UNIVERSIDAD DE ORIENTE

ESCUELA DE INGENIERÍA Y CIENCIAS APLICADAS

DEPARTAMENTO DE COMPUTACIÓN Y SISTEMAS

TALLER DE INTELIGENCIA ARTIFICIAL



**INFORME 1**

Fundamentos de la IA e Introducción al desarrollo de los Agentes Inteligentes.

Profesor: Integrantes

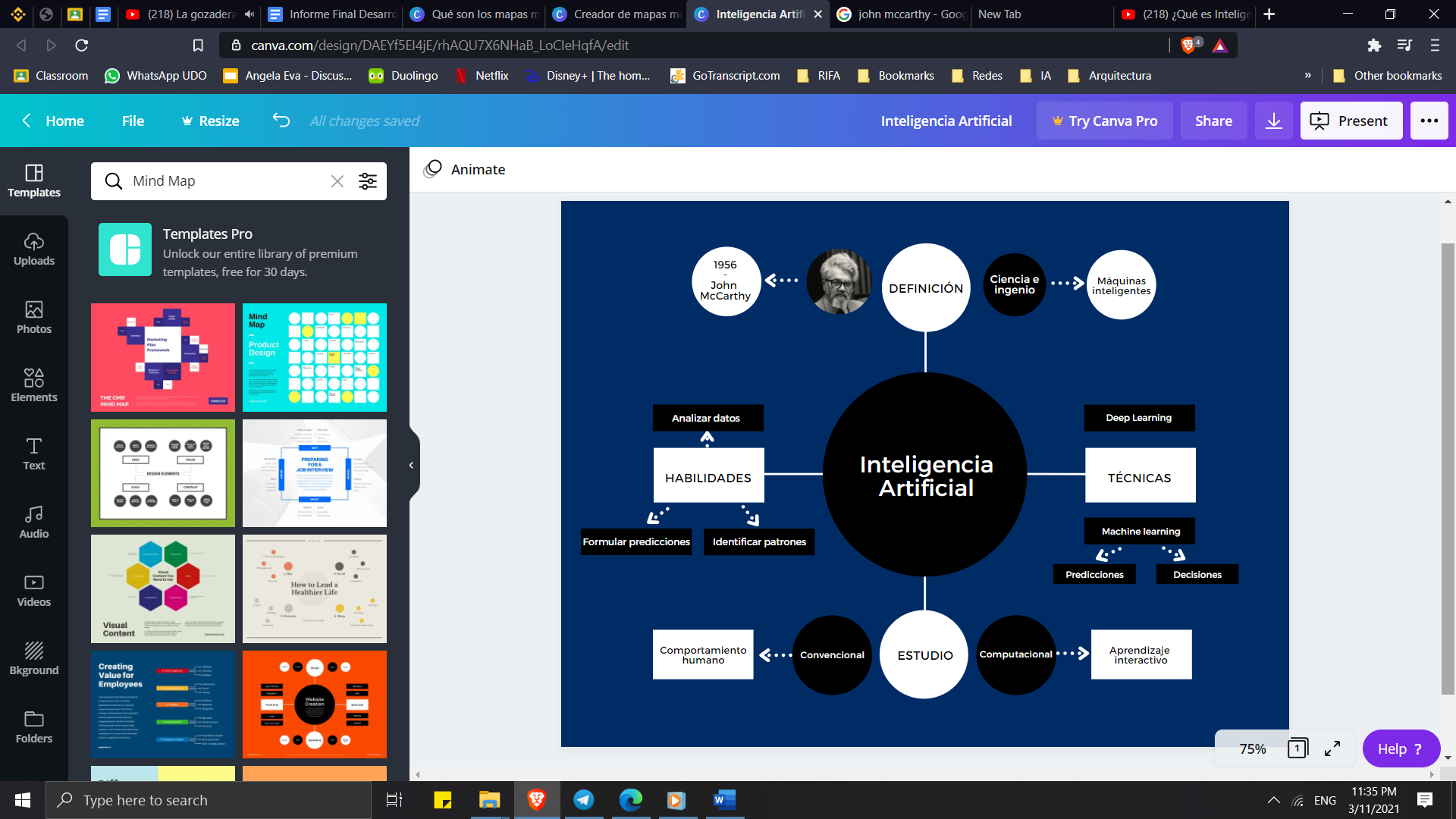
Claudio Cortínez Angela Ortiz C.I. 24.437.193

Daniel Guaicara C.I 25.572.667

Abril de 2021.

**Parte I: Fundamentos de la IA**

**1.** Elabore un mapa mental sobre la Inteligencia Artificial



**2**. Todos los años se otorga el premio Loebner al programa que lo hace mejor en una Prueba de Turing concreta. Investigue y haga un informe sobre el último ganador del premio Loebner. ¿Qué técnica utiliza? ¿Cómo ha hecho que progrese la investigación en el campo de la IA?

El último premio Loebner celebrado fue en el 2019, otorgado por cuarta vez al Chatbot creado en el 2005 por Steve Worswick, llamado Mitsuku. El cual básicamente es una máquina con capacidad de aprendizaje con propósitos educativos y de esparcimiento.

Ahora, para comprender un poco más sobre cómo se comunica Mitsuki y de todo lo que es capaz, primero debemos comprender realmente qué es un chatbot. Así como no los dice su nombre, es un programa que abarca los conceptos de mensaje automatizado, robot y el de inteligencia artificial. Lo que nos lleva a un concepto más técnico, “son sistemas de conversación de máquinas que interactúan con usuarios humanos a través de una conversación en lenguaje natural” (Shawar y Atwell, 2005).

Sim embargo, para lograr mantener esta función principal, este debe estar programado para dar respuestas acordes a palabras claves mediante un sistema de diálogo. Esto lo logra a partir de tres componentes principales como lo son: el contestador, el clasificador y el graphmaster.

Estos componentes son los que permiten la comunicación del Chatbot con el usuario. El contestador es el software preparado para las tareas repetitivas a través de internet simulados el comportamiento de un humano, transfiriendo y controlando la entrada y salida de datos del usuario al clasificador. Por su parte el clasificador procesa la entrada ingresada por el usuario en componentes lógicos y transfiere la oración al graphmaster, manejando a su vez las instrucciones de sintaxis de la base de datos. Por último, el graphmaster se encarga de organizar y manejar el proceso de coincidencia de patrones que implican búsquedas avanzadas.

Dicho esto, Mitsuko es un chatbot personificado por una animación de una chica en el estilo anime (formato japonés de ilustración) de 18 años. Ella puede llevar una conversación semejante a la que tendrías con una persona, aunque se encuentra mayormente orientada a la compañía, el ocio y el entretenimiento. Esta es capaz de dar respuestas rápidas y coherentes, aunque todavía con ciertos fallos de acuerdo al nivel de complejidad de la conversación.

Se comunica a través del idioma inglés, y al igual que en una conversación con un ser humano, ella también es capaz de hacerte preguntas, una característica que además le ayuda a tomar una u otra postura según tus respuestas y el tono de la conversación. Esto gracias a que Mitsuko utiliza un lenguaje marcado de inteligencia artificial o “coincidencia de patrones”, para interrogar e imitar la vasta simbología de una conversación humana.

Al igual que hace uso de técnicas de diseño especialmente creadas para las funciones de un chatbot, como la tecnología AIML (Artificial Intelligence Mark-up Languaje) usada por Worswick para crear a Mitsuko. AIML simplifica el trabajo de modelado conversacional, en relación con un proceso de respuesta de estímulo, es un lenguaje marcado con bases en XML, lo que además le permite a Mitsuko basarse en gran medida en señales semánticas para construir un conjunto de conocimientos, interpretando las conversaciones y aprendiendo de ellas, a través de la división de frases y oraciones en dos partes, el "núcleo" y el "comodín". De esta manera, descifra lo que los expertos en aprendizaje profundo llaman la intención original.

Lo que quiere decir, de la forma más básica posible, es que Mitsuko a través de la plataforma Pandorabots, esta creada en base a un conjunto de archivos de texto con un formato especial que son los que permiten su respuesta a cada una de nuestras preguntas u oraciones. De hecho, su propio creador ofrece a los usuarios de Mitsuku la lista de archivos AIML que él mismo utilizó para Mitsuko, e incentiva así la creación de nuevos Chatbots.

Y aunque debido a la imprevisibilidad de frases y preguntas que muchos de los usuarios pueden utilizar al conversar con Mitsuko, esta se encuentra limitada en sus respuestas en cierto punto, ella es capaz de recordar datos e información, incluso puedes corregirla. Por lo que, aunque sigue sin ser verdaderamente conversacional, es una pequeña prueba de todos los progresos que la inteligencia artificial ha logrado con respecto al procesamiento del lenguaje natural y las interfaces conversacionales.

“¿Qué tan humano puedes hacer un chatbot?”, es la pregunta que se hizo Travis Nelson, jefe de producto y diseño de Pandorabots, empresa que creó a Mitsuku, quien afirma que, “Filosóficamente, pensamos que los chatbots deberían identificarse a sí mismos como bots. Por ejemplo, no estamos tratando de ser humanos, ¿verdad? No estamos tratando de crear algo que pueda engañar a alguien haciéndole creer que está hablando con un humano real”, dejándonos saber así que Mitsuku es una demostración de lo que se puede lograr a través de plataformas como Pandorabots, que ofrece sus servicios web para crear e implementar chatbots.

Es por esto que gracias a Mitsuko y proyectos similares, hemos permitido como usuarios que las computadoras nos conozcan cada vez más, y de tal forma puedan atender y reaccionar a nuestras necesidades, incluso ayudándonos con ciertas tareas como las que ofrecen los asistentes virtuales, que, aunque no están preparados todavía para mantener conversaciones, se encuentra en una rápida expansión de su conocimiento y servicios cada vez más humanos. Para muestra, Ray Kurzweil, fundador de Singularity University (universidad financiada por Google), asegura que, “la inteligencia artificial llegará a ser más inteligente y poderosa que la de un ser humano, las máquinas estarán dotadas de conciencia y emociones, lo que las hará más autónomas”.

Mientras, Mitsuko seguirá comunicándose vía internet, día y noche, con millones de usuarios alrededor del mundo a quienes puede dar a conocer ciertos rasgos de su “personalidad” a partir de curiosas afirmaciones, como su gusto por los kebats, que Terminator sea su película favorita, sus conocimientos de las leyes de la robótica de Asimov y su opinión sobre tópicos como el clima, los horóscopos y hasta el sentido de la vida.

**3**. Consulte en la literatura existente sobre la IA si alguna de las siguientes tareas se puede efectuar con computadores:

1. Jugar una partida de tenis de mesa (ping-pong) decentemente.

Sí. Fue logrado por el robot de Andersson, capaz de jugar un juego de tenis de mesa razonablemente bien.

1. Conducir un coche en el centro del Cairo.

No bien, pero se puede resolver. Técnicamente no puede hacerse. Se han hecho pruebas sobre esto, pero ninguna resultó ser completamente precisa. Sigue necesitando más tiempo. Cambiar algunos carriles y curvas es algo que puede hacerse en caminos claramente marcados para así moderar el tráfico. Manejar en el centro del Cairo es muy impredecible para que esto funcione. Google está trabajando en un carro que se maneja solo, pero necesita demasiados sensores de movimiento y tendría problemas para llenar el tanque de gasolina.

1. Comprar comestibles para una semana en el mercado.

Sí, con suficiente data. La computadora necesitaría conocer que quiere el usuario y tener una forma de identificar alimentos sin códigos de barra.

1. Comprar comestibles para una semana en la web.

Sí. Se debe decir a la computadora lo que quieres, dimensiones (si aplica), cantidad, etc. También tu información de pago para que pueda hacer la transacción.

1. Jugar una partida de bridge decentemente a nivel de competición.

Sí. Hay sitios web que permiten jugar juegos de bridge a nivel avanzado, si bien sigue siendo inferior al de los mejores jugadores humanos.

1. Descubrir y demostrar nuevos teoremas matemáticos.

Puede probar teoremas existentes usando algún tipo de explicación provista a la computadora, pero no puede descubrir nada por sí misma. Necesitaría tener conciencia.

1. Escribir intencionadamente una historia divertida.

No. Las computadoras no entienden el concepto de comedia. El usuario puede poner chistes, pero la computadora no sabrá cómo crear una historia divertida con material nuevo.

1. Ofrecer asesoría legal competente en un área determinada.

En algunos casos, sí. La inteligencia artificial tiene una larga historia en cuanto a investigación de aplicaciones de razonamiento legal automatizado. Por ejemplo, los sistemas expertos basados en Prolog del Reino Unido usados para guiar miembros del público en los procesos y detalles de la seguridad social y leyes de nacionalidad. Pero para cualquier cosa más compleja que eso, no.

1. Traducir inglés hablado al sueco hablado en tiempo real.

De forma limitada, pero sí, incluso ya se está haciendo. Google Translator lo hace, pero no es suficientemente preciso.

1. Realizar una operación de cirugía compleja.

No todavía. Existen robots que han logrado hacer operaciones simples, pero algo más complejo tiene demasiadas variables cambiando continuamente que la computadora no puede analizar correctamente.

**Parte II: Introducción al desarrollo de los Agentes Inteligentes.**

4. Metodologías para desarrollar Agentes Inteligentes.

Existen diferentes metodologías que permiten el desarrollo de agentes inteligentes, entre las más utilizadas se encuentran GAIA, INGENIAS, Vowel Engineering, MAS-CommonKADS, TROPOS, AUML y Prometheus. A continuación, rasgos importantes de algunas de estas metodologías:

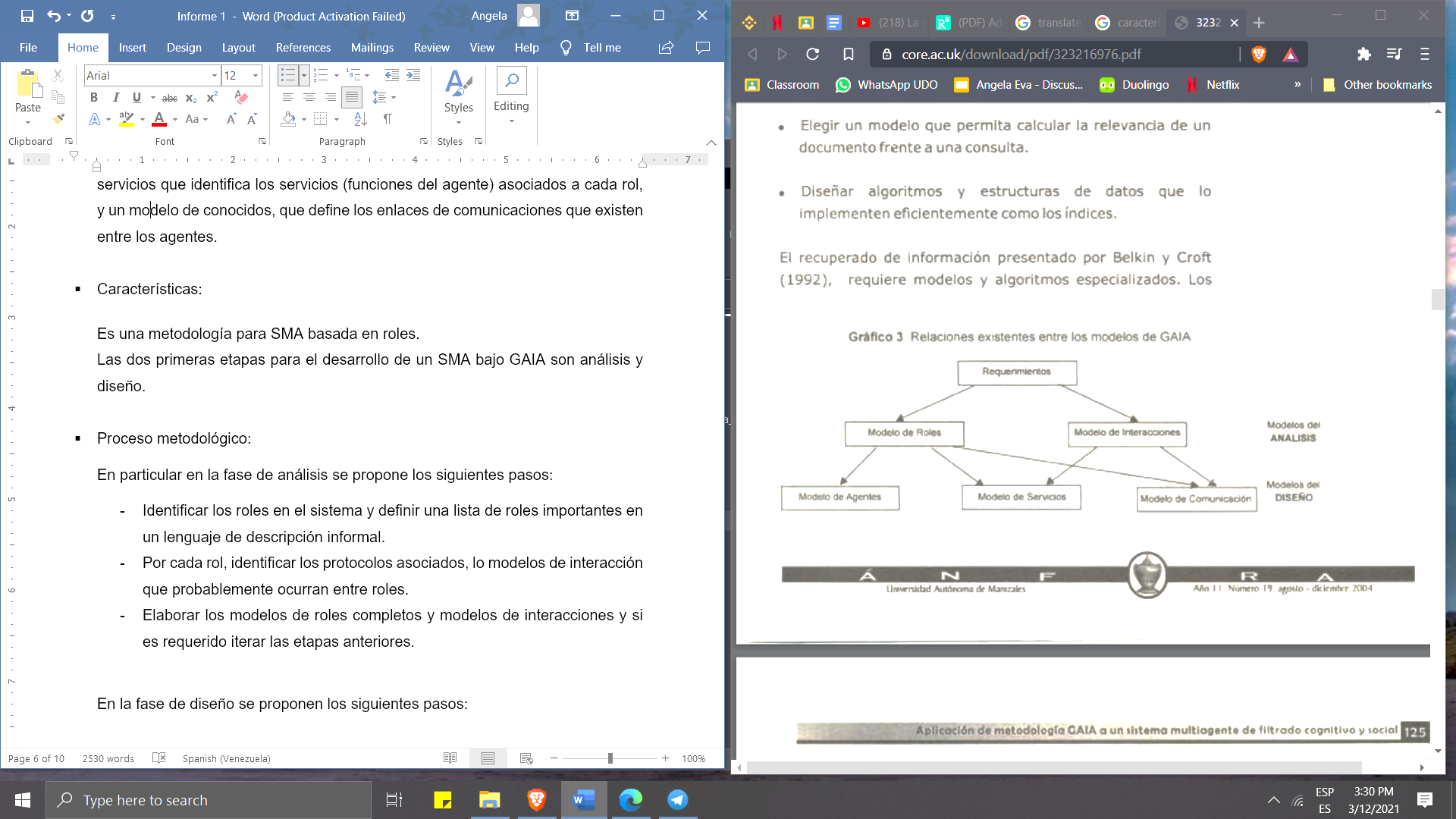
**GAIA:** es una metodología para el diseño de sistemas basados en agentes que tiene como objetivo la obtención de un sistema que maximice alguna medida de calidad global. Esta metodología presta ayuda al analista para ir sistemáticamente desde unos requisitos iniciales a un diseño lo suficientemente detallado como para ser implementado directamente.

El objetivo de análisis de GAIA es conseguir comprender el sistema y su estructura sin referenciar ningún aspecto de implementación. Esto se consigue a través de la idea de organización, que en esta metodología se trata de una colección de roles, los cuales mantienen ciertas relaciones con otros y toman parte en patrones institucionalizados de interacción con otros roles. Los roles agrupan cuatro aspectos: responsabilidades del agente, los recursos que se le permite utilizar, las tareas asociadas e interacciones.

* Antecedentes: Gaia es una metodología orientada a agentes desarrollada inicialmente por Wooldridge, y posteriormente revisada y mejorada por Zambonelli. En la versión original, el proceso de desarrollo que sustenta a Gaia estaba compuesto solo por dos fases - análisis y diseño - y los conceptos clave adoptados fueron roles junto con las responsabilidades, permisos, actividades y protocolos asociados. Los roles podían interactuar entre sí de ciertas formas institucionalizadas, que se definen en los protocolos de los roles respectivos.

La metodología Gaia fue una de las primeras que apareció en el campo AOSE y funcionó como una especie de punto de referencia para el desarrollo de las subsiguientes metodologías AOSE. Sin embargo, la versión original de Gaia era bastante limitada, de hecho, no tenía en cuenta varios aspectos como el diseño de agentes interesados, el diseño de sistemas dinámicos y abiertos, las estructuras organizativas y el diseño de protocolos de cooperación. Estas limitaciones se abordan en la nueva versión de la metodología, donde Zambonelli introdujo el análisis y el diseño de las estructuras organizacionales, el análisis del entorno y la perspectiva del diseño para el cambio para apoyar sistemas abiertos y dinámicos. Estas mejoras llevaron a una reformulación del proceso de Gaia, incluida la introducción de una nueva fase - Diseño Arquitectónico - y una serie de nuevos modelos.

* Fundamentos: GAIA propone trabajar inicialmente con un análisis a alto nivel. En este análisis se usan dos modelos, el modelo de roles para identificar los roles clave en el sistema junto con sus propiedades definitorias y el modelo de interacciones que define las interacciones mediante una referencia a un modelo institucionalizado de intercambio de mensajes. Tras esta etapa, se entraría en lo que GAIA considera diseño a alto nivel. El objetivo de este diseño es generar tres modelos: el modelo de agentes que define los tipos de agente que existen, cuántas instancias de cada tipo y qué papeles juega cada agente, el modelo de servicios que identifica los servicios (funciones del agente) asociados a cada rol, y un modelo de conocidos, que define los enlaces de comunicaciones que existen entre los agentes.
* Características:
* Es una metodología para SMA basada en roles.
* Las dos primeras etapas para el desarrollo de un SMA bajo GAIA son análisis y diseño.
* Es una metodología a un nivel de abstracción alto.
* Hace uso combinado de fórmulas lógicas.
* Omite las dependencias entre los modelos propuestos.
* No cuenta con una gran cantidad de herramientas que la soporte.
* Proceso metodológico:



En particular en la fase de análisis se propone los siguientes pasos:

* Identificar los roles en el sistema y definir una lista de roles importantes en un lenguaje de descripción informal.
* Por cada rol, identificar los protocolos asociados, los modelos de interacción que probablemente ocurran entre roles.
* Elaborar los modelos de roles completos y modelos de interacciones y si es requerido iterar las etapas anteriores.

En la fase de diseño se proponen los siguientes pasos:

* Identificar el modelo de agente, es decir, agregar roles en tipos de agentes y refinarlos.
* Identificar los servicios que los agentes tienen que proveer en orden para que cumplan los roles asignados.
* Desarrollar el modelo de conocimiento para identificar ineficiencias en el diseño y si es requerido iterar las etapas anteriores.

**MAS-CommonKADS:** se trata de una metodología orientada al desarrollo utilizando experiencia en sistemas expertos. Esta gira entorno del modelo de experiencia y está pensada para desarrollar sistemas expertos que interactúen con el usuario.

* Antecedentes: CommonKADS fue propuesta y desarrollada por un grupo de investigadores pertenecientes a diversos países de la Unión Europea, a través del programa ESPRIT para la innovación y la aplicación de Tecnología Informática avanzada. El trabajo se comenzó en 1983 cuando había poco interés en este tipo de metodologías, para esa fecha la construcción de sistemas de conocimiento estaba enmarcada bajo el paradigma de desarrollo por prototipos y de representación del conocimiento a través de reglas de producción, con hardware y software de propósito especial como máquinas LISP y herramientas especiales para sistemas expertos.

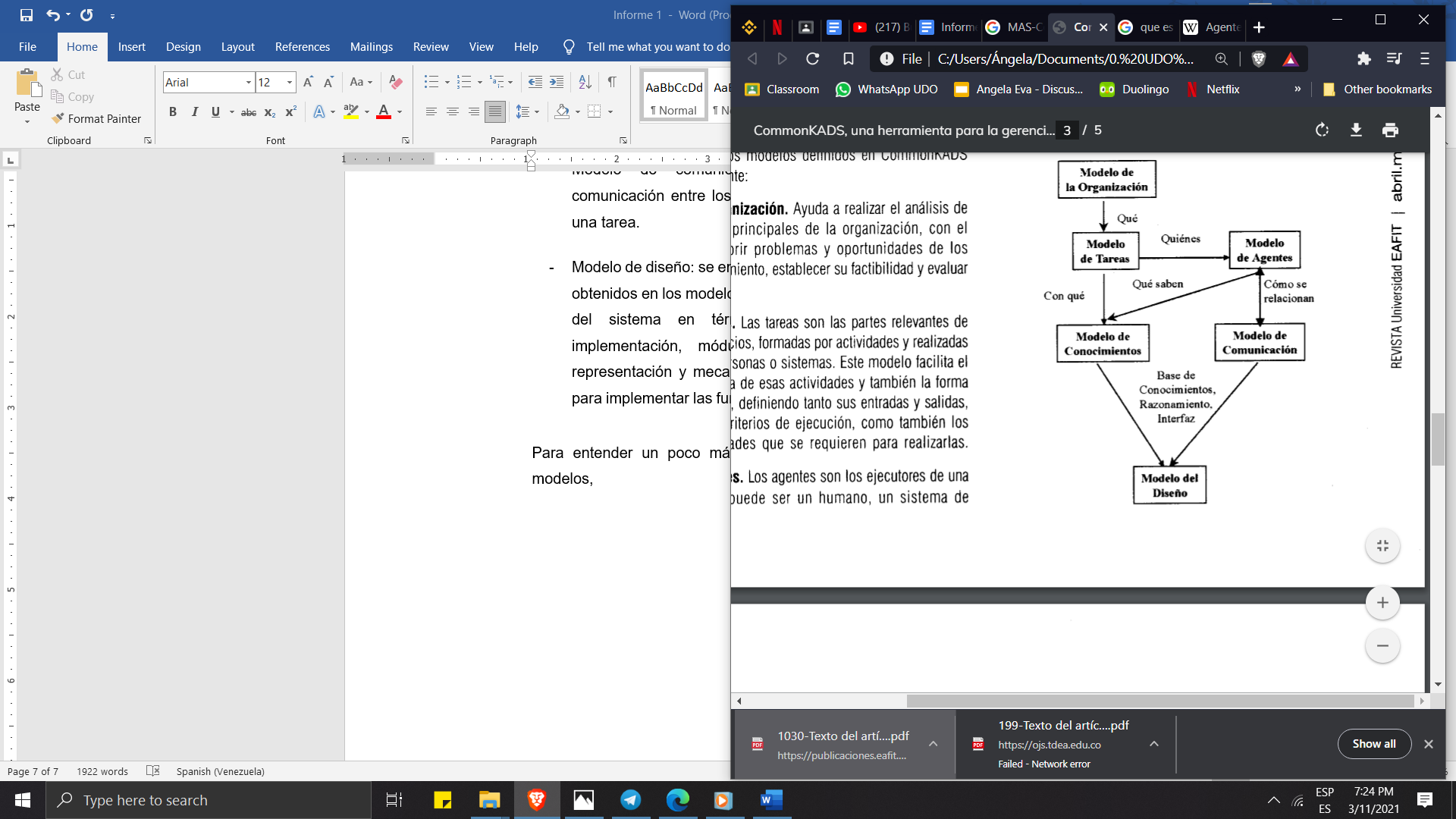
A pesar de que el proyecto termino en 1994 se continuaron desarrollando investigaciones alrededor de dicha metodología, a través de tesis doctorales que le han adicionado funcionalidad a la metodología, como propuestas para comenzar a generar métodos de solución de problemas para el dominio del diagnóstico. Otras propuestas fueron para añadirle a la metodología características que permitan que sea utilizada en el desarrollo de sistemas multiagentes o para sistemas inteligentes en tiempo real, pasando a ser conocida en este ámbito como MAS-CommonKADS.

* Fundamentos: esta metodología está orientada a la realización de tres actividades fundamentales:

1. Actividades de modelación: es el desarrollo de un conjunto de modelos que expresan diferentes perspectivas de la situación puesta en análisis y para la cual se piensa construir el sistema basado en el conocimiento. Son modelos que conforman el denominado Modelo del producto, el cual permite realizar el estudio del problema y de su solución. Dichos modelos son:

* Modelo de organización: ayuda con el análisis de las características principales de la organización, con el objetivo de descubrir problemas y oportunidades de los sistemas de conocimientos, estableciendo su factibilidad e impacto en ella.
* Modelo de tareas: este modelo permite el análisis de las actividades que forman las tareas realizadas por uno o varias personas o sistemas, cómo se relacionan y definiendo sus entradas y salidas, precondiciones y criterios de ejecución, al igual que sus recursos y habilidades requeridas para realizarlas.
* Modelo de agentes: describe las características de los agentes, sus habilidades, formas de actuar y las restricciones que pueden tener en ciertas tareas. A su vez, define las formas de comunicación existentes entre los agentes que realizan una determinada tarea.
* Modelo de conocimiento: el cual tiene como propósito explicar en detalle los tipos y estructuras del conocimiento usado en la ejecución de una tarea. Este aporta una descripción de la implementación de la tarea, independientemente del papel que los diferentes componentes del conocimiento juegan en la solución del problema.
* Modelo de comunicación: modela las transacciones de comunicación entre los agentes involucrados en la realización de una tarea.
* Modelo de diseño: se encuentra basado en todos los requerimientos obtenidos en los modelos anteriores y da una especificación técnica del sistema en términos de arquitectura, plataforma de implementación, módulos de software, construcción de la representación y mecanismos computacionales, todos necesarios para implementar las funciones que han sido previamente definidas.

Para entender un poco más sobre cómo funciona este conjunto de modelos, veamos el siguiente esquema:



Aunque cabe resaltar que no siempre es necesario desarrollar todos estos modelos, pues su creación dependerá de los objetivos del proyecto.

1. Actividades de administración del proyecto: es un ciclo que consiste en identificar objetivos del sistema, hacer el estudio de riesgos y amenazas, así como definir las actividades requeridas con la finalidad de lograr dichos objetivos, hacer las tareas definidas y comparar el plan de trabajo realizado con lo definido en los objetivos iniciales. En caso de no haber logrado lo planteado, se hace un replanteamiento de objetivos y se continúa con los pasos, lo cual generará un ciclo que terminará una vez que se alcancen las metas y se logra obtener la solución definitiva.
2. Actividades de reutilización: se da la reutilización de modelos y del código creado para una solución específica, con lo cual se busca mejorar la productividad en el desarrollo de sistemas basados en conocimientos y una buena calidad de productos.

* Características:
* Plantea un desarrollo de SMA integrado con un ciclo de vida de software, concretamente el espiral dirigido por riesgos.
* Propone una lista detallada de elementos y relaciones a identificar en el sistema.
* Describe con detalle cómo se debe definir el sistema teniendo en cuenta las dependencias entre los modelos.
* Requiere del apoyo de herramientas de soporte.
* La información que hay que obtener se expresa con lenguaje natural.
* Proceso metodológico: MAS-CommonKADS es probablemente una de las primeras metodologías en plantear un desarrollo de SMA integrado con un ciclo de vida de software, concretamente el espiral dirigido por riesgos. Pues con los modelos propuestos para la definición del sistema, se presentan referencias a la teoría sobre la que se basa. Cada modelo en sí parte de una descripción gráfica que luego se complementa con explicaciones en lenguaje natural de cada elemento, además, existe por cada modelo una descripción de las dependencias respecto de otros modelos y de las actividades involucradas.

Estos modelos se hallan descritos ampliamente en lenguaje natural, complementándose con otras notaciones como SDL (Specification and Description Language) o MSC (Message Sequence Chart) para describir el comportamiento de los agentes cuando interaccionan.

El inconveniente para resaltar de esta metodología es que el nivel de detalle alcanzado en la descripción no es realizable sin el apoyo de herramientas de soporte. Lo que propone MAS-CommonKADS, entre otras cosas, no es una notación sino una lista detallada de elementos y relaciones a identificar en el sistema. Un desarrollador puede seguir la lista y generar la documentación requerida de forma manual, sin embargo, el proceso es demasiado costoso y dado a errores.

**5.** Herramientas para desarrollar Agentes Inteligentes

1. **JADE**

* Descripción: Java Agent DEvelopment Framework, o JADE, es una plataforma [software](https://es.wikipedia.org/wiki/Software) para el desarrollo de [agentes](https://es.wikipedia.org/wiki/Agente_inteligente_(Inteligencia_Artificial)), implementada en [Java](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Java). La plataforma JADE soporta la coordinación de múltiples agentes [FIPA](https://es.wikipedia.org/wiki/FIPA) y proporciona una implementación estándar del lenguaje de comunicación FIPA-ACL, que facilita la comunicación entre agentes y permite la detección de servicios que se proporcionan en el sistema. JADE fue desarrollado originalmente por [Telecom Italia](https://es.wikipedia.org/wiki/Telecom_Italia) y se distribuye como [software libre](https://es.wikipedia.org/wiki/Software_libre).

JADE es un [middleware](https://es.wikipedia.org/wiki/Middleware) que facilita el desarrollo de sistemas multi-agente bajo el estándar [FIPA](https://es.wikipedia.org/wiki/FIPA) para lo cual crea múltiples contenedores destinados a los agentes, cada uno de los cuales puede ejecutarse en uno o en varios sistemas. Queda establecido que un conjunto de contenedores constituye una plataforma.

* Lenguajes que soporta: Java.
* Características: Se trata de una plataforma de agentes distribuida, que tiene un contenedor por cada host en el que se están ejecutando los agentes. De manera adicional la plataforma posee diferentes herramientas de depuración, movilidad de los agentes de código y contenido, la posibilidad de la ejecución paralela del comportamiento de los agentes, así como el soporte para la definición de lenguajes y [ontologías](https://es.wikipedia.org/wiki/Ontologia). Cada plataforma debe tener un contenedor principal que tiene dos agentes especiales denominados AMS y DF.
* El DF (Directory Facilitator) proporciona un directorio que anuncia qué agentes hay disponibles en la plataforma.
* El AMS (Agent Management System) controla la plataforma. Es el único que puede crear y destruir a otros agentes, destruir contenedores y parar la plataforma.

Para acceder al agente DF se usa la clase "jade.domain.DFService" y sus métodos estáticos: register, deregistrer, modify y Search.

Para acceder a la información del AMS Service se crea un agente el cual se ejecuta automáticamente el método register del AMS por defecto antes de ejecutar el método setup del nuevo agente. Cuando se destruye un agente se ejecuta su método takedown() y automáticamente se llama a deregister del AMS por defecto.

La clase Agent es una super clase común que permite a los usuarios crear [software](https://es.wikipedia.org/wiki/Software) de agentes JADE. Para crear un agente solo hay que heredar de la clase Agent(). Habitualmente cada agente registrará varios servicios los cuales a su vez deberían de ser implementados por uno o más comportamientos.

Esta clase suministra métodos que permiten ejecutar las tareas básicas de los agentes como:

* Pasar mensajes utilizando objetos ACLMessage, con correspondencia de patrones.
* Dar soporte al ciclo de vida de un agente.
* Planificar y ejecutar múltiples actividades al mismo tiempo.

El ciclo de vida de un agente JADE sigue el ciclo propuesto por [FIPA](https://es.wikipedia.org/wiki/FIPA). Estos agentes pasaran por diferentes estados definidos como:

1. Initiated/ Iniciado: El agente se ha creado pero no se ha registrado todavía el AMS.
2. Active/ Activo: El agente ya ha sido registrado y posee nombre. En este estado puede comunicar con otros agentes.
3. Suspended/ Suspendido: El agente se encuentra parado porque su hilo de ejecución se encuentra suspendida.
4. Waiting/ Esperando: Se encuentra bloqueado a la espera de un suceso.
5. Deleted/ Eliminado: El agente ha terminado por tanto el hilo terminó su ejecución y ya no estará más en el AMS.
6. Transit/ Transito: El agente se está migrando a una nueva ubicación.

El comportamiento define las acciones bajo un determinado evento. Dicho comportamiento del agente se define en el método setup mediante el método addBehaviour. Los diferentes comportamientos que el agente adoptará se definen a partir de la clase abstracta Behaviour. La clase Behaviour contiene los métodos abstractos:

* action(): Se ejecuta cuando la acción tiene lugar.
* done(): Se ejecuta al finalizar el comportamiento.

Por otra parte el usuario puede redefinir los métodos onStart() y onEnd() que el agente poseerá. Adicionalmente existen otros métodos como block() y restart() usados para la modificación del comportamiento del agente. Cuando un agente esté bloqueado se puede desbloquear de diferentes maneras.

El paso de mensajes ACL (Agent Communication Language) es la base de la comunicación entre agentes. El envío de mensajes se realiza mediante el método send de la clase Agent. A dicho método hay que pasarle un objeto de tipo ACLMessage que contiene la información de los destinatarios, lenguaje, codificación y el contenido del mensaje. Estos mensajes se envían de modo asíncrono, mientras que los mensajes que se van recibiendo se irán almacenando en una cola de mensajes. Existen dos tipos de recepción de los mensajes ACL, bloqueante o no bloqueante. Para ello se porporcionan los métodos blockingReceive() y receive() respectivamente. En ambos métodos se puede hacer filtrado de los mensajes que se quieren recuperar de la cola estableciendo diferentes plantillas.

JADE tiene una extensión denominada [WADE](https://es.wikipedia.org/wiki/WADE) (Workflows and Agents Development Environment) que es un sistema de [workflow](https://es.wikipedia.org/wiki/Workflow" \o "Workflow) que permite crear procesos mediante un editor gráfico llamado WOLF.

* Ejemplos y/o aplicaciones:

1. WANTS, por Telecom Italia.
2. La creación de Jadescript.
3. **ABLE**

* Descripción: El proyecto Agent Building and Learning Environment (ABLE) comenzó en el laboratorio de investigaciones T.J.Watson de IBM a principios de 1999. El objetivo del proyecto era producir un conjunto de herramientas veloz, reusable y reescalable para la creación de aplicaciones de software inteligentes. La versión 1.0 de ABLE fue publicada en el sitio web alphaWorks de IBM en mayo del año 2000. El equipo de desarrollo se ha encargado de brindar actualizaciones con regularidad en alphaWorks, con pre-lanzamientos centrados en el framework central y herramientas basadas en Swing, además de muchos años de de trabajo en el lenguaje basado en reglas ABLE (ARL) y motores, seguidos por un agente multiplataforma para ABLE y herramientas para el uso con Eclipse. Las subidas más recientes añaden modelado basado en agentes y extensiones de simulación además del juego de herramientas ABLE y mejoras de soporte a Eclipse 3.7.
* Lenguaje que la soporta: JAVA y ARL.
* Características:
* Un framework en JAVA para la creación de agentes inteligentes con soporte para eventos, comunicación y operaciones asíncronas.
* Agentes de machine learning para la clasificación, predicción, previsión y clustering con soporte en Beans que proveen accesos a archivos y bases de datos, y filtrados.
* Un componente de razonamiento que incluye un lenguaje de programación basado en reglas conocido como ABLE Rule Language (ARL). ARL es un lenguaje similar a JAVA con motores de procesamiento pro reglas que varían desde scripting procedural simple, pasando por encadenado, Prolog y más.
* Un set de simulación de clases y agentes para el soporte de modelados basados en agentes de gran escala y estudios de simulación.
* Una plataforma de agentes que facilita el manejo de agentes y comunicación a través de una red distribuida de sistemas.

ABLE incluye las siguientes herramientas:

* Agent Editor, un editor de agente para el montado y conexión de componentes (plugin de Eclipse).
* Rule Editor, un editor de texto para desarrollar y debuggear sets de reglas (plugin de Eclipse).
* Simulation Editor, un editor gráfico para la creación de modelos de simulación incluyendo agentes y escenarios (plugin de Eclipse).
* Platform Console, una aplicación para manejar agentes distribuidos (plugin de Eclipse).
* Ejemplos y/o aplicaciones
* Autonomic Computing: Towards a Self-Healing System

<http://ilin.asee.org/Conference2007program/Papers/Conference%20Papers/Session%201A/Laster.pdf>

* RFID enhanced MAS for warehouse management

<https://www.tandfonline.com/doi/abs/10.1080/13675560701427379?journalCode=cjol20>