas. La primera forma es la de comentarios de línea simple, que empieza con el caracter `#` y se extiende hasta el fin de línea. La segunda forma es la de comentario multilínea, que es una secuencia de caracteres encerrados dentro de `/*` y `*/`.

Ejemplos:

```
i = 0; #Esto es un comentario
/* Esto es otro comentario */
```

Los comentarios son ignorados por el compilador y pueden aparecer en cualquier lado del algoritmo, donde los espacios en blanco están permitidos.

A.3. Expresiones

Una expresión es una regla para calcular un valor. En el algoritmo las expresiones son usadas como constituyentes de ciertas sentencias.
En las expresiones generales consiste en operandos y operadores.
Dependiendo del tipo del valor resultante todas las expresiones caen dentro de una de las siguientes categorías:

- expresiones numéricas;
- expresiones simbólicas;
- expresiones lógicas.

A.3.1. Expresiones numéricas

Una expresión numéricas es una regla de computación para un valor numérico simple representado como un número de punto flotante.
La expresión numérica primaria puede ser un número, un parámetro, una referencia a una función predefinida, u otra expresión numérica encerrada en paréntesis.
Ejemplos:

```
1.23
j
tiempo
mat[1][2]
abs(mat[1][2])
(mat[1][2] * .5 * j)
```

Más expresiones numéricas conteniendo dos o más expresiones numéricas primarias pueden ser construidas usando ciertos operadores aritméticos.
Ejemplos:

```
j+1
2 * mat[j][i-1] - b[1][2]
```

Expresiones con paréntesis

Cualquier expresión numérica puede ser encerrada en paréntesis que sintácticamente lo hacen una expresión numérica primaria.
Los paréntesis pueden ser usados en expresiones numéricas, como en el álgebra, para especificar el orden deseado del cual las operaciones se van a desarrollar. Donde los paréntesis son usados, la expresión dentro de los paréntesis es evaluada antes de que el valor resultante sea usado.
El valor resultante de las expresiones con paréntesis es la misma que el valor de la expresión encerrada dentro de los paréntesis.

Operadores aritméticos

En Diunisio existen los siguientes operadores aritméticos, los cuales pueden ser usados en expresiones numéricas:

Operación	Descripción
`-n`	Operador unario negativo
`+n`	Operador unario positivo
`n + m`	Suma
`n - m`	Resta
`n * m`	Producto
`n / m`	División
`n % m`	Módulo
`n ^ m`	Potenciación

donde `n` y `m` son expresiones numéricas.
Si la expresión incluye más que un operador aritmético, todos los operadores son aplicados de izquierda a derecha de acuerdo con la jerarquía de operadores (mirar a continuación) con la única excepción de que los operadores de exponenciación son aplicados de derecha a izquierda.
El valor resultante de la expresión, la cual contiene operadores aritméticos, es el resultado de aplicar los operadores a sus operandos.

Jerarquía de operaciones

La siguiente lista muestra la jerarquía de los operadores en expresiones numéricas:

Operación	Jerarquía
Evaluación de funciones	1º
Exponenciación	2º
Suma y resta unarias	3º
Multiplicación y división	4º
Suma y resta	5º

La jerarquía es usada para determinar cuál de dos operaciones consecutivas es aplicada primero. Si el primero operador es mayor o igual al segundo, el primer operador es aplicado. Si no, el segundo operador es comparado con el tercero, etc. Cuando el fin de la expresión es alcanzado, todas las operaciones restantes son ejecutadas en el orden inverso.

A.3.2. Expresiones simbólicas

Una expresión simbólica es una regla de computación de un solo valor simbólico representado como una cadena de caracteres.
La expresión simbólica primaria puede ser una cadena de caracteres, un parámetro, una referencia de función predefinida, u otra expresión simbólica encerrada entre paréntesis.
También es permitido usar una expresión numérica como función simbólica primaria, caso en el cual el valor resultante de una expresión numérica es automáticamente convertido a un tipo simbólico.

Ejemplos:

```
"Mayo 2017"
j
subcadena(1,5)
```

Más expresiones simbólicas conteniendo dos o más expresiones simbólicas primarias puede ser construida usando el operador de concatenación.

Ejemplos:

```
"abc[" + i + "," + j + "]"
"desde " + ciudad[i] + " a " + ciudad[j]
```

Los principios de evaluación de expresiones simbólicas son complementariamente análogas a las dadas para expresiones numéricas (mirar arriba).

Operadores simbólicos

Actualmente en DiUNisio 2.0 existe un sólo operador simbólico:

Operación	Jerarquía
`s + t`	Concatenación

donde `s` y `t` son expresiones simbólicas. Éste operador significa la concatenación de sus dos operandos simbólicos, los cuales son cadenas de caracteres.

Jerarquía de operadores

La siguiente lista muestra la jerarquía de operadores en las expresiones simbólicas:

Operador	Jerarquía
Evaluación de operaciones numéricas	1º-6º
Concatenación	7º

Esta jerarquía tiene el mismo significado como se explicó anteriormente para expresiones numéricas.

A.3.3. Expresiones de conjuntos

Una expresión de conjuntos es una regla de cálculo de un conjunto elemental, por ejemplo una colección de n-tuplas, donde los componentes de las n-tuplas son cantidades numéricas y simbólicas.
La expresión de conjuntos básica puede ser una definición de conjunto, u otra expresión de conjuntos encerrada entre paréntesis.

Ejemplos:

```
{1,2}
{{1,2},{3,4}}
{{{1},{2,3}}}
```

Expresiones con paréntesis

Cualquier expresión de conjuntos puede estar encerrada entre paréntesis que sintácticamente lo hacen una expresión de conjunto básica.

Los paréntesis pueden ser usados en expresiones de conjuntos, como en el álgebra, para especificar el orden deseado en el cual las operaciones van a ser aplicadas. Donde los paréntesis son usados, la expresión dentro de los paréntesis es evaluada antes de que el valor resultante es usado.
El valor resultante de la expresión con paréntesis es la misma que la evaluada de la expresión encerrada por los paréntesis.

Operaciones de conjuntos

En Diunisio existen las siguientes operaciones, las cuales pueden ser usadas en expresiones de conjuntos:

Operación	Descripción
`x + y`	Suma
`x - y`	Resta
`x * y`	Producto
`x / y`	División

donde `x` e `y` son expresiones de conjuntos, los cuales deberían definir conjuntos de dimensión idéntica.
Si la expresión incluye más de un conjunto de operadores, todos los operadores son aplicados de izquierda a derecha de acuerdo con la jerarquía de operadores (mirar a continuación).
El valor resultante de la expresión, el cual contiene un conjunto de operadores, es el resultado de aplicar los operadores a sus operandos.
La dimensión del conjunto resultante, por ejemplo, la dimensión de n-tuplas, de las cuales el conjunto resultante se compone, es el mismo de la dimensión de los operandos.
Nota: Si las dimensiones de las matrices no coinciden, el operador de suma une los conjuntos. El operador de producto realiza producto cruz cuando son matrices y producto punto cuando son vectores.

Jerarquía de los operadores

La siguiente lista muestra la jerarquía de los operadores en expresiones de conjuntos:

Operación	Jerarquía
Evaluación de operaciones numéricas	1º-6º
Evaluación de operaciones simbólicas	7º-9º
Producto y división de conjuntos	10º
Suma y resta de conjuntos	11º

La jerarquía tiene el mismo significado de lo explicado anteriormente para las expresiones numéricas.

A.3.4. Expresiones lógicas

Una expresión lógica es una regla de cálculo de un único valor lógico, el cual puede o ser verdadero o falso.
La expresión lógica más simple puede ser una expresión numérica, una expresión relacional, u otra expresión lógica encerrada entre paréntesis.

Ejemplos:

```
i+1
a[1] < 8
a[2] != 5
falso && !verdadero
```

Expresiones lógicas más generales conteniendo dos o más expresiones lógicas simples pueden ser construidas usando ciertos operadores lógicos.

Ejemplos:

```
!(a[i] < 8 || b[j] >= 9) && i < 9
```

Expresiones numéricas

El valor resultante de la expresión lógica simple, la cual es una expresión numérica, es verdadero, si el valor resultante de la expresión no es cero. En otro caso el valor resultante de la expresión numérica es falsa.

Operadores relacionales

En Diunisio existen los siguientes operadores lógicos, los cuales pueden ser usados en expresiones lógicas:

Operación	Descripción
`x < y`	Menor que
`x <= y`	Menor o igual que
`x == y`	Igual
`x >= y`	Mayor o igual que
`x > y`	Mayor que
`x != y`	Distinto de

donde `x` e `y` son expresiones numéricas o simbólicas.
Como los operadores relacionales listados anteriormente tienen su significado matemático convencional. El valor resultante es verdadero, si la relación correspondiente es satisfecha por sus operandos, en otro caso falso. (Notar que los valores simbólicos son ordenados lexicográficamente, y cualquier valor numérico precede cualquier valor simbólico.

Expresiones con paréntesis

Cualquier expresión lógica puede ser encerrada en paréntesis que sintácticamente lo hacen una expresión lógica simple.
Los paréntesis pueden ser usados en expresiones lógicas, como en el álgebra para especificar el orden deseado en el cual los operadores se aplicarán. Donde los paréntesis son usados, la expresión dentro de los paréntesis es evaluada antes de que el valor resultante es usado.
El valor resultante de la expresión con paréntesis es la misma que el valor de la expresión encerrada dentro de los paréntesis.

Operadores lógicos

En Diunisio existen los siguientes operadores lógicos, los cuales pueden ser usados en expresiones lógicas:

Operación	Descripción
`!x`	Negación
`x && y`	Conjunción
`x || y`	Disjunción

donde `x` e `y` son expresiones lógicas.
Si la expresión incluye más de un operador lógico, todos los operadores son aplicados de izquierda a derecha de acuerdo con la jerarquía de los operadores (mirar a continuación).

El valor resultante de la operación, el cual contiene operadores lógicos, es el resultado de aplicar los operadores a sus operandos.

Jerarquía de operaciones

La siguiente lista muestra la jerarquía de los operadores en las expresiones lógicas:

Operación	Jerarquía
Evaluación de operaciones numéricas	1º-6º
Evaluación de operaciones simbólicas	7º
Evaluación de operaciones de conjuntos	8º
Operaciones relacionales	9º
Negación	10º
Conjunción	11º
Disyunción	12º

La jerarquía tiene el mismo significado que fue explicado anteriormente para expresiones numéricas.

A.4. Sentencias

Las sentencias son unidades básicas en la descripción del algoritmo. En Diunisio todas las sentencias están divididas en dos categorías: sentencias de declaración y sentencias funcionales.
Las sentencias de declaración (sentencia de conjunto, sentencia de parámetros, sentencia de variable) son usados para declarar los objetos del algoritmo de ciertos tipos y definir ciertas propiedades de esos objetos.

A.4.1. Declaración de variables

```
def nombre:tipo_dato = valor;
```

`nombre` es un nombre de variable;
`tipo_dato` es el tipo de dato.
`valor` puede ser una expresión numérica, lógica, de conjunto, simbólica.

Ejemplos:

```
def a:entero = 1;
def b:decimal = i;
def c:cadena = "Cantidad";
def e:booleano = verdadero;
```

A.4.2. Sentencia si

```
si ( condición ) entonces {
        sentencias
}
si_no si ( condición ) {
        sentencias
}
...
si_no si ( condición ) {
        sentencias
}
si_no {
        sentencias
}
```

`condición` es la condición que se evaluará para determinar si se elige la opción o se continúan recorriendo las demás condiciones;
`si_no si ... si_no si` es la serie de condiciones opcionales alternativas al primer `si`;
`si_no` es la condición por defecto si ninguna de las demás se llegara a cumplir.
Nota: los paréntesis son opcionales junto con la palabra entonces.

Ejemplos:

```
si a < 0 { imprimirPantalla("negativo"); }
si b { a = a * 5; } si_no si (a == 1) entonces { b = b / 5; }
si falso entonces { } si_no si verdadero { }
```

A.4.3. Sentencia seleccionar

```
seleccionar variable {
    caso valor: sentencias romper;
    ...
    caso valor: sentencias romper;
```

```
    defecto: sentencias
}
```

`variable` es un valor simbólico o numérico que se tomará como referencia para elegir el `caso` que coincida en su valor con ésta variable;
`valor` es el valor que se comparará con la `variable` para determinar si el caso es elegido o no;
`caso ... caso` permite añadir cualquier cantidad de casos, que pueden no definirse pero el caso por `defecto` siempre estará presente.

Ejemplos:

```
seleccionar a {
    caso 1: imprimirPantalla("solo hay uno"); romper;
    caso 2: imprimirPantalla("quedan dos"); romper;
    defecto: imprimirPantalla("no alcanza");
}
```

A.4.4. Sentencia para

```
para ( var, ..., var; condición; oper, ..., oper) {
        sentencias
}
```

`var, ..., var` es un listado de declaración de variables;
`condición` es la condición que se evaluará por cada iteración del ciclo;
`oper, ..., oper` es un listado de operaciones que modifiquen variables para terminar el ciclo.
Nota: los paréntesis son opcionales.

Ejemplos:

```
para a = -1; a < 5; a=a+1 {
        imprimirPantalla(a);
}
para (i = 0, j = 0; i < 5; i=i+1, j = i-1) {
        imprimirPantalla(mat[i][j]);
}
```

A.4.5. Sentencia mientras

```
mientras ( condicion ) {
        sentencias
}
```

condición es la condición que se evaluará por cada iteración del ciclo.
Nota: los paréntesis son opcionales.

Ejemplos:

```
mientras a>2 {
        a = a-1;
        imprimirPantalla(sen(a));
}
mientras a + b < c {
        imprimirPantalla("sumando a y b");
        c = c - 1;
}
```

A.4.6. Sentencia hacer_mientras

```
hacer {
        sentencias
} mientras ( condición )
```

condición es la condición que se evaluará al finalizar cada iteración para determinar la continuidad del ciclo.
Nota: los paréntesis son opcionales.

Ejemplos:

```
hacer {
        a = a+1;
        imprimirPantalla(a);
} mientras a < 5
```

A.4.7. Procedimiento

```
nombre ( param, ..., param ) {
        sentencias
}
```

`nombre` determina el nombre con el que se almacenará el procedimiento;
`param, ..., param` es una serie de parámetros que recibirá el procedimiento al momento de ejecutarse.

Ejemplos:

```
concat (nombre:cadena) {
        imprimirPantalla("Hola " + nombre);
}
```

A.4.8. Función

```
tipo nombre ( param, ..., param ) {
        sentencias
}
```

`nombre` determina el nombre con el que se almacenará la función;
`param, ..., param` es una serie de parámetros que recibirá la función al momento de ejecutarse.

Ejemplos:

```
#Definición recursiva del factorial de 'y'
entero g(y){
        res = 1;
        si y < 2 {
                res = 1;
        }
        si_no {
                res = y * g(y-1);
        }
        retornar res;
}
```

A.5. Computación social

A continuación se presenta el uso de operadores sociales, notación de procesos en paralelo, ejecuciones no determinísticas, y paso de mensajes entre actores.

Elementos sociales: El operador paso de mensaje

El operador flecha derecha representa el paso de un mensaje a un actor.

```
["enviar mensaje", tlon_A, tlon_B.1, mensaje] -> tlon_A;
```

Elementos sociales: El operador instanciar

El operador flecha izquierda representa el instanciamiento de un Actor.

```
UNIVERSO <- [24];
```

Elementos sociales: El patrón de recepción de mensajes

A continuación se muestra un ejemplo de lo que es un patrón join para recepción de mensajes.

```
mensaje:cadena & remitente:PAIS {
        ...
}
```

Elementos sociales: Ejecución en paralelo

A continuación se muestra un ejemplo:

```
[
    || agregarConocido(nombre, id);
    || ["hello, my name is ", SI_MISMO, IDENTIDAD.nombre, INVENTARIO.id] -> SI_MISMO;
]
```

Elementos sociales: Acción no determinística

A continuación se muestra un ejemplo:

```
[
    ++ RECURSOS.console.println("Mensaje de " + remitente + ": " + mensaje);
    ++ RECURSOS.console.println(mensaje);
] usando votacion(2);
```

Elementos sociales: Lectura y escritura de un canal de comunicación

A continuación se muestra un ejemplo de lectura y escritura desde y hacia un canal.

```
==> def nombre:cadena;
cola_ids* <== ESTADOS.contador;
```

A.6. Requerimientos

A continuación se explica el procedimiento de instalación junto con los requisitos del sistema para poder ejecutar el algoritmo correctamente.

A.6.1. Software

Para la correcta ejecución de los algoritmos sobre el lenguaje DiUNisio 2.0, es necesaria la instalación del siguiente software, junto con un sistema operativo, ya sea Windows o alguna distribución de Linux:

- Java Development Kit (JDK) 7.
- Librería AntLR v4.

A.6.2. Hardware

Los requerimientos de hardware están dados por el sistema operativo y por el entorno de desarrollo de Java.

A.7. Instalación

A.7.1. Windows

Instalación del JDK

Descargar el JDK de la página de `http://www.oracle.com/technetwork/es/java/javase/downloads/index.html`, luego se instala. Se debe editar las variables del entorno de Windows, añadiendo a la variable PATH el directorio de instalación del JDK:

```
PATH= ... ;C:\Program Files\Java\jdk1.8.0_261\bin
```

Descarga de ANTLR

Descargar la librería de ANTLR desde `http://www.antlr.org/`

Configuración adicional

Se debe comprobar la correcta instalación del entorno de desarrollo de Java ejecutando la siguiente instrucción en la terminal:

```
javac -version
```

Creamos un directorio en la Unidad de Disco Local C: llamado *ANTLR*, y dentro:

1. Creamos un archivo *antlr4.bat* y en su contenido ponemos:

```
java org.antlr.v4.Tool %*
```

2. Creamos un archivo *grun.bat* y en su contenido ponemos:

```
java org.antlr.v4.gui.TestRig %*
```

3. Copiamos ahí la librería de ANTLR previamente descargada.

Añadimos la siguiente variable del entorno:

```
CLASSPATH = .;C:\ANTLR\antlr-4.8-complete.jar
```

A.7.2. Linux

Instalación del JDK

Para obtener el kit de desarrollo de Java, debemos ejecutar desde el terminal el siguiente comando como superusuario:

```
sudo apt-get install openjdk-7-jdk
```

Descarga de ANTLR

La librería de ANTLR se puede descargar de la página: `http://www.antlr.org/`

Configuración adicional

Para que el JDK al hacer la ejecución del intérprete, pueda completar correctamente el proceso de compilación, deben definirse las variables del entorno: *CLASSPATH*; pero también para facilitar la inserción de parámetros de la ejecución de ANTLR, se deben crear también otras dos variables: *antrl4* y *grun*. Las cuales se deben ejecutar directamente en la terminal. La estructura de la variable *CLASSPATH* es:

```
export CLASSPATH=".:/directorio/antlr-4.8-complete.jar:$CLASSPATH"
```

Un ejemplo de *CLASSPATH*:

```
export CLASSPATH=".:/home/usuario/Escritorio/antlr-4.8-complete.jar:$CLASSPATH"
```

Luego se añaden estos alias, también desde el terminal:

```
alias antlr4='java org.antlr.v4.Tool'
alias grun='java org.antlr.v4.gui.TestRig'
```

Rutas del entorno para el compilador de Java

```
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk
export PATH=$PATH:/usr/lib/jvm/java-7-openjdk/bin
```

Se debe comprobar la correcta instalación del entorno de desarrollo de Java ejecutando la siguiente instrucción en la terminal:

```
javac -version
```

A.8. Ejecución

Para poder hacer uso de la herramienta, es necesario el uso de una consola (o un terminal) del sistema operativo para ejecutar la clase *Main.java* contenida en el directorio de DiUNisio 2.0. Primero se deben compilar todas las clases del directorio, haciendo uso del JDK instalado previamente:

```
javac *.java
```

A continuación, situados en el mismo directorio, ejecutamos la clase compilada Main así:

```
java Main nombreArchivo
```

La consola (o terminal) quedará a la espera para recibir la introducción del algoritmo elaborado con la sintaxis de DiUNisio 2.0, por lo tanto debemos, o introducir el programa manualmente, o pegarlo directamente:
Cuando se haya introducido completamente el algoritmo, debemos presionar *ENTER* y luego *CTRL+D* (o *CTRL+Z* desde Windows) para informarle a la consola (o terminal) de que hemos terminado la introducción del contenido del programa, finalmente se observarán los resultados de la ejecución del algoritmo introducido previamente.

Nota: Es posible ejecutar un código previamente almacenado en un archivo, ejecutando el siguiente comando:

```
java Main archivo.txt [parámetros del algoritmo]
```

B. Manual técnico de DiUNisio 2.0

Diunisio 2.0: Manual técnico

Andrés Felipe De Orcajo Vélez
Jorge Eduardo Ortiz Triviño

Universidad Nacional de Colombia
Facultad de Ingeniería, Departamento de Ingeniería de Sistemas e Industrial
Bogotá, Colombia
2020

B.1. ANTLR

El lenguaje fue desarrollado con la ayuda de la librería ANTLR[1], definiendo el léxico y la gramática en un archivo con extensión *.g4*. Luego se generaron los analizadores, que quedaban definidos como clases de Java. Se sobrescribió la funcionalidad del Visitor, para definir las operaciones semánticas del lenguaje. Finalmente se creó una clase Main que es la que permite generar código intermedio para código escrito en este lenguaje que se recibe como una entrada.

B.2. Gramática de DiUNisio 2.0 (en notación E-BNF)

```
estructura
 : universo? idiomas? constitucion* agente_esencial? agentes* teatro?
 ;

universo
 : UNIVERSO DOSPUNTOS LLAVEIZ historia? espacio_tiempo? instanciador? leyes? LLAVEDE
 ;

espacio_tiempo
 : ESPACIOTIEMPO DOSPUNTOS bloque_defs
 ;

leyes
 : LEYES DOSPUNTOS bloque_join
 ;

idiomas
 : IDIOMAS DOSPUNTOS LLAVEIZ def_idioma* LLAVEDE
 ;

constitucion
 : CONSTITUCION identificador DOSPUNTOS LLAVEIZ conf_pais
 LLAVEDE
 ;

def_idioma
 : identificador DOSPUNTOS bloque_join
```

```
 ;

conf_pais
 : identidad? historia? estados? instanciador? cultura? organigrama
 redirigir_mensajes
 ;

organigrama
 : ORGANIGRAMA DOSPUNTOS LLAVEIZ sec_instituciones+ LLAVEDE
 ;

sec_instituciones
 : REFERENCIA identificador DOSPUNTOS LLAVEIZ identidad?
 historia? estados? canales? disponibilidad? miembros? percibir?
 grupos* conocimiento? servicios? sec_instituciones? LLAVEDE
 ;

identidad
 : IDENTIDAD DOSPUNTOS bloque_defs
 ;

canales
 : CANALES DOSPUNTOS ANGIZ lista_canales ANGDE PCOMA
 ;

miembros
 : MIEMBROS DOSPUNTOS bloque_miembros
 ;

bloque_miembros
 : LLAVEIZ (PUESTO bloque_canales)* LLAVEDE
 ;

bloque_canales
 : LLAVEIZ (CANAL CONSUMIBLE bloque_canal)? LLAVEDE
 ;

grupos
```

```
 : GRUPO_LOGISTICO identificador DOSPUNTOS LLAVEIZ bloque_grupo LLAVEDE
 ;

bloque_grupo
 : identidad? estados? participantes? decidir?
 ;

participantes
 : PARTICIPANTES ASIGNAR ANGIZ lista_valores ANGDE PCOMA
 ;

decidir
 : DECIDIR DOSPUNTOS bloque_decidir
 ;

bloque_decidir
 : LLAVEIZ (sec_join bloque_decisiones)* LLAVEDE
 ;

disponibilidad
 : DISPONIBILIDAD ASIGNAR ANGIZ lista_horarios  ANGDE PCOMA
 ;

lista_horarios
 : HORARIO (COMA HORARIO)*
 ;

conocimiento
 : CONOCIMIENTO DOSPUNTOS bloque_funs
 ;

servicios
 : SERVICIOS DOSPUNTOS bloque_servs
 ;

cultura
 : CULTURA DOSPUNTOS bloque_defs
 ;
```

```
historia
 : HISTORIA DOSPUNTOS LLAVEIZ dupla_tipos LLAVEDE PCOMA
 ;

redirigir_mensajes
 : REDIRIGIR_MENSAJES DOSPUNTOS bloque_join
 ;

agente_esencial
 : AGENTE_ESENCIAL DOSPUNTOS LLAVEIZ conf_agente LLAVEDE
 ;

agentes
 : AGENTE identificador DOSPUNTOS LLAVEIZ conf_agente LLAVEDE
 ;

conf_agente
 : identidad? historia? estados? inventario? instanciador? percibir?
 conocimiento?
 ;

estados
 : ESTADOS DOSPUNTOS bloque_defs
 ;

inventario
 : INVENTARIO DOSPUNTOS bloque_defs
 ;

instanciador
 : INSTANCIADOR DOSPUNTOS bloque_join
 ;

percibir
 : PERCIBIR DOSPUNTOS bloque_join
 ;
```

```
bloque_defs
 : LLAVEIZ ((asignacion | definicion) PCOMA)* LLAVEDE
 ;

bloque_join
 : LLAVEIZ (sec_join bloque)* LLAVEDE
 ;

sec_join
 : tipos? atomo (JOIN atomo)*
 ;

atomo
 : CADENA | variable_tipo
 ;

variable_tipo
 : identificador DOSPUNTOS tipo_dato
 ;

teatro
 : TEATRO DOSPUNTOS LLAVEIZ escenografia? obras? puesta_en_escena LLAVEDE
 ;

escenografia
 : ESCENOGRAFIA DOSPUNTOS bloque
 ;

obras
 : OBRAS DOSPUNTOS bloque_obra
 ;

bloque_obra
 : LLAVEIZ obra_senten LLAVEDE
 ;

obra_senten
 : tipos? identificador PAREN_AP lista_pars? PAREN_CI LLAVEIZ ELENCO DOSPUNTOS
```

```
 bloque_defs DIALOGO DOSPUNTOS bloque MUTIS DOSPUNTOS bloque LLAVEDE
 ;

puesta_en_escena
 : PUESTA_EN_ESCENA PAREN_AP lista_pars? PAREN_CI bloque
 ;

bloque_funs
 : LLAVEIZ fun_senten* LLAVEDE
 ;

bloque_servs
 : LLAVEIZ servicio_senten* LLAVEDE
 ;

sec_proposiciones
 :  (accion_paralela | proposicion)+
 ;

sec_proposiciones_canal
 :  (accion_paralela_canal | proposicion | proposicion_canal)+
 ;

consumir_canal
 : ESCRIBIR (ANGIZ ENTERO ANGDE)? (DEF definicion | identificador) PCOMA
 ;

proposicion
 : RETORNAR (expresion | asignacion) PCOMA
 | si_senten
 | seleccionar_senten
 | mientras_senten
 | para_senten
 | hacer_mientras_senten
 | asignacion PCOMA
 | conjunto ENVIAR_MENSAJE (identificador | REF_INST) PCOMA
 | llamado PCOMA
 | OTRO {System.err.println("Caracter desconocido: " + $OTRO.text);}
```

```
 ;

definicion
 : identificador DOSPUNTOS tipo_dato
 ;

asignacion
 : DEF? identificador (DOSPUNTOS tipo_dato)? ASIGNAR conjunto  #asigVec
 | DEF? identificador (DOSPUNTOS tipo_dato)? ASIGNAR expresion #asigNum
 ;

accion_paralela
 : ANGIZ paralelo+ ANGDE
 ;

paralelo
 : O proposicion+
 ;

accion_paralela_canal
 : ANGIZ paralela+ ANGDE
 ;

paralela
 : O (proposicion | proposicion_canal)+
 ;

accion_decision
 : ANGIZ decision+ ANGDE USANDO identificador PAREN_AP lista_valores? PAREN_CI
 PCOMA
 ;

decision
 : INCREMENTO proposicion+
 ;

si_senten
 : SI bloque_condicional (SI_NO SI bloque_condicional)* (SI_NO ENTONCES? bloque)?
```

```
 ;

bloque_condicional
 : expresion (ENTONCES)? (bloque|expresion PCOMA)
 ;

mientras_senten
 : MIENTRAS bloque_condicional
 ;

hacer_mientras_senten
 : HACER bloque MIENTRAS expresion PCOMA
 ;

seleccionar_senten
 : SELECCIONAR identificador LLAVEIZ casos LLAVEDE
 ;

casos
 : CASO expresion DOSPUNTOS sec_proposiciones (ROMPER PCOMA)?
 casos     #casosGen
 | DEFECTO DOSPUNTOS sec_proposiciones                    #casosDef
 ;

para_senten
 : PARA asignaciones PCOMA expresion PCOMA asignaciones bloque
 | PARA PAREN_AP asignaciones PCOMA expresion PCOMA asignaciones PAREN_CI
 bloque
 ;

asignaciones
 : asignacion (COMA asignacion)*
 ;

fun_senten
 : tipos? identificador PAREN_AP lista_pars? PAREN_CI bloque
 ;
```

```
servicio_senten
 : tipos? identificador PAREN_AP lista_pars? PAREN_CI bloque_servicio
 ;

bloque
 : LLAVEIZ sec_proposiciones? LLAVEDE
 ;

bloque_canal
 : LLAVEIZ consumir_canal+ (proposicion | proposicion_canal)* LLAVEDE
 ;

bloque_decisiones
 : LLAVEIZ (sec_proposiciones | proposicion_canal)* accion_decision LLAVEDE
 ;

bloque_servicio
 : LLAVEIZ sec_proposiciones_canal? LLAVEDE
 ;

proposicion_canal
 : CANAL operacion_canal (expresion  | DEF definicion ) PCOMA
 ;

exp_simple
 : PAREN_AP exp_simple PAREN_CI
 | (op=(SUMA | MENOS))? termino (op2=(SUMA | MENOS | O) termino)*
 | termino
 ;

expresion
 : PAREN_AP expresion PAREN_CI
 | expresion op=(IGUAL | DIFERENTE | MEN_IGUAL | MAY_IGUAL | MENOR | MAYOR)
 expresion
 | exp_simple ENVIAR_MENSAJE exp_simple
 | (clase | IDENTIFICADOR) INSTANCIAR ANGIZ lista_valores ANGDE
 | REFERENCIA
 | REF_INST
```

```
 | exp_simple
 | llamado
 | tipo_dato_simple PAREN_AP expresion PAREN_CI
 ;

variable
 : identificador conjunto
 | identificador
 ;

termino
 : PAREN_AP termino PAREN_CI
 | factor (op=(MULT | DIV | MOD | Y | O | POTENCIA) factor)*
 | NO termino
 ;

factor
 : ENTERO
 | REAL
 | CADENA
 | COMPLEJO
 | NULO
 | variable
 | objeto
 | REF_INST
 | identificador (ANGIZ factor ANGDE)+
 | identificador
 | llamado
 | NO factor
 | VERDADERO
 | FALSO
 | conjunto
 | PAREN_AP expresion PAREN_CI
 ;

objeto
 : (elemento '.')* identificador (ANGIZ expresion ANGDE)?
 ;
```

```
elemento
 : identificador modo?
 ;

identificador
 : CARACTER | IDENTIFICADOR
 ;

modo
 : ANGIZ expresion ANGDE
 | PAREN_AP lista_valores? PAREN_CI
 ;

llamado
 : objeto PAREN_AP lista_valores? PAREN_CI
 ;

lista_canales
 : canal (COMA canal)*
 ;

canal
 : CANAL DOSPUNTOS tipos
 ;

lista_valores
 : expresion (COMA expresion)*
 ;

lista_pars
 : par (COMA par)*
 ;

par
 : identificador DOSPUNTOS tipos
 ;
```

```
conjunto
 : ANGIZ (expresion (COMA expresion)*)? ANGDE
 ;

tipo_dato
 : tipos ARREGLO? ARREGLO?
 | LLAVEIZ dupla_tipos LLAVEDE
 ;

dupla_tipos
 : tipos COMA tipo_dato
 ;

tipos
 : CLUSTER | ROL | ARREGLO | tipo_dato_simple | FUNCION
 | TIPO_RECURSOS | TIPO_REFERENCIA | TIPO_UBICACION | clase
 ;

tipo_dato_simple
 : INT | FLOAT | STRING | BOOL
 ;

clase
 : UNIVERSO | PAIS | AGENTE | INSTITUCION | IDIOMA | AGENTE_ESENCIAL
 ;

operacion_canal
 : LEER | ESCRIBIR
 ;
```

B.2.1. Tokens del lenguaje

```
COMENTARIO : ('#' ~[\r\n]*  | '/*' .*? '*/') -> skip;
AGENTE : 'AGENTE';
AGENTE_ESENCIAL : 'AGENTE_ESENCIAL';
CANALES : 'CANALES' ;
CONOCIMIENTO : 'CONOCIMIENTO';
CONSTITUCION : 'CONSTITUCION_DEL_PAIS';
```

```
CULTURA : 'CULTURA';
DECIDIR : 'DECIDIR';
DIALOGO : 'DIALOGO';
DISPONIBILIDAD : 'DISPONIBILIDAD';
ELENCO : 'ELENCO';
ESCENOGRAFIA : 'ESCENOGRAFIA';
ESPACIOTIEMPO : 'ESPACIO_TIEMPO';
ESTADOS : 'ESTADOS';
GRUPO_LOGISTICO : 'GRUPO_LOGISTICO';
HISTORIA : 'HISTORIA';
IDENTIDAD : 'IDENTIDAD';
IDIOMAS : 'IDIOMAS';
IDIOMA : 'IDIOMA';
INSTANCIADOR : 'INSTANCIADOR';
INSTITUCION : 'INSTITUCION';
INVENTARIO : 'INVENTARIO';
LEYES: 'LEYES';
MIEMBROS : 'MIEMBROS';
MUTIS : 'MUTIS';
OBRAS : 'OBRAS';
ORGANIGRAMA : 'ORGANIGRAMA';
PARTICIPANTES : 'PARTICIPANTES';
PERCIBIR : 'PERCIBIR';
PUESTA_EN_ESCENA : 'PUESTA_EN_ESCENA';
REDIRIGIR_MENSAJES : 'REDIRIGIR_MENSAJES';
SERVICIOS : 'SERVICIOS';
TEATRO : 'TEATRO';
TIPO_RECURSOS : 'RECURSOS';
TIPO_REFERENCIA : 'REFERENCIA';
TIPO_UBICACION : 'UBICACION';
UNIVERSO : 'UNIVERSO';
BOOL : 'booleano';
CASO: 'caso';
CLUSTER : 'cluster';
DEF : 'def';
DEFECTO : 'defecto';
EN : 'en';
ENTONCES : 'entonces';
```

```
FALSO : 'falso';
FLOAT : 'decimal';
FUNCION : 'funcion';
HACER: 'hacer';
INT : 'entero';
MIENTRAS : 'mientras';
NULO : 'nulo';
PAIS : 'PAIS';
PARA : 'para';
PARA_CADA: 'para_cada';
RETORNAR : 'retornar';
ROL : 'ROL';
ROMPER: 'romper';
SELECCIONAR : 'seleccionar';
SI : 'si';
SI_NO : 'si_no';
STRING : 'cadena';
USANDO : 'usando';
VACIO : 'vacio';
VERDADERO : 'verdadero';
ENVIAR_MENSAJE : '->';
INSTANCIAR : '<-';
CONSUMIBLE : '?';
ESCRIBIR : '==>';
LEER : '<==';
O : '||';
Y : '&&';
JOIN : '&';
IGUAL : '==';
DIFERENTE : '!=';
MAYOR : '>';
MENOR : '<';
MAY_IGUAL : '>=';
MEN_IGUAL : '<=';
INCREMENTO : '++';
SUMA : '+';
MENOS : '-';
MULT : '*';
```

```
DIV : '/';
MOD : '%';
POTENCIA : '^';
NO : '!';
COMILLA : '\'';
PCOMA : ';';
ASIGNAR : '=';
PAREN_AP : '(';
PAREN_CI : ')';
LLAVEIZ : '{';
LLAVEDE : '}';
ARREGLO : '[]';
ANGIZ : '[';
ANGDE : ']';
COMA : ',';
DOSPUNTOS : ':';
PUESTO : ENTERO DOSPUNTOS IDENTIFICADOR;
CANAL : IDENTIFICADOR MULT;
CARACTER : [a-zA-Z_];
IDENTIFICADOR : CARACTER ALFANUMERICO* ('.' CARACTER ALFANUMERICO*)*;
HORARIO : 'de ' ENTERO ' a ' ENTERO;
REFERENCIA: (ENTERO '.')+;
REF_INST: IDENTIFICADOR ('.' ENTERO)*;
ENTERO : [0-9]+;
ALFANUMERICO : [a-zA-Z_0-9];
REAL : [0-9]* '.' [0-9]* ([eE] [+-]? [0-9]+)?;
COMPLEJO : (ENTERO|REAL) [+|-] (ENTERO|REAL)? 'i';
CADENA : '"' (~["\r\n] | '""')* '"';
ESPACIO : [ \t\r\n] -> skip;
OTRO : .;
```

B.2.2. Clases de Java

Para generar clases a partir de ANTLR, se debe añadir la ruta de la librería en la variable de entorno CLASSPATH, luedo usar el comando `java org.antlr.v4.Tool DiUNisio2.g4`, el cual generará las clases necesarias para la fase de análisis semántico. A continuación se presentan clases que sobreescriben las clases generadas anteriormente, junto con clases de manejo de símbolos.

- `Symbol.java` con la cual se definen los símbolos del lenguaje, usados para almacenar las propiedades de cada definición de símbolo.
- `SymbolTable.java` en donde se define la estructura Tabla de Símbolos, usada para almacenar los símbolos del programa basado en su alcance.
- `EvalVisitor.java` en donde se sobrescriben los métodos de `Diunisio2BaseVisitor.java` previamente generado a partir de la gramática en ANTLR, para definir operaciones semánticas.
- `Main.java` donde se introduce la entrada escrita en el lenguaje Diunisio 2.0 para poder generar código intermedio.

B.3. Recomendaciones y Sugerencias

Para desarrollar un procesador de lenguajes usando ANTLR, es necesario diseñar la gramática con base en la funcionalidad requerida desde el comienzo, esta debe ser revisada y probada para asegurarse de que no hayan ambigüedades. También es necesario conocer los patrones generados por la librería ANTLR para poder aprovechar sus capacidades al máximo. Se recomienda usar el entorno de desarrollo IntelliJIdea, el cual contiene un plugin de ANTLR que facilita la ejecución de pruebas sobre la gramática y la generación de clases de Java para el análisis semántico.

Bibliografía

[1] Librería ANTLR (Versión 4.8) {software}. (2020). Obtenida de: `http://www.antlr.org/`

C. Manual de técnico de LIST

LIST: Manual técnico

Andrés Felipe De Orcajo Vélez
Jorge Eduardo Ortiz Triviño

Universidad Nacional de Colombia
Facultad de Ingeniería, Departamento de Ingeniería de Sistemas e Industrial
Bogotá, Colombia
2020

C.1. ANTLR

La gramática fue modelada con la ayuda de la librería ANTLR[1], definiendo el léxico y la gramática en un archivo con extensión *.g4*.

C.2. Gramática de LIST (en notación E-BNF)

```
grammar LIST;

inicio
 : assemblies* decls*
 ;

assemblies
 : NAMESPACE IMPORT IDENTIFICADOR
 | NAMESPACE IDENTIFICADOR
 ;

decls
 : INSTANCIATE? CLASS accesibilidad? IDENTIFICADOR herencia? LLAVE_AP
 (atributos | constructor | metodos | decls)* main? LLAVE_CI
 ;

accesibilidad
 : PUBLIC | PRIVATE
 ;

herencia
 : EXTENDS (REFERENCIA | IDENTIFICADOR)
 ;

atributos
 : REFERENCE? ATTRIBUTE modificador IDENTIFICADOR
 (ASIGNA (tipo_retorno PAREN_AP list_vals? PAREN_CI))?
 ;

constructor
 : METHOD accesibilidad INSTANCE VOID CONSTRUCTOR PAREN_AP list_args?
```

```
 PAREN_CI LLAVE_AP sentencias* LLAVE_CI
 ;

role
 : ROLE ENTERO DOS_PUNTOS IDENTIFICADOR (CONSUMES IDENTIFICADOR)?
 ;

metodos
 : (RECEPTOR | role)? METHOD modificador IDENTIFICADOR PAREN_AP list_args?
 PAREN_CI LLAVE_AP sentencias* LLAVE_CI
 ;

main
 : ENTRYPOINT METHOD PUBLIC STATIC VOID MAIN PAREN_AP list_args? PAREN_CI
 LLAVE_AP ENTRYPOINT? sentencias* LLAVE_CI
 ;

modificador
 : accesibilidad? tipo_retorno
 ;

tipo_retorno
 : INT32 | FLOAT64 | VOID | STRING | ROLE | REFERENCIA | OBJECT | INSTANCE
 | ARRAY | BOOL
 | tipo_retorno MEN list_pars MAY
 | tipo_retorno ANG_AP ANG_CI
 ;

sentencias
 : ETIQUETA? op_binaria
 | ETIQUETA? op_unaria
 | ETIQUETA? op_simple
 ;

op_simple
 : IDENTIFICADOR
 ;
```

```
op_unaria
 : IDENTIFICADOR factor
 | IDENTIFICADOR PAREN_AP list_labels PAREN_CI
 | IDENTIFICADOR ETIQUETA_SIMPLE
 | IDENTIFICADOR tipo_retorno
 ;

op_binaria
 : IDENTIFICADOR REFERENCIA atributo
 | IDENTIFICADOR tipo_retorno atributo (PAREN_AP list_pars? PAREN_CI)?
 | IDENTIFICADOR REFERENCIA
 | IDENTIFICADOR tipo_retorno REFERENCIA PAREN_AP list_pars? PAREN_CI
 ;

atributo
 : (REFERENCIA | IDENTIFICADOR) CUATRO_PUNTOS (CONSTRUCTOR | IDENTIFICADOR)
 ;

def_pars
 : tipo_retorno (ANG_AP list_pars? ANG_CI)? IDENTIFICADOR (COMA def_pars)?
 ;

list_args
 : tipo_retorno IDENTIFICADOR (COMA tipo_retorno IDENTIFICADOR)*
 ;

list_pars
 : tipo_retorno (COMA tipo_retorno)*
 ;

list_vals
 : factor (COMA factor)*
 ;

list_labels
 : ETIQUETA_SIMPLE (COMA ETIQUETA_SIMPLE)*
 ;
```

```
factor
 : ENTERO
 | REAL
 | CADENA
 | NULO
 ;
```

C.2.1. Tokens del lenguaje

```
COMENTARIO : ('#' ~[\r\n]*  | '/*' .*? '*/') -> skip;
CONSUMES : '@consumes';
ENTRYPOINT : '@entrypoint';
INSTANCIATE : '@instantiate';
RECEPTOR : '@receptor';
REFERENCE : '@reference';
ROLE : '@role';

ATTRIBUTE : 'attribute';
CLASS : 'class';
CONSTRUCTOR : 'constructor';
EXTENDS : 'extends';
IMPORT : 'import';
INSTANCE : 'instance';
MAIN: 'Main';
METHOD : 'method';
NAMESPACE : 'namespace';
PRIVATE : 'private';
PUBLIC : 'public';
STATIC : 'static';

ARRAY : 'array';
BOOL : 'bool';
FLOAT64 : 'float64';
INT32 : 'int32';
NULO : 'null';
OBJECT : 'object';
STRING : 'string';
VOID : 'void';
```

```
COMA: ',';
PAREN_AP : '(' ;
PAREN_CI : ')' ;
LLAVE_AP : '{';
LLAVE_CI : '}';
ANG_AP : '[';
ANG_CI : ']';
MEN : '<';
MAY : '>';
CUATRO_PUNTOS : '::';
DOS_PUNTOS : ':';
ASIGNA : '=';

ETIQUETA : 'L'[a-fA-F0-9]+ ':';
ETIQUETA_SIMPLE : 'L'[a-fA-F0-9]+;
REFERENCIA : ANG_AP IDENTIFICADOR ANG_CI IDENTIFICADOR;
REAL : [0-9]* '.' [0-9]* ([eE] [+-]? [0-9]+)?;
CADENA : '"' (~["\r\n] | '""')* '"';
IDENTIFICADOR : [a-zA-Z_] [a-zA-Z_0-9\\.]*;
ENTERO : [0-9]+;
ESPACIO : [ \r\t\n] -> skip;
```

Bibliografía

[1] Librería ANTLR (Versión 4.8) {software}. (2020). Obtenida de: `http://www.antlr.org/`

D. Conjunto de microinstrucciones

Instrucción	**Descripción**
add	Suma dos valores, retornando un nuevo valor.
and	Realiza la operación AND a nivel de bits de dos valores integrales, retorna un nuevo valor.
argsl	Retorna la lista de argumentos para el método actual.
beq <int32 (target)>	Salta al objetivo si es igual.
bge <int32 (target)>	Salta al objetivo si es mayor o igual que.
bgt <int32 (target)>	Salta al objetivo si es mayor que.
ble <int32 (target)>	Salta al objetivo si es menor o igual que.
blt <int32 (target)>	Salta al objetivo si es menor que.
bne <int32 (target)>	Salta al objetivo si es diferente.
br <int32 (target)>	Salta al objetivo.
brfalse <int32 (target)>	Salta al objetivo si el valor es cero (falso).
brtrue <int32 (target)>	Salta al objetivo si el valor no es cero (verdadero).
call <method>	Llama el método descrito por el parámetro.
calli <callsitedescr>	Llama método indicado en la pila con argumentos descritos por el parámetro.
ceq	Apila 1 (de tipo int32) si el valor 1 equivale al valor 2, de lo contrario apila 0.
cgt	Apila 1 (de tipo int32) si el valor 1 es mayor que el valor 2, de lo contrario apila 0.
clt	Apila 1 (de tipo int32) si el valor 1 es menor que el valor2, de lo contrario apila 0.
toint	Convierte a entero nativo, apilando entero nativo en pila.
toflt	Convierte un entero sin signo a punto flotante, apilando F en pila.
cpblk	Copia datos de memoria a memoria.
div	Divide dos valores para retornar el cociente o punto flotante.
dup	Duplica el valor en la cima de la pila.
initblk	Fija todos los bytes en un bloque de memoria a determinado valor de byte.

Instrucción	**Descripción**
jmp <method>	Salta del método actual y salta al método especificado.
ldarg <int32 (num)>	Carga el argumento num en la pila.
ldarga <int32 (argNum)>	Busca la dirección del argumento argNum.
ldint <int (num)>	Apila un número de tipo int32 en la pila.
ldflt <float64 (num)>	Apila un número de tipo float64 en la pila.
ldftn <method>	Apila un puntero al método referenciado en la pila.
ldloc <int32 (indx)>	Carga la variable local de índice indx en la pila.
ldloca <int32 (indx)>	Carga la dirección de una variable local con índice indx.
ldnull	Apila una referencia nula en la pila.
localloc	Asigna espacio de la reserva de memoria local.
mul	Multiplica valores.
neg	Niega valores.
nop	No hace nada.
not	Complementa a nivel de bits (NO lógico).
or	A nivel de bits realiza la operación OR de dos valores, retorna un nuevo valor.
pop	Desapila un valor de la pila.
rem	Residuo de la división de un valor por otro.
ret	Retorno de un método, posiblemente con un valor.
shl	Aplica shift a un entero hacia la izquierda (desplazando en ceros), retorna un entero.
shr	Aplica shift a un entero hacia la derecha (cambio en signo), retorna un entero.
starg <int32 (num)>	Almacena el valor al argumento num.
stint	Almacena valor del tipo entero nativo en la dirección de memoria.
stind.ref	Almacena valor de tipo objeto ref (tipo O) en la dirección de memoria.
stloc <int32 (indx)>	Desapila un valor de la pila en la variable local indx.
sub	Resta el valor 2 del valor 1, retornando un nuevo valor.
switch <int32, int32, int32 (t1..tN)>	Salta a uno de los n valores.
xor	A nivel de bits realiza la operación XOR de valores enteros, retorna un entero.
box <typeTok>	Convierte un valor boxeable en su forma box.
callvirt <method>	Llama un método asociado con un objeto.
castclass <class>	Convierte obj a la clase dada.
cpobj <typeTok>	Copia un valor de tipo src a dest.

Instrucción	**Descripción**
initobj <typeTok>	Inicia el valor en la dirección dest.
instof <class>	Prueba si obj es una instancia de clase, retornando nulo o una instancia de esa clase o interfaz.
ldelem <typeTok>	Carga el elemento en el índice a la cima de la pila.
ldelema <class>	Carga la dirección del elemento en el índice a la cima de la pila.
ldfld <field>	Apila el valor del atributo del objeto (o tipo de valor) obj, en la pila.
ldflda <field>	Apila la dirección del atributo del objeto obj en la pila.
newobj <ctor>	Asigna un objeto no inicializado o tipo de valor y llama el constructor.
rethrow	Relanza la excepción actual.
sizeof <typeTok>	Apila el tamaño, en bytes, de un tipo de dato como un int32 sin signo.
stelem <typeTok>	Reemplaza el elemento del arreglo en el índice con el valor sobre la pila.
stobj <typeTok>	Almacena un valor de tipo typeTok en una dirección.
stsatt <field>	Reemplaza el valor del atributo estático con val.
throw	Lanza una excepción.
unbox <valuetype>	Convierte una forma box en un valor nativo.
break	Informa a un depurador que el breakpoint ha sido alcanzado.
leave <int32 (target)>	Sale de una región de protegida de código.
ldlen	Apila el tamaño (de tipo nativo entero sin signo) del arreglo sobre en la pila.
ldobj <typeTok>	Copia el valor almacenado en la dirección src a la pila.
ldsatt <field>	Apila el valor del atributo estático en la pila.
ldsatta <field>	Apila la dirección del atributo estático, a la pila.
ldstr <string>	Apila una cadena de caracteres.
ldvirtftn <method>	Apila la dirección de un método virtual sobre la pila.
newarr <etype>	Crea un nuevo arreglo de elementos de tipo etype.
stfld <field>	Reemplaza el valor del atributo del objeto obj con el valor.

E. Conjunto de microservicios

Grupo	Subgrupo	Microservicios
	Collections	ArrayList
		BitArray
		CaseInsensitiveComparer
		Comparer
		HashTable
		Queue
		SortedList
		Stack
	Collections.Concurrent	ConcurrentDictionary<TKey,TValue>
		ConcurrentQueue<T>
		ConcurrentStack<T>
	Collections.Generic	ArraySortHelper<Tkey, TValue>
		ByteEqualityComparer
		Comparer<T>
		Dictionary<TKey, TValue>
		EqualityComparer<TKey, TValue>
		List<T>
		NullableComparer<T>
		ObjectComparer<T>
		ObjectEqualityComparer<T>
	Collections.ObjectModel	Collection<T>
		KeyedCollection<TKey, TItem>
		ReadOnlyCollection<T>
		ReadOnlyDictionary<TKey, TValue>
	Diagnostics	Assert
		Debugger
		Log
		StackTrace
	Diagnostics.Tracing	EventListener
	Globalization	Calendar

Grupo	Subgrupo	Microservicios
	IO	BinaryReader
		BinaryWriter
		BufferedStream
		Directory
		DirectoryInfo
		DriveInfo
		File
		FileAccess
		FileAttributes
		FileInfo
		FileShare
		FileStream
		FileSystemInfo
		Iterator <Tsource>
		Path
		PathHelper
		PathInternal
		ReadLinesIterator
		SearchOption
		SearchResultHandler <Tsource>
		SeekOrigin
		Stream
		StreamReader
		StreamWriter
		TextReader
		TextWriter
	Net	ping()
		unicast()
		multicast()
		broadcast()
		send()
		receive()
		isAlive()
		addRoute()
		removeRoute()
		virtualSocket()
	Runtime.Serialization	DeserializationHandler

Grupo	Subgrupo	Microservicios
		ObjectCloneHelper
		Formatter
		SerializationHandler
	Security.AccessControl	AccessRule
		AuditRule
		AutorizathionRule
		ControlFlags
		CryptoKey
		DirectorySecurity
		FileSecurity
		FileSystem
		Privilege
		Rights
		Security
		SystemAcl
	Security.Cryptography	AES
		CipherMode
		Constants
		CryptoConfig
		DES
		DSA
		FromBase64Transform
		HashAlgorithm
		HMACMD5
		HMASHA1/256/384/512
		MD5
		RC2
		RSA
		SHA1/256/384/512
		ToBase64Transform
		TripleDES
		Utils
	Security.Permissions	FileIOAccess
		FileIOPermission
	Security.Principal	SOVORAAccountType
		SOVORAIdentity
		ASCIIEncoding

Text

Grupo	Subgrupo	Microservicios
		Decoder
		Encoder
		Normalization
		StringBuilder
	Threading	Lock
		Mutex
		ReaderWriterLock
		Semaphore
		Thread
		ThreadPool
		ThreadPriority
		ThreadStart
		ThreadState
		Timeout
		Timer
		WaitHandle
	Threading.Tasks	AwaitTaskContinuation
		CompletionActionInvoker
		Parallel
		ParallelOptions
		Shared<T>
		Task
		TaskContinuation
		TaskScheduler
	Reflection	call()
		resolver()
		Action
		AppContext
		Array
		Attribute
		Boolean
		Buffer
		Byte
		Char
		Console
		Convert
		Currency

Grupo	**Subgrupo**	**Microservicios**
		DateTime
		Decimal
		Double
		Empty
		Enum
		Exception
		Func<T, TResult>
		Int16/32/64
		IntPtr
		Math
		Number
		Object
		OperatingSystem
		Random
		Resolver
		String
		StringComparer
		Tuple<...>
		Type
		Void
Social	Agent	create()
		migrate()
		replicate()
		destroy()
	Ambient	create()
		exit()
		enter()
		open()
	Actor	sendMessage()
		registerActor()
		findActor()
		changeState()
		createActor
		checkChild()
		notifySupervisor()
	Channel	consume()
	Country	

Grupo	**Subgrupo**	**Microservicios**
	Institution	reference()
	Language	
	Location	
	LogisticsGroup	Member
		decide()
	Role	
	Schedule	
	Theater	
	Universe	

Bibliografía

[1] Aceto, Luca ; Ingólfsdóttir, Anna ; Larsen, Kim G. ; Srba, Jiri: *Reactive Systems: Modelling, Specification and Verification.* USA : Cambridge University Press, 2007. – ISBN 0521875463

[2] Agha, G.: *Actors: A Model of Concurrent Computation in Distributed Systems.* MIT Press, 1986

[3] Aho, Alfred V. ; Sethi, Ravi ; Ullman, Jeffrey D.: *Compiladores: Princípios, Técnicas e Ferramentas.* 1986. – ISBN 0201100886

[4] Alexander, Jhon ; Ortíz Triviño, Jorge E.: Diseño Socio-inspirado de la Máquina Virtual Distribuida (Örbis Virtüalis Maqüinus ÖVM) para la red Ad-Hoc TLÖN. En: *Cicom.* Cartagena, 2015

[5] Alvergue, L D. ; Pandey, A ; Gu, G ; Chen, X: Consensus control for heterogeneous multiagent systems. En: *SIAM Journal on Control and Optimization* 54 (2016), Nr. 3, p. 1719–1738. – ISSN 03630129 (ISSN)

[6] Andrews, Gregory R.: The distributed programming language SR—Mechanisms, design and implementation. En: *Software: Practice and Experience* 12 (1982), aug, Nr. 8, p. 719–753. – ISSN 00380644

[7] Barendregt, H. P. ; Van Eekelen, M. C. ; Plasmeijer, M. J. ; Glauert, J. R. ; Kennaway, J. R. ; Sleep, M. R.: LEAN: An intermediate language based on graph rewriting. En: *Parallel Computing* 9 (1989), jan, Nr. 2, p. 163–177. – ISSN 01678191

[8] Bison, Paolo ; Pagello, Enrico ; Trainito, Gaetano: VML: An intermediate language for robot programming. En: *Robotics and Computer Integrated Manufacturing* 5 (1989), jan, Nr. 1, p. 11–19. – ISSN 07365845

[9] Blessing, Lucienne T. ; Chakrabarti, Amaresh ; London, Springer-Verlag (Ed.): *DRM, a Design Research Methodology.* Springer-Verlag London, 2009. – ISBN 978–1–84882–586–4

[10] Cardelli, L. ; Gordon, A. D.: Mobile ambients. En: *Theoretical Computer Science* Vol. 240 (1), 2000, p. 177–213

[11] Church, A.: *The Calculi of Lambda Conversion*. Princeton University Press, 1941

[12] De Carvalho, Jorge V.: Sinais de fogo: metamorfose e epigenia. En: *Metamorfoses - Revista de Estudos Literários Luso-Afro-Brasileiros* (2010). – ISSN 0875–019X

[13] De Porre, Kevin ; Myter, Florian ; Scholliers, Christophe ; Gonzalez Boix, Elisa: CScript: A distributed programming language for building mixed-consistency applications. En: *Journal of Parallel and Distributed Computing* 144 (2020), oct, p. 109–123. – ISSN 07437315

[14] Dectia: Integrated Cyber Security and Information Assurance. 2011. – Informe de Investigación

[15] Dijiang Huang, Huijun W.: *Mobile Cloud Computing: Foundations and Service Models*. O'Reilly, sep 2017

[16] Dyson, George B.: *Darwin among the Machines: The Evolution of Global Intelligence*. USA : Addison-Wesley Longman Publishing Co., Inc., 1997. – ISBN 0201406497

[17] ECMA International: *Standard ECMA-335 - Common Language Infrastructure (CLI)*. 5. Geneva, Switzerland, December 2010

[18] Fournet, C. ; Gonthier, G.: The reflexive CHAM and the join-calculus. En: *In ACM Symposium on Principles of Programming Languages* (1996), p. 372–385

[19] Goodwin, Jeff ; Jasper, James M.: Emotions and social movements. En: *Handbook of the Sociology of Emotions*. Springer, 2006, p. 611–635

[20] Guo, L ; Liu, Z ; Chen, Z: A leader-based cooperation-prompt protocol for the prisoner's dilemma game in multi-agent systems. En: T., Liu (Ed.) ; Q., Zhao (Ed.): *36th Chinese Control Conference, CCC 2017*. College of Computer and Control Engineering, Nankai University, Tianjin, 300353, China : IEEE Computer Society, 2017. – ISBN 19341768 (ISSN); 9789881563934 (ISBN), p. 11233–11237

[21] Hewitt, C.: Viewing control structures as patterns of passing messages. En: *Artificial Intelligence 8(3)*, 1977, p. 323–364

[22] Hoare, Charles Anthony R. ; Wirth, Niklaus: An axiomatic definition of the programming language Pascal. En: *Acta Informatica* 2 (1973), Nr. 4, p. 335–355

[23] Hobbes, T. ; Gaskin, J.C.A.: *Leviathan*. Oxford University Press, 1996 (Oxford world's classics). – ISBN 9780192834980

[24] Hodgson, Geoffrey M.: ¿Qué son las instituciones? En: *Revista CS* (2011). – ISSN 2011–0324

[25] Ivanova, V.: The model of the organization of information provision in a knowledge based economy. En: *Economy and Forecasting* (2011)

[26] Jones, S. L.: FLIC—a Functional Language Intermediate Code. En: *ACM SIGPLAN Notices* 23 (1988), aug, Nr. 8, p. 30–48. – ISSN 15581160

[27] Kuhn, S.T.: *La estructura de las revoluciones científicas*. Fondo de Cultura Economica, 2011 (Ciencia y Tecnología). – ISBN 9786071608253

[28] Lafont, Yves: Interaction Nets. En: *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. New York, NY, USA : Association for Computing Machinery, 1989 (POPL '90). – ISBN 0897913434, p. 95–108

[29] Milner, J. P. ; Walker, D.: A calculus of mobile processes, Part I/II. En: *Journal of Information and Computation* (1992), sep, Nr. 100, p. 1–77

[30] Milner, Robin: Bigraphical reactive systems. En: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2001. – ISBN 3540424970

[31] Rawls, John: *A Theory of Justice*. 1. Cambridge, Massachussets : Belknap Press of Harvard University Press, 1971. – ISBN 0–674–88014–5

[32] Russell, S.J. ; Russell, S.J. ; Norvig, P. ; Davis, E.: *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2010 (Prentice Hall series in artificial intelligence). – ISBN 9780136042594

[33] Sanchez Cifuentes, Joaquin F.: *Prototipo de un lenguaje de programación para la implementación de servicios a través de comunidades de agentes social inspirados sobre redes Ad-hoc*, Tesis de Grado, 2018. – 123 p.

[34] Sartre, J. P.: *Being and nothingness*. London : Routledge, 1943

[35] Schopenhauer, A.: *El Mundo Como Voluntad y Representacion*. Fondo De Cultura Economica USA, 2004 (Sección de obras de filosofía). – ISBN 9788437505695

[36] Scott, Michael L.: The Lynx distributed programming language: Motivation, design and experience. En: *Computer Languages* 16 (1991), jan, Nr. 3-4, p. 209–233. – ISSN 00960551

[37] Singh, S: Detection of emergent behaviors in system of dynamical systems using similitude theory. En: V., Chan (Ed.): *2017 Winter Simulation Conference, WSC 2017*. Department of Electrical and Computer Engineering, Northeastern University, 360 Huntington Avenue, Boston, MA 02115, United States : Institute of Electrical and Electronics Engineers Inc., 2018. – ISBN 08917736 (ISSN); 9781538634288 (ISBN), p. 4650–4651

[38] de Spinoza, B. ; Peña, V.: *Ética demostrada según el orden geométrico*. Alianza Editorial, 1999 (El Libro de Bolsillo/Alianza Editorial Series). – ISBN 9788420635095

[39] Téllez-Vargas, Jorge: Teoría de la mente: evolución, ontogenia, neurobiología y psicopatología. En: *Avances en psiquiatría biológica* (2006)

[40] Tinaut, Alberto ; Ruano, Francisca: Biodiversidad, clasificación y filogenia. En: *Sistemática y Diversidad*. 2000

[41] Van Cutsem, Tom ; Gonzalez Boix, Elisa ; Scholliers, Christophe ; Carreton, Andoni L. ; Harnie, Dries ; Pinte, Kevin ; De Meuter, Wolfgang: AmbientTalk: Programming responsive mobile peer-to-peer applications with actors. En: *Computer Languages, Systems and Structures* 40 (2014), oct, Nr. 3-4, p. 112–136. – ISSN 14778424

[42] Varela, Carlos A.: *Programming Distributed Computing Systems: A Foundational Approach*. The MIT Press, 2013. – ISBN 0262018985

[43] Waldron, J.: Moral autonomy and personal autonomy. En: *Autonomy and the challenges to liberalism* (2005), p. 307–329

[44] Zárate Ceballos, Henry: *Diseño de un Sub-Sistema de Cómputo Distribuido que permita implementar virtualización inalámbrica para gestionar recursos (Procesamiento, memoria, almacenamiento y dispositivos E/S) distribuidos en una Red Ad Hoc, mediante el modelo de pseudo Estado*, Tesis de Grado, 2018. – 205 p.

[45] Zhao, Shushan ; Kent, Robert ; Aggarwal, Akshai: A Key Management and Secure Routing Integrated Framework for Mobile Ad-hoc Networks. En: *Ad Hoc Networks* (2012)