UNIVERSIDAD
DE LOS ANDES
MERIDA VENEZUELA

Ingeniería del Software Orientada a Agentes

Jose Aguilar

Ingeniería del Software Orientada a Agentes

- Los agentes representan un **nuevo nivel de abstracción** que puede ser utilizado por los desarrolladores de software para **entender, modelar y desarrollar** de un modo más natural una clase importante de sistemas distribuidos.

- Las técnicas de **desarrollo software habituales no son adecuadas para esta tarea**, ya que no son capaces de capturar los aspectos únicos de los SMA:
  - **Comportamiento flexible, autónomo, de resolución de problemas**
  - **Riqueza en sus interacciones**
  - **Complejidad de la estructura organizacional**

Metodologías de desarrollo de SMA

Como todo desarrollo de software se requieren unos estándares de desarrollo.

La metodología debe permitir ir actualizando los agentes en función de la caracterización que sea necesaria en el SMA.

Tener en cuenta las necesidades de especificación de los SMA

- Complejidad de la estructura organizacional
- Planificación de tareas
- Riqueza en sus interacciones: Intercambio de información con lenguajes de comunicación orientados a agentes
- Movilidad del código
- Motivación de los componentes del sistema
- Comportamiento flexible, autónomo, de resolución de problemas

La construcción de SMA integra tecnologías de distintas áreas de conocimiento

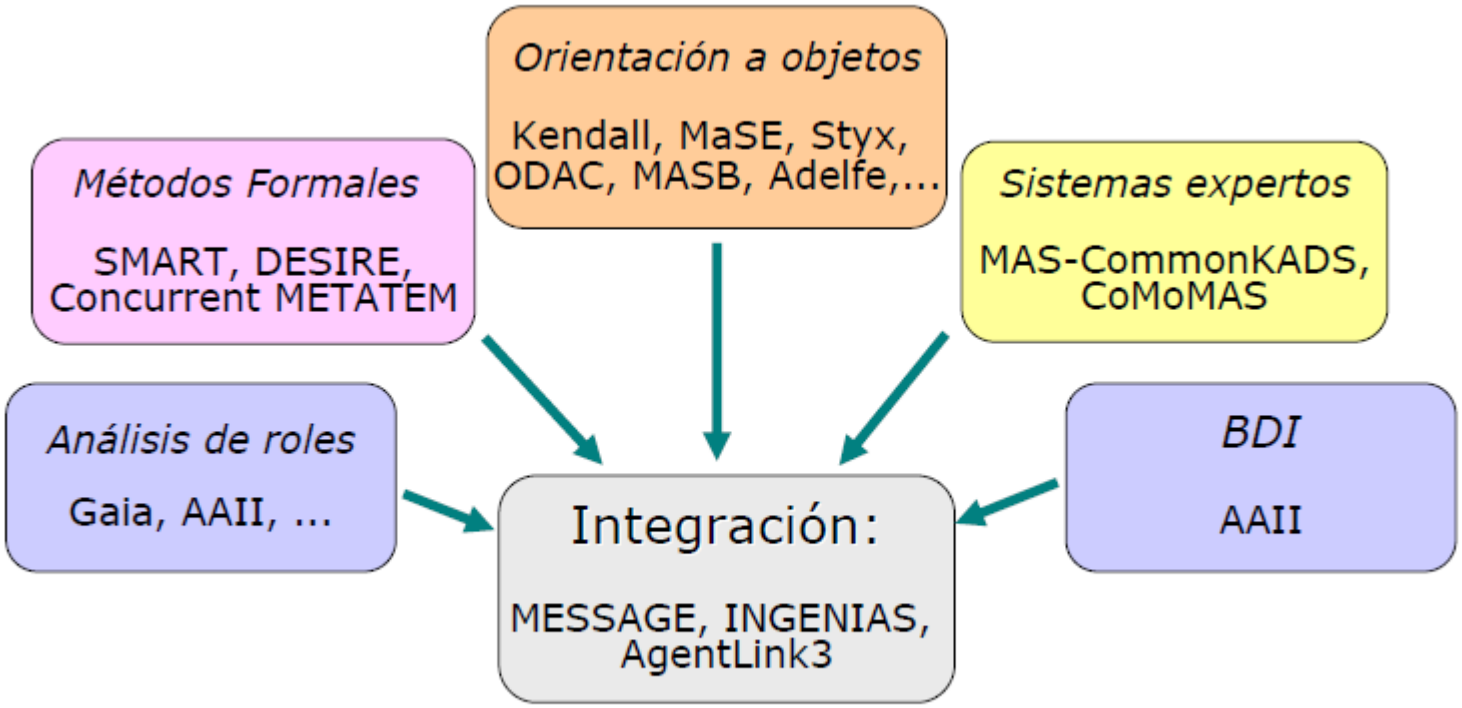
- **Técnicas de *ingeniería del software :*** *estructurar el proceso* de desarrollo
- **Técnicas de *inteligencia artificial :*** dotar a los programas de capacidad para tratar situaciones imprevistas y tomar decisiones
- **Programación concurrente y distribuida :** *tratar* la coordinación de tareas ejecutadas en diferentes
- máquinas bajo diferentes políticas de planificación.

Ingeniería del Software Orientada a Agentes

Resolver

- Las entidades que estarán representados en el sistema (del dominio);
- Definición de las percepciones y acciones que cada agente puede realizar;
- Definición de creencias y objetivos.
- Definición de las relaciones de comunicación entre agentes (de orden);

Metodologías de desarrollo de SMA



Ingeniería del Software Orientada a Agentes

- Agent-Oriented Software Engineering (AOSE)
  - Gaia
  - MaSE
  - Agent UML
  - Prometheus
  - MASINA
- Ingeniería del Conocimiento Orientada a Agentes
  - MASCommonKADS
  - DESIRE
  - Cassiopeia
- Métodos formales orientados a agentes
  - Métodos formales en AOSE
  - Especificación en Z

MASINA

Metodología que permite especificar Sistemas Multi-agentes, la cual es una extensión de MAS-CommonKADS.

Fases

Conceptualización

- Casos de uso (descripción de acciones necesarias para producir un resultado útil)
- Actores (roles desempeñados por alguna persona, una pieza de software, u otro sistema)

Análisis y Diseño

- Modelos para describir los agentes del sistema, sus tareas, su organización y los medios de comunicación.
- Diseño técnico del sistema (modelo de implementación).

Codificación y prueba

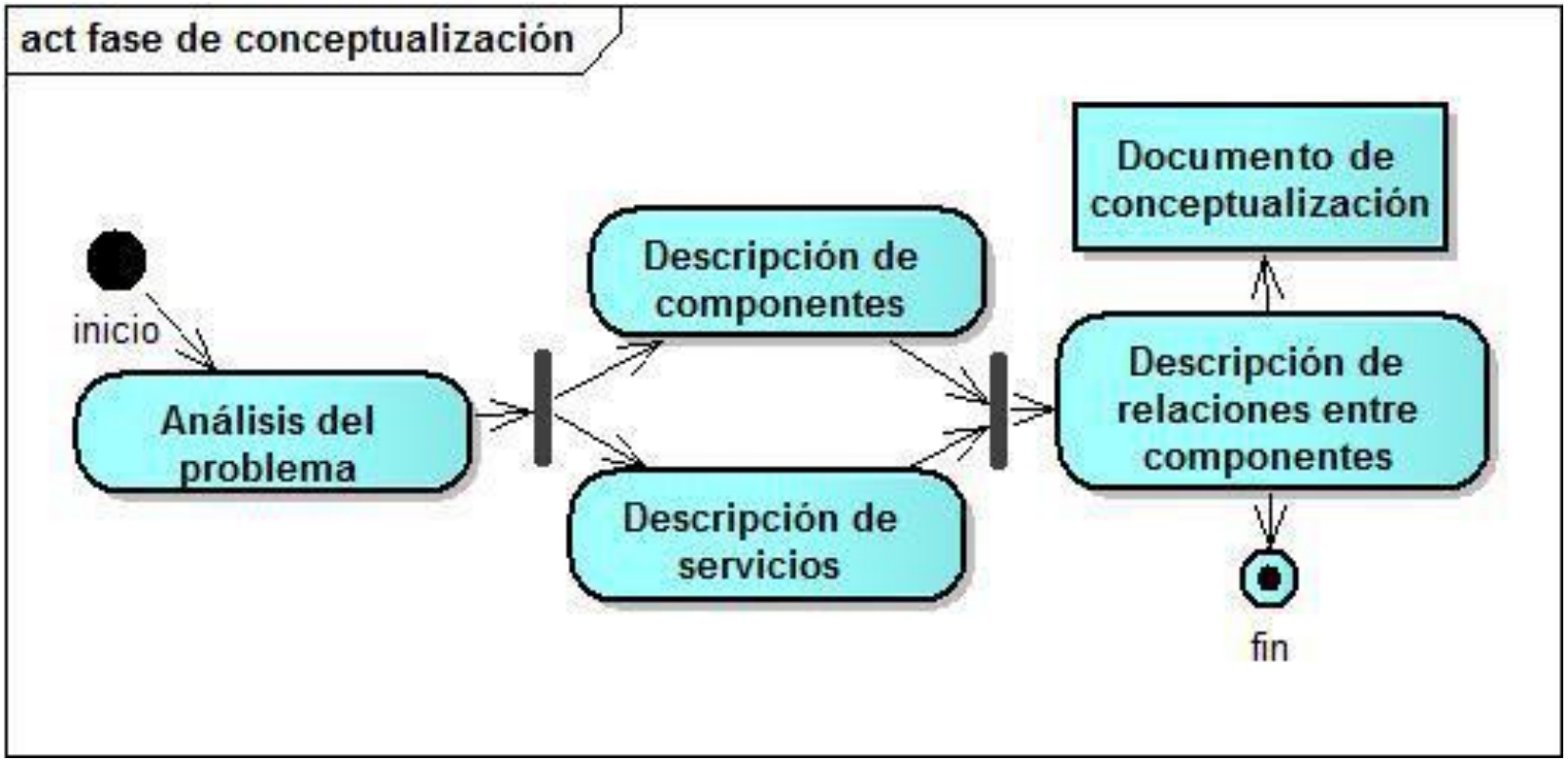
Integración

Operación y mantenimiento

UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

MASINA

Fase de Conceptualización



UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

MASINA

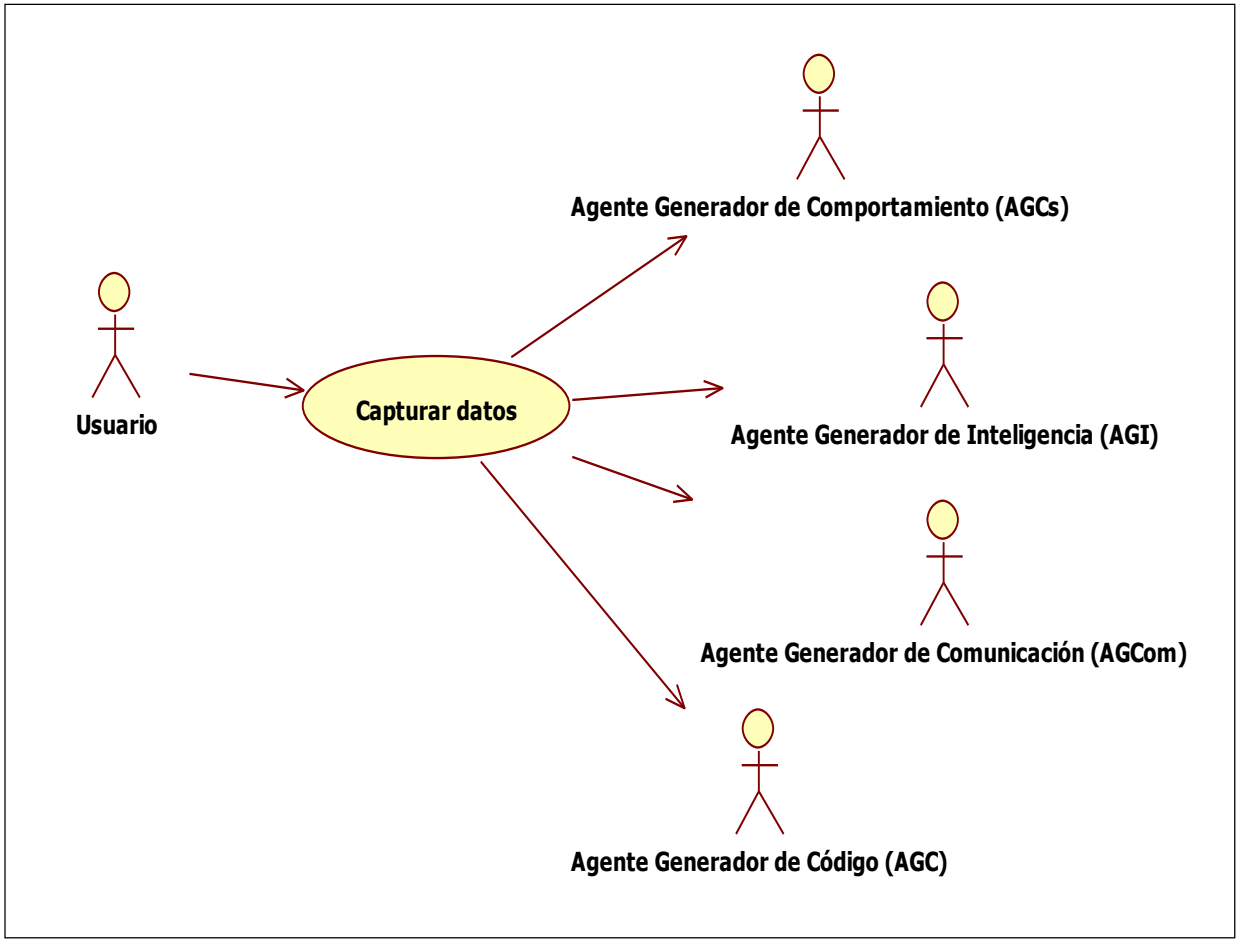
- El producto de esta fase es un **documento de conceptualización**:
  - La descripción de los componentes (agentes) del sistema,
  - La especificación de los servicios y de las actividades para prestar los servicios ofrecidos por cada componente del sistema
  - La descripción general de las relaciones entre los componentes del sistema.
- Para eso se usan: **casos de uso y diagramas de actividades**

UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

MASINA

Fase de Conceptualización

Casos de uso

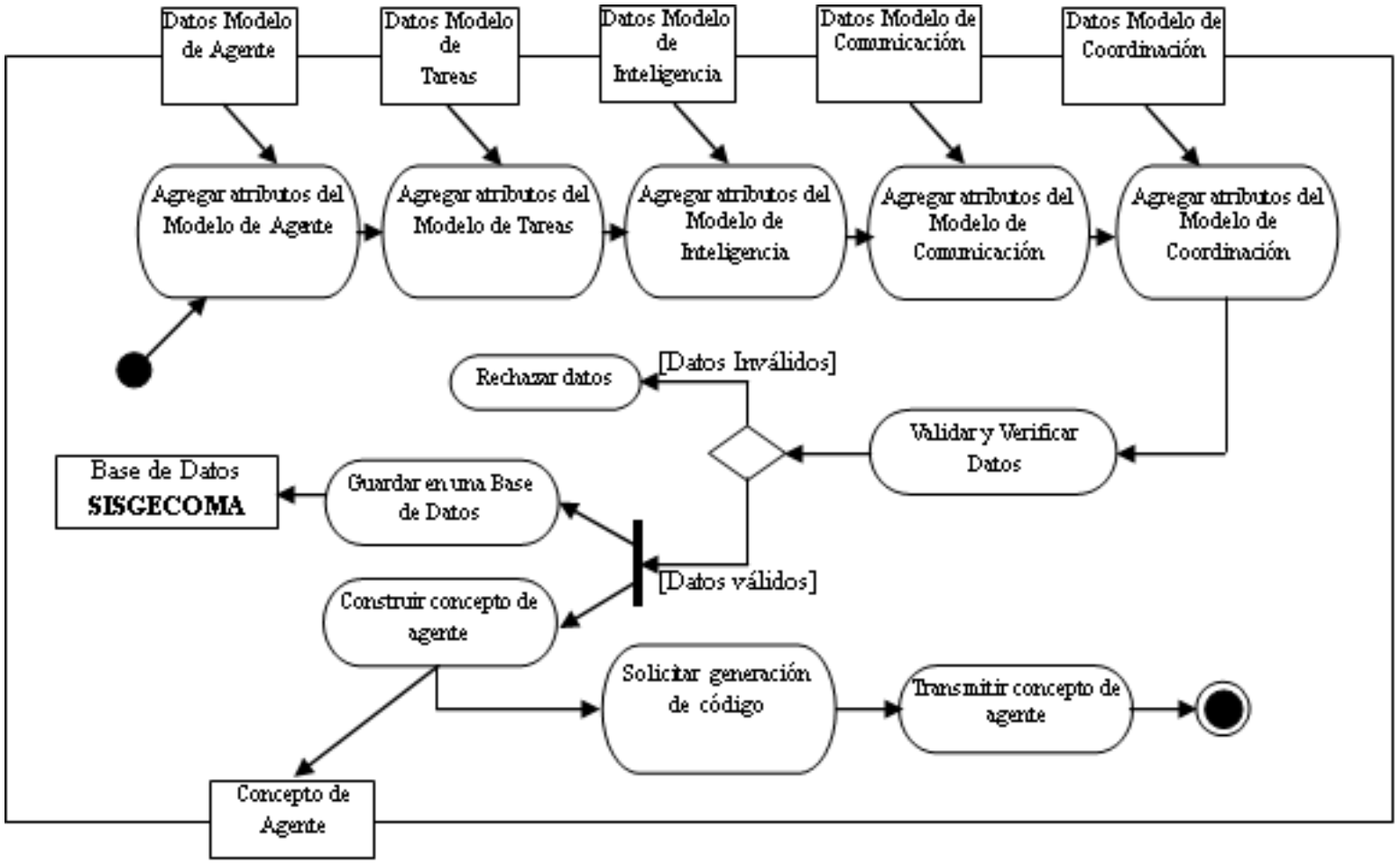


UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

MASINA

Fase de Conceptualización

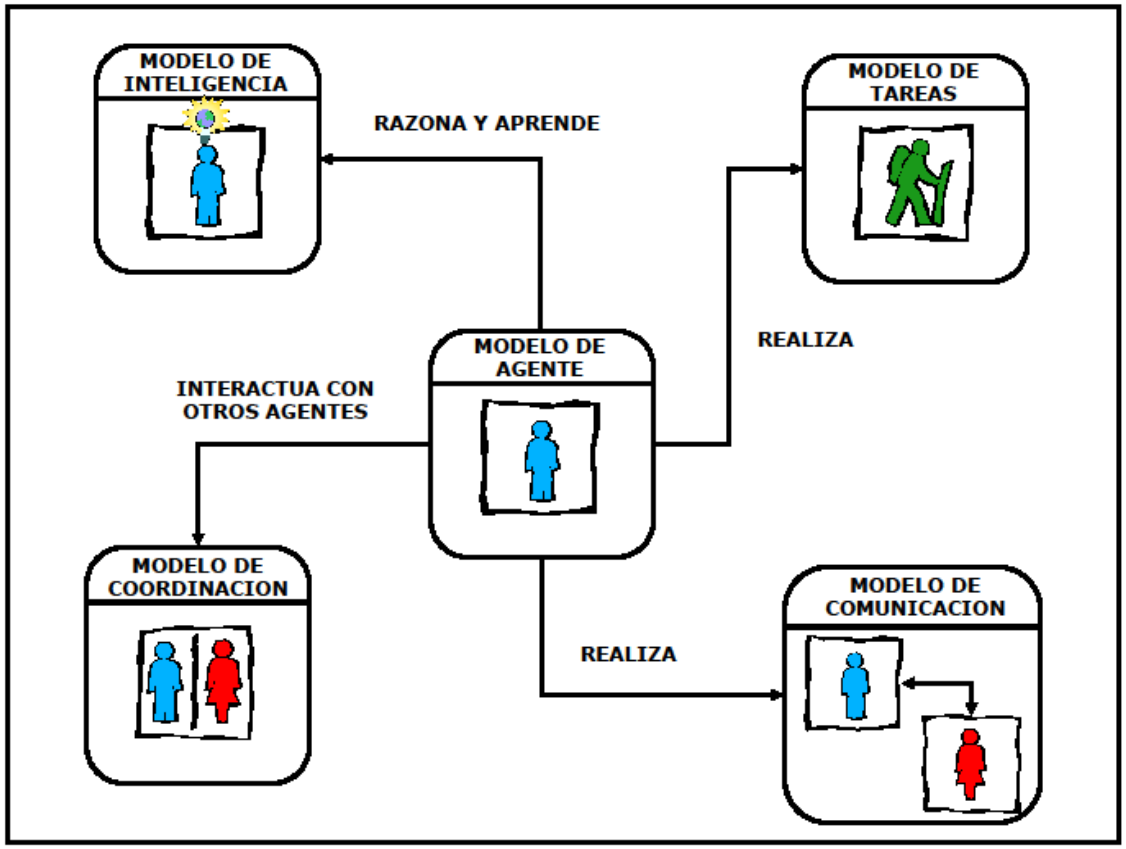
Diagrama de actividades



UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

MASINA

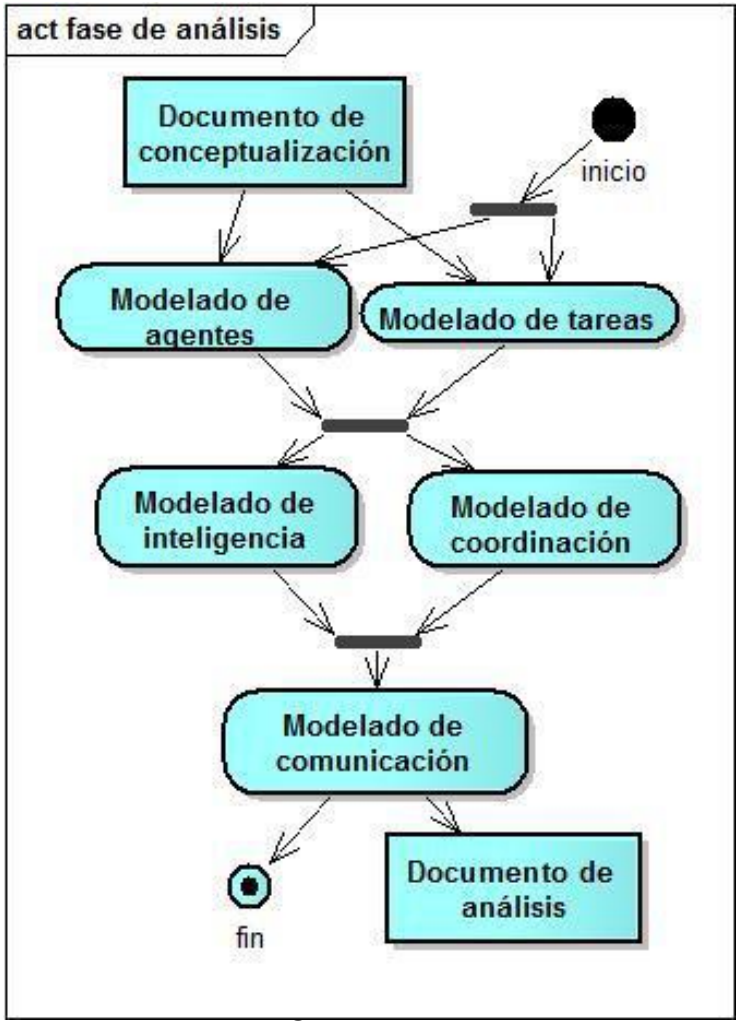
Fase de Análisis



UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

MASINA

Fase de Análisis



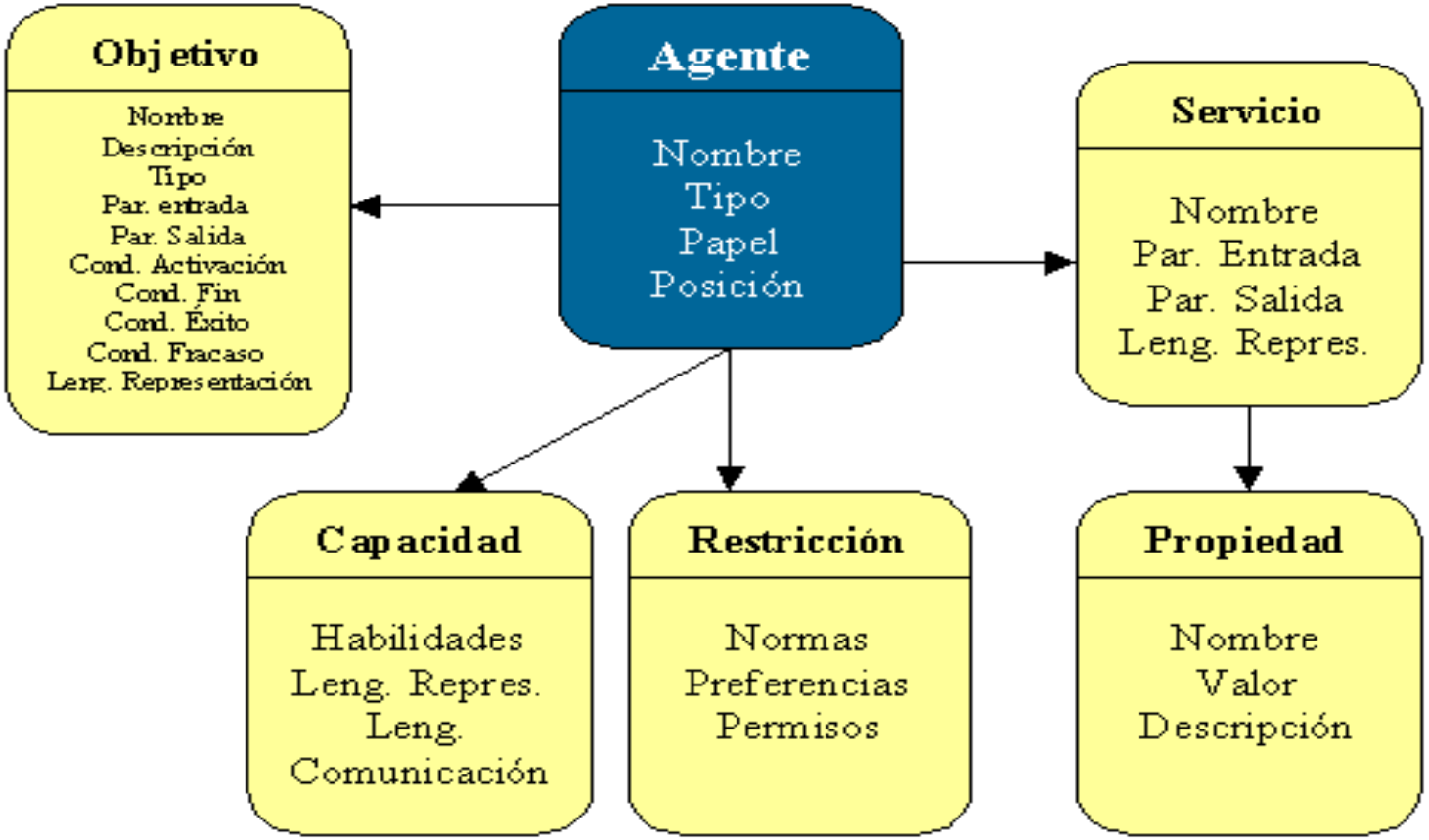
UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

MASINA

- El producto de esta fase es un **documento de análisis** contentivo de:
  - Modelo de agentes,
  - Modelo de tareas,
  - Tabla relación Agentes-Tareas
  - **Modelo de inteligencia**
  - **Modelo de Coordinación/conversación**
  - **Diagrama de Interacción**
  - **Modelo de comunicación.**

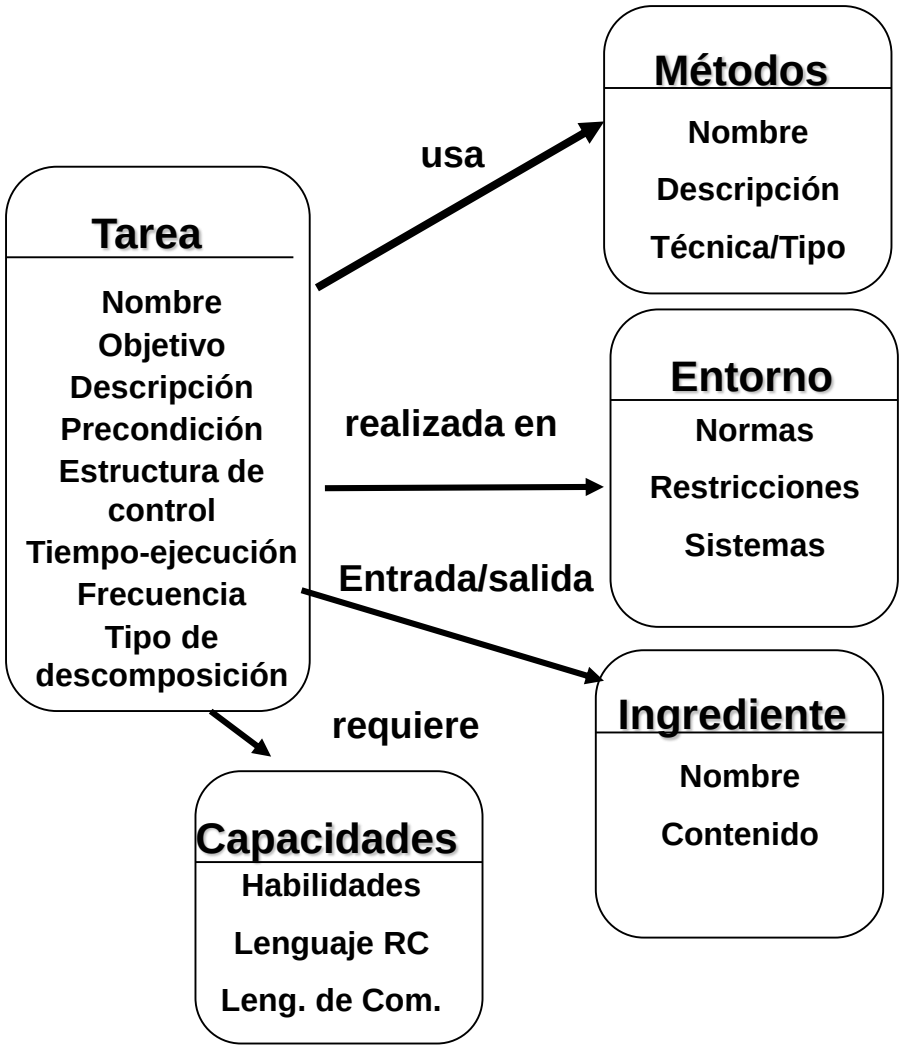
UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

Modelo de Agente



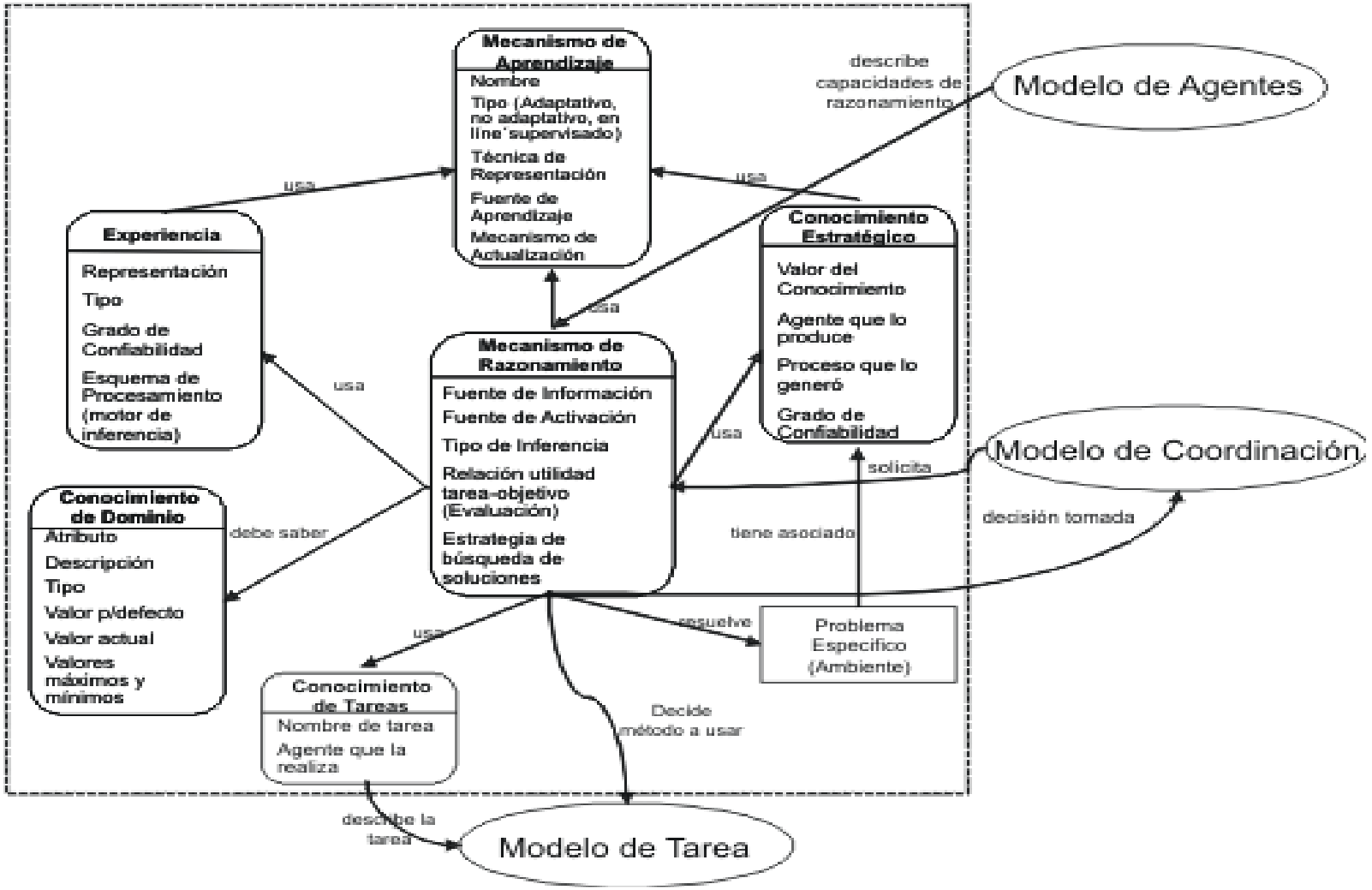
UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

Modelo de Tareas

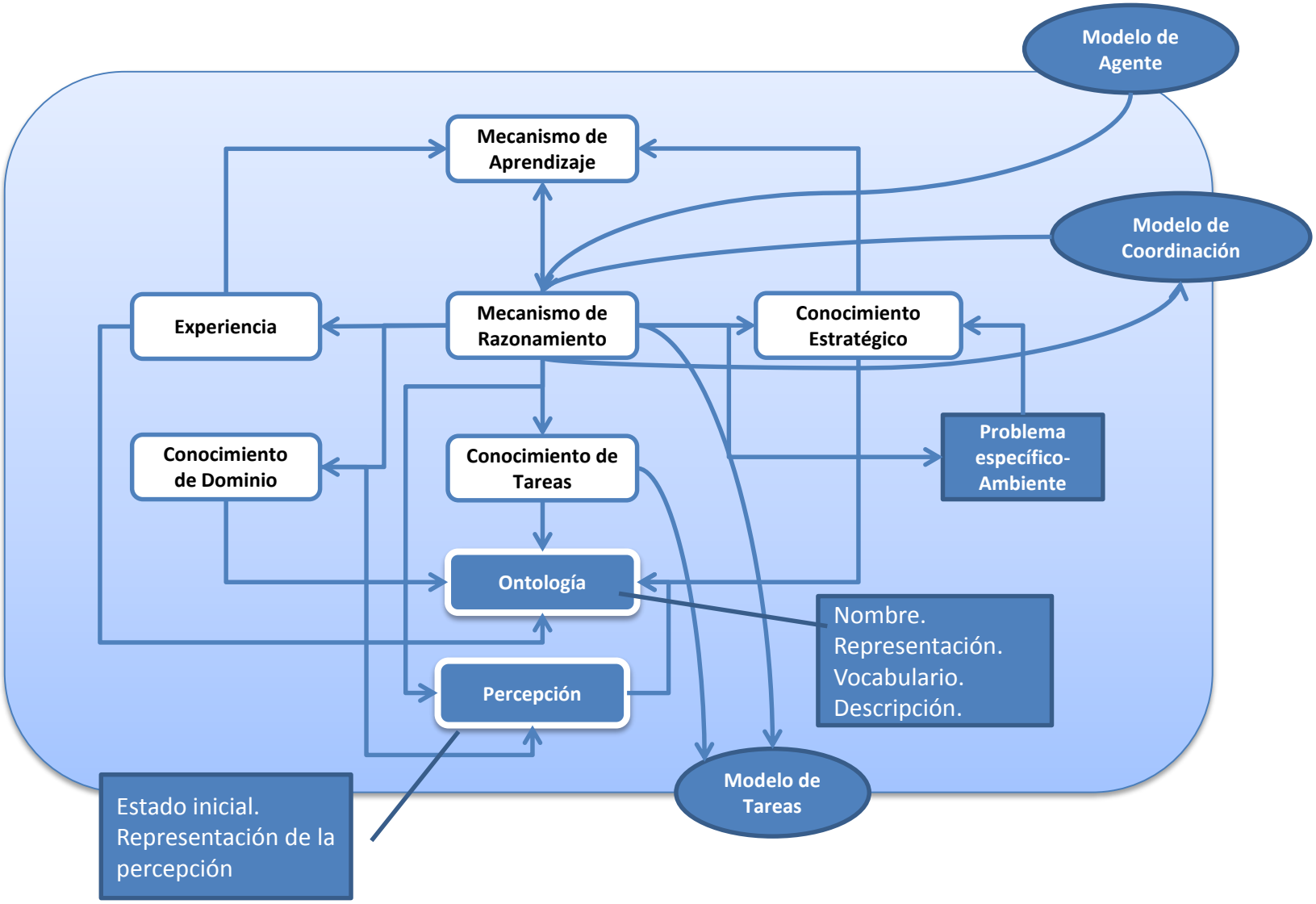


UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

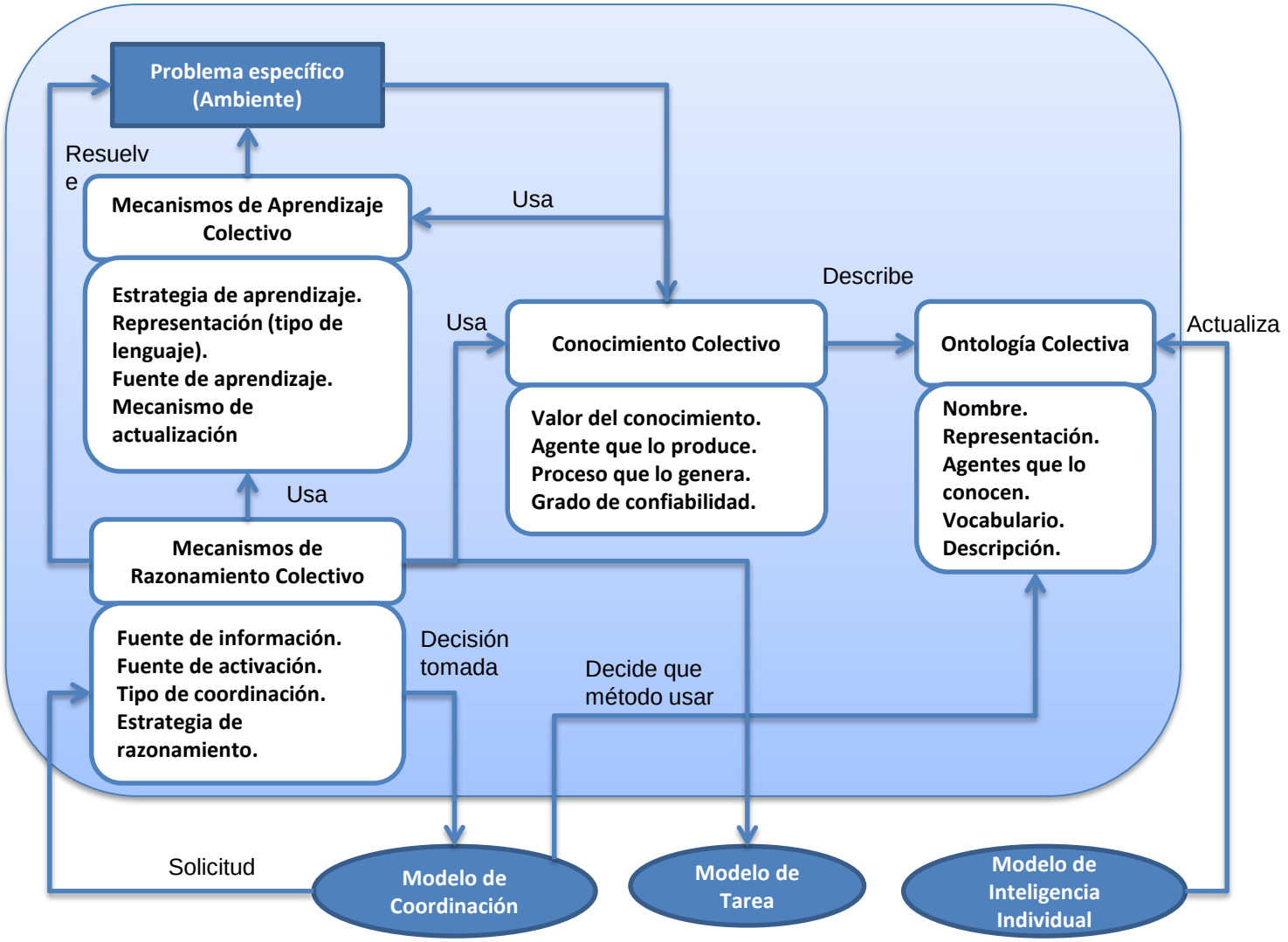
Modelo de Inteligencia



Modelo de Inteligencia individual de MASINA

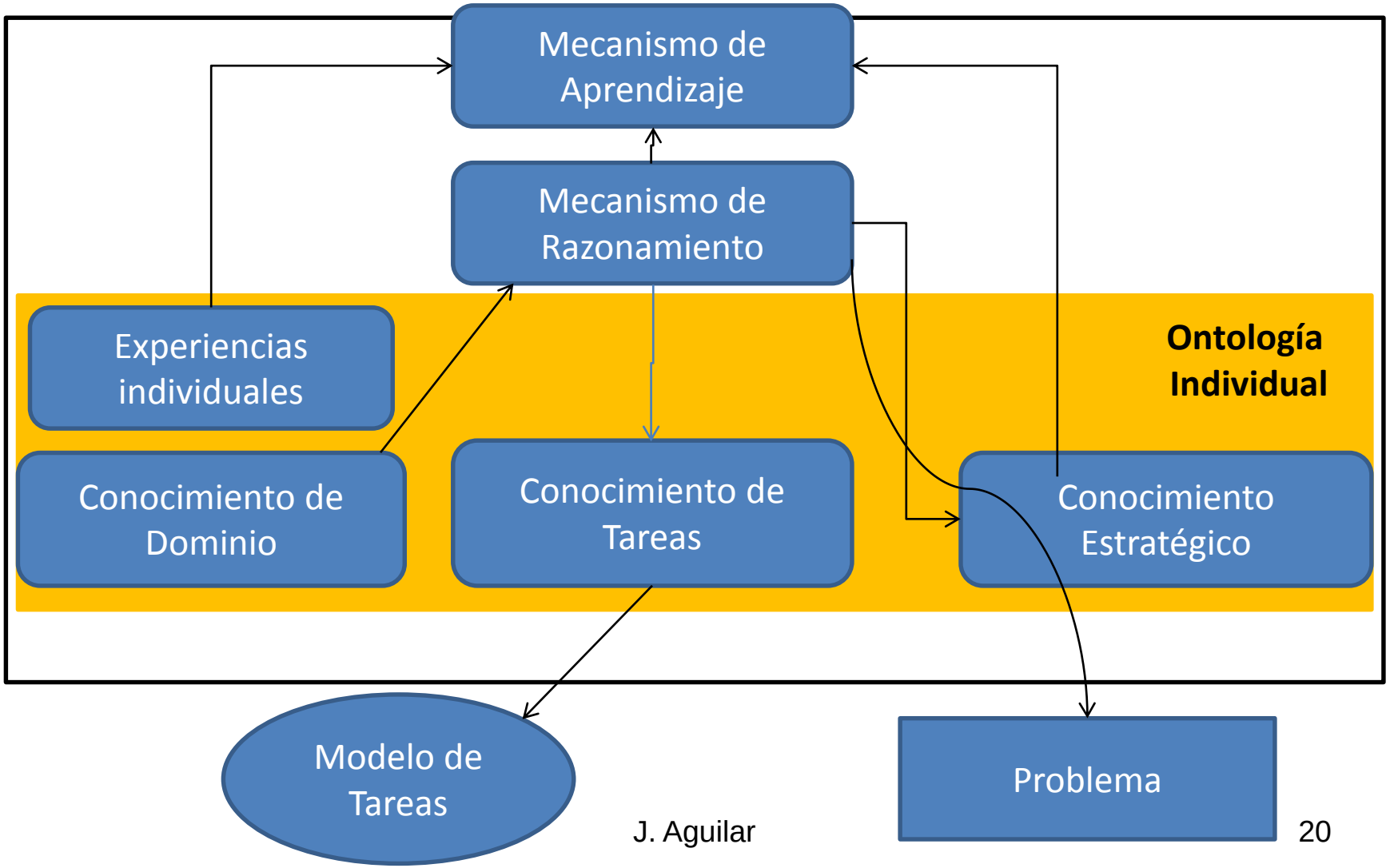


Modelo de Inteligencia Colectiva de MASINA



UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

Modelo de Inteligencia



UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

Modelo de Inteligencia

Mecanismo de Aprendizaje	
Nombre	Nombre del mecanismo de aprendizaje
Tipo	Supervisado, no supervisado, reforzamiento
Técnica de representación	Redes neuronales, Arboles de decisión, reglas, clasificación, redes bayesianas, minería de datos, etc.
Fuente de aprendizaje	Origen de la información que se usa para aprender. la misma normalmente se divide en una parte para entrenar y otra para probar
Mecanismo de actualización	hebbiano, corrección de error, etc.

Mecanismo de Razonamiento	
Fuente de información	Origen de la información utilizada para el proceso de razonamiento.
Fuente de alimentación	Tareas que requieren el proceso de razonamiento.
Técnica de inferencia	Lógica difusa, reglas, lógica de predicados, etc.
Lenguaje de representación de conocimiento	OWL, RDF, etc.
Relación tarea-inferencia (Resultado esperado)	Relación que existe entre la tarea que activo el proceso de razonamiento y el resultado obtenido.
Estrategias de razonamiento	Deductivo, inductivo, abductivo

UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

Modelo de Inteligencia

Experiencias (ontología histórica)	
Descripción	La experiencia en si del agente
Caracterización	Conceptos, relaciones, propiedades, etc
Fuente	Basada en casos, empírico, imitación, sus actividades, etc.
Valores por omisión (para cada concepto)	Valor inicial
Valores max y min	Valores máximos y mínimos que puede tomar la variable, rangos o bandas.
Momento (tiempo)	Manera como cambia en el tiempo.

Conocimiento de Dominio (ontología de dominio)	
Descripción	El conocimiento en si de un ámbito dado
Caracterización	Conceptos, relaciones, propiedades, etc.
Fuente	Taxonomía de un.área de conocimiento
Valores por omisión (para cada concepto)	Valor inicial
Valores max y min	Valores máximos y mínimos que puede tomar la variable, rangos o bandas.

UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

Modelo de Inteligencia

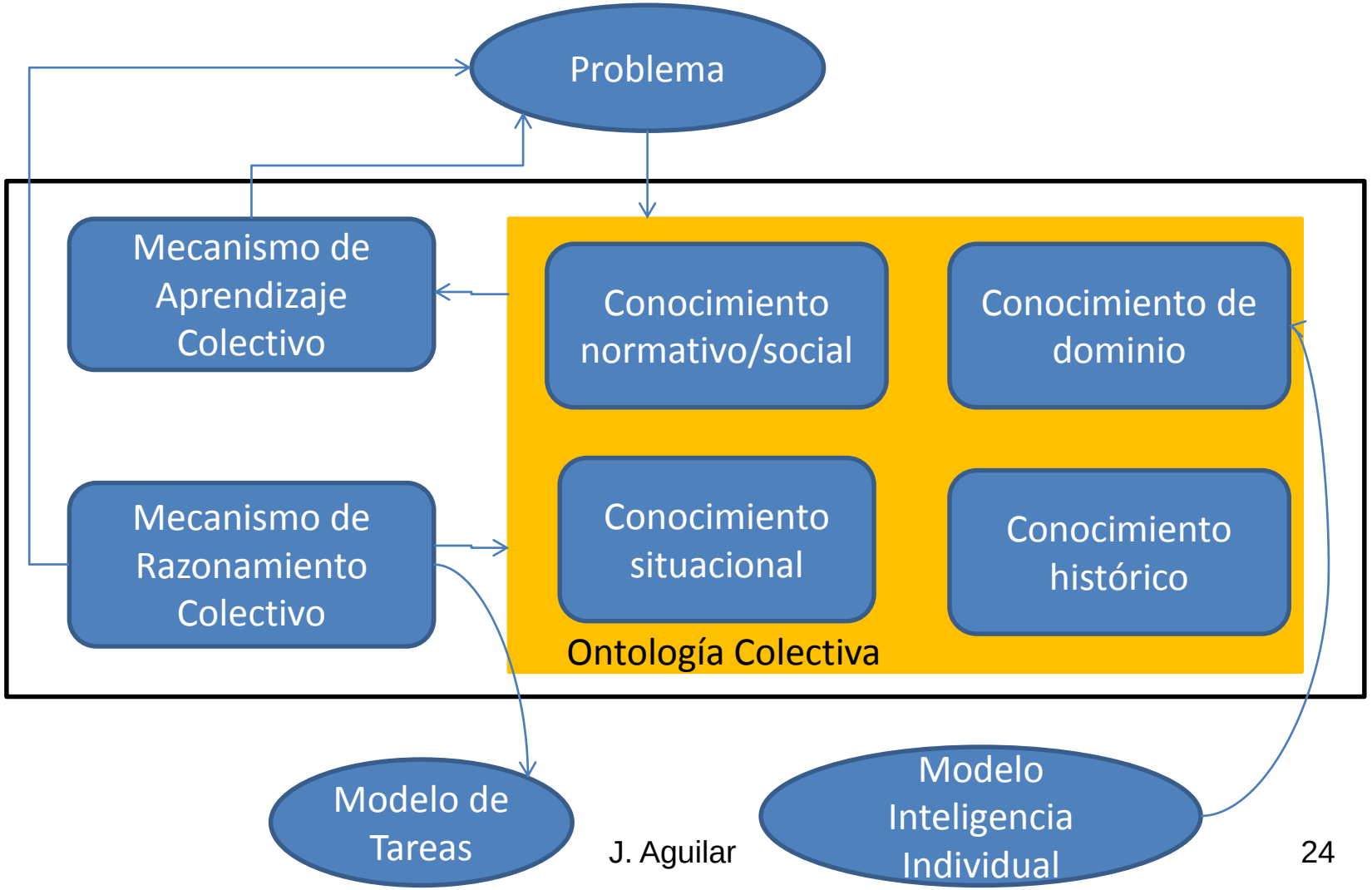
Ontología de contextualización o situacional: está compuesto por dos tipos de conocimiento

Conocimiento estratégico	
Valor del conocimiento	Pasos a seguir para resolver un problema, característica especificas de un ambiente, etc.
Agente o ambiente que lo produce	Agente que produce el conocimiento.
Proceso que lo genero (cuando sea el caso)	Proceso que genero el conocimiento.
Grado de confiabilidad	Nivel de certitud del conocimiento adquirido
Caracterización	Conceptos, relaciones, propiedades, etc.
Valores por omisión (para cada concepto)	Valor inicial
Valores max y min	Valores máximos y mínimos que puede tomar la variable, rangos o bandas.

Conocimiento de tareas	
Nombre de la tarea	Nombre de la tarea a realizar.
Agente que la realiza	Agente que origina la tarea a realizar.

Marco Ontológico del individuo	
Nombre	Nombre de la ontología
Descripción	Descripción de “lo que conoce el agente”
Ontologías que la integran	Listado de conocimiento (ontologías) que lo componen

Modelo de Inteligencia Colectiva



Modelo de Inteligencia Colectiva

Mecanismo de Aprendizaje Colectivo	
Nombre	Nombre del mecanismo de aprendizaje
Tipo	Supervisado, no supervisado, reforzamiento
Técnica de representación	Redes neuronales, Arboles de decisión, reglas, clasificación, redes bayesianas, minería de datos, etc.
Fuente de aprendizaje	Origen de la información que se usa para aprender. la misma normalmente se divide en una parte para entrenar y otra para probar
Mecanismo de actualización	hebbiano, corrección de error, etc.

Mecanismo de Razonamiento Colectivo	
Fuente de información	Origen de la información utilizada para el proceso de razonamiento.
Fuente de alimentación	Tareas que requieren el proceso de razonamiento.
Técnica de inferencia	Lógica difusa, reglas, lógica de predicados, etc.
Lenguaje de representación de conocimiento	OWL, RDF, etc.
Relación tarea-inferencia (Resultado esperado)	Relación que existe entre la tarea que activo el proceso de razonamiento y el resultado obtenido.
Estrategias de razonamiento	Deductivo, inductivo, abductivo

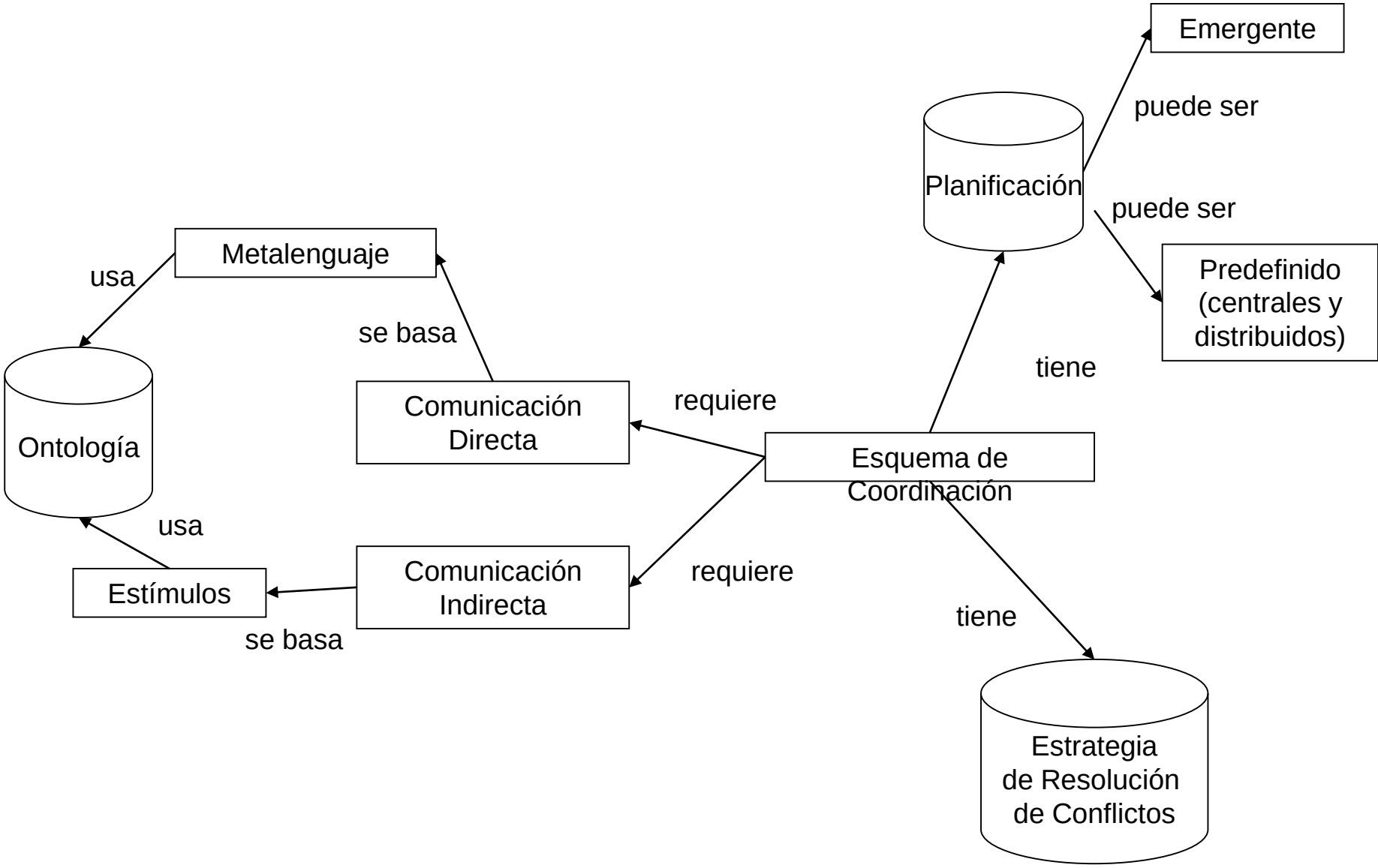
Modelo de Inteligencia Colectiva

Ontología situacional colectiva	
Descripción	Conocimiento colectivo
Caracterización	Conceptos, relaciones, propiedades, etc
Fuente	Quien la genera, quien la actualiza, en el SMA
Grado de confiabilidad	Nivel de certitud del conocimiento adquirido
Caracterización	Conceptos, relaciones, propiedades, etc.
Valores por omisión (para cada concepto)	Valor inicial
Valores max y min	Valores máximos y mínimos que puede tomar la variable, rangos o bandas.

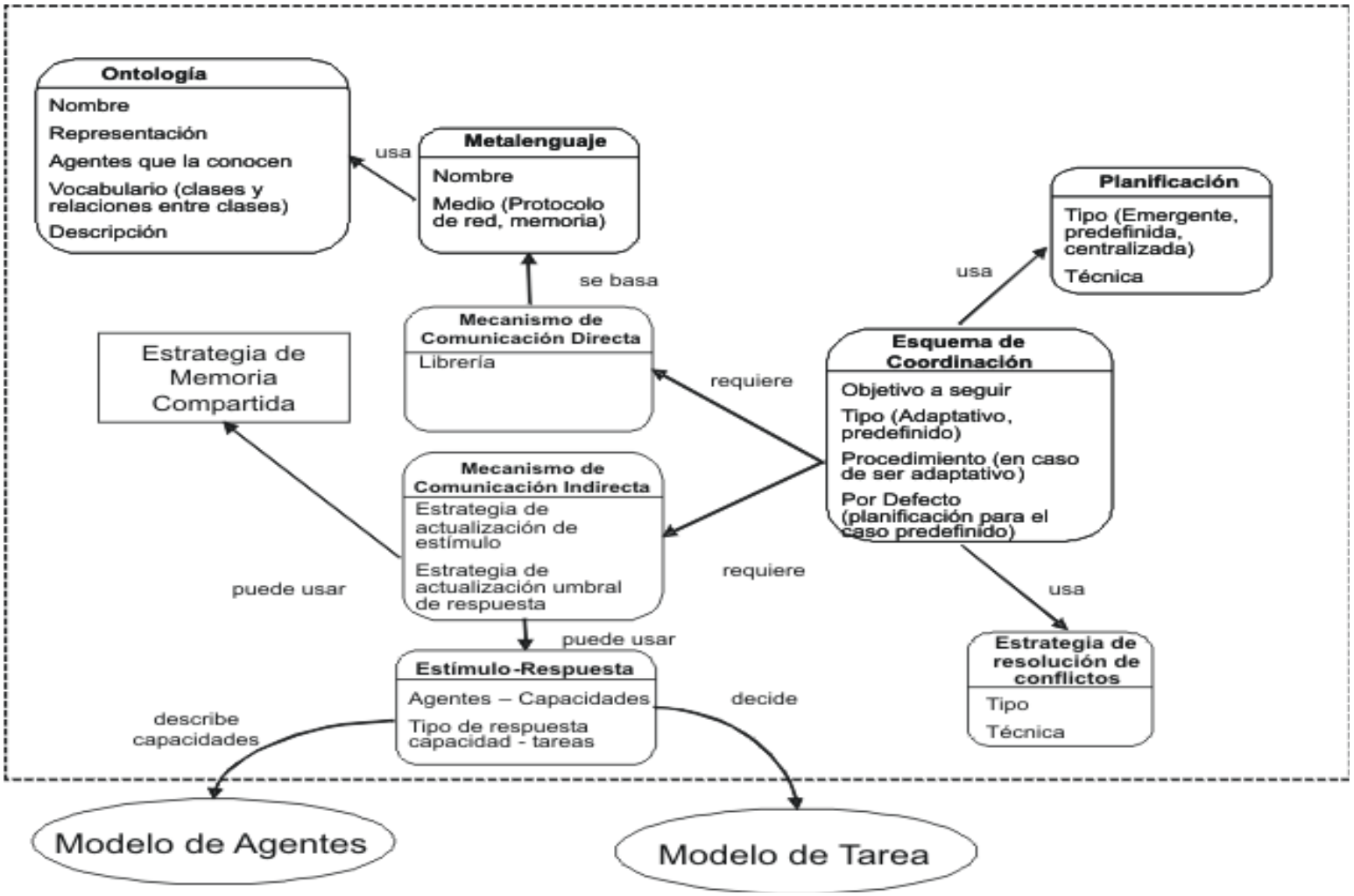
Marco Ontológico colectivo	
Nombre	Nombre de la ontología
Objetivos colectivos para lo que se usa	Qué tareas colectivas lo usan para alcanzar objetivos grupales
Descripción	Descripción de “lo que conoce el agente”
Ontologias que la integran	Listado de ontologías situacionales que lo componen
Agentes que la conocen	Miembros del SMA que usan esa ontología en sus tareas colectivas

UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

Problema de Coordinación



Modelo de Coordinación



Modelo de Coordinación y Comunicación

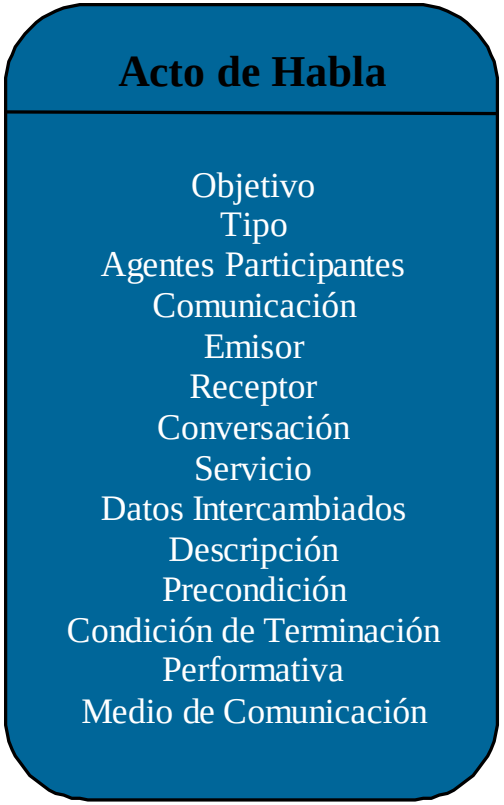
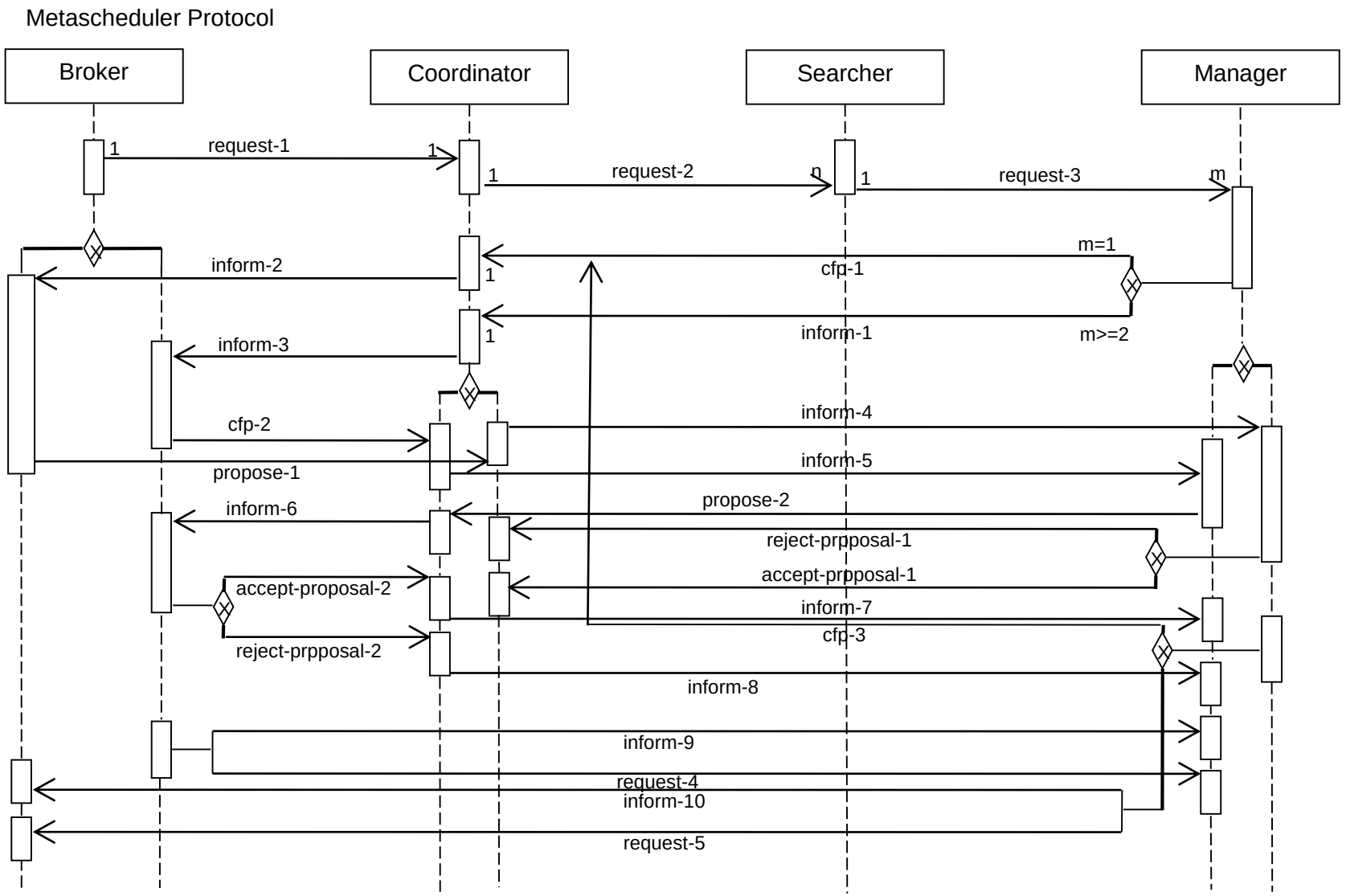


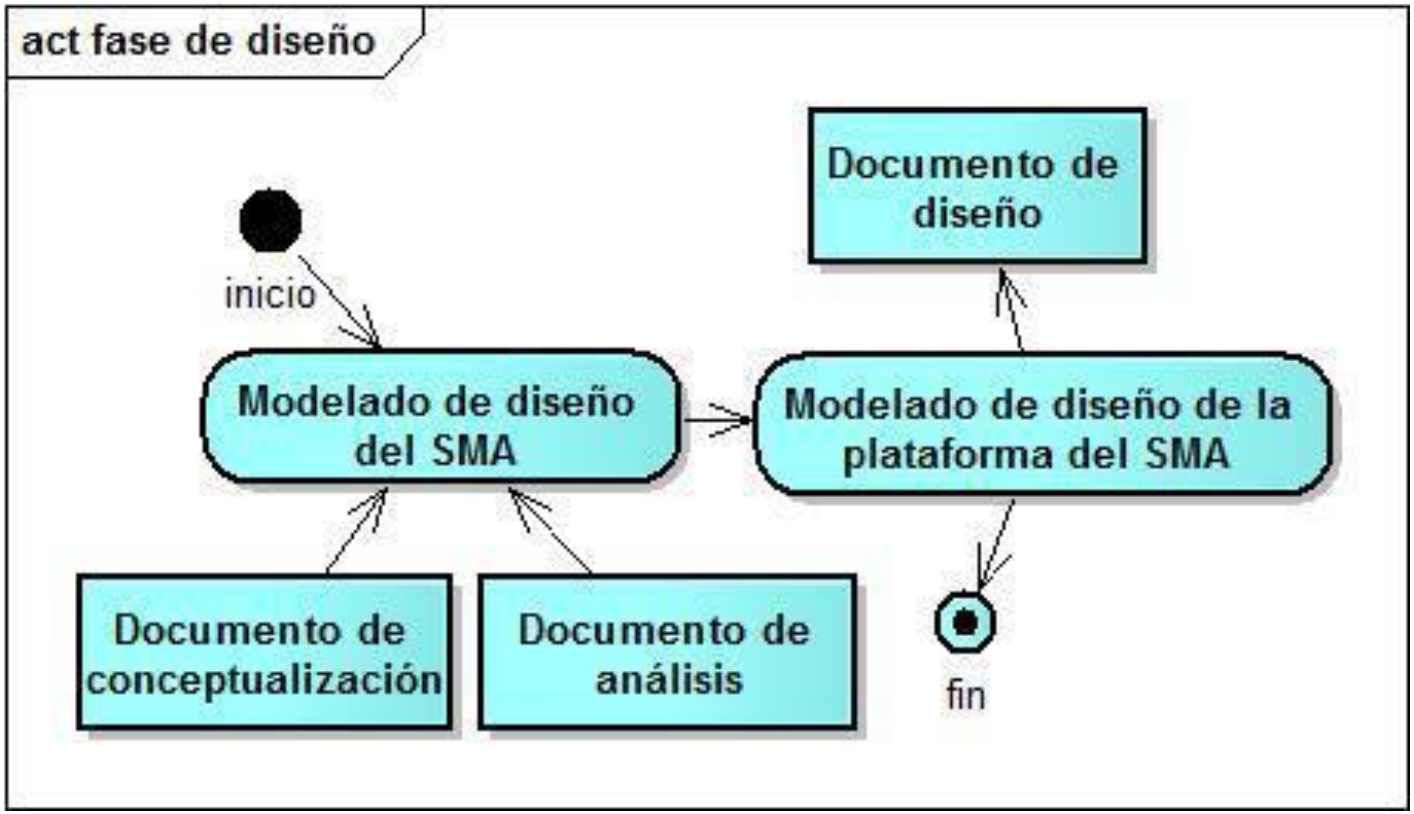
Diagrama Interacción: Protocolo de interacción FIPA



UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

MASINA

Fase de diseño



UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

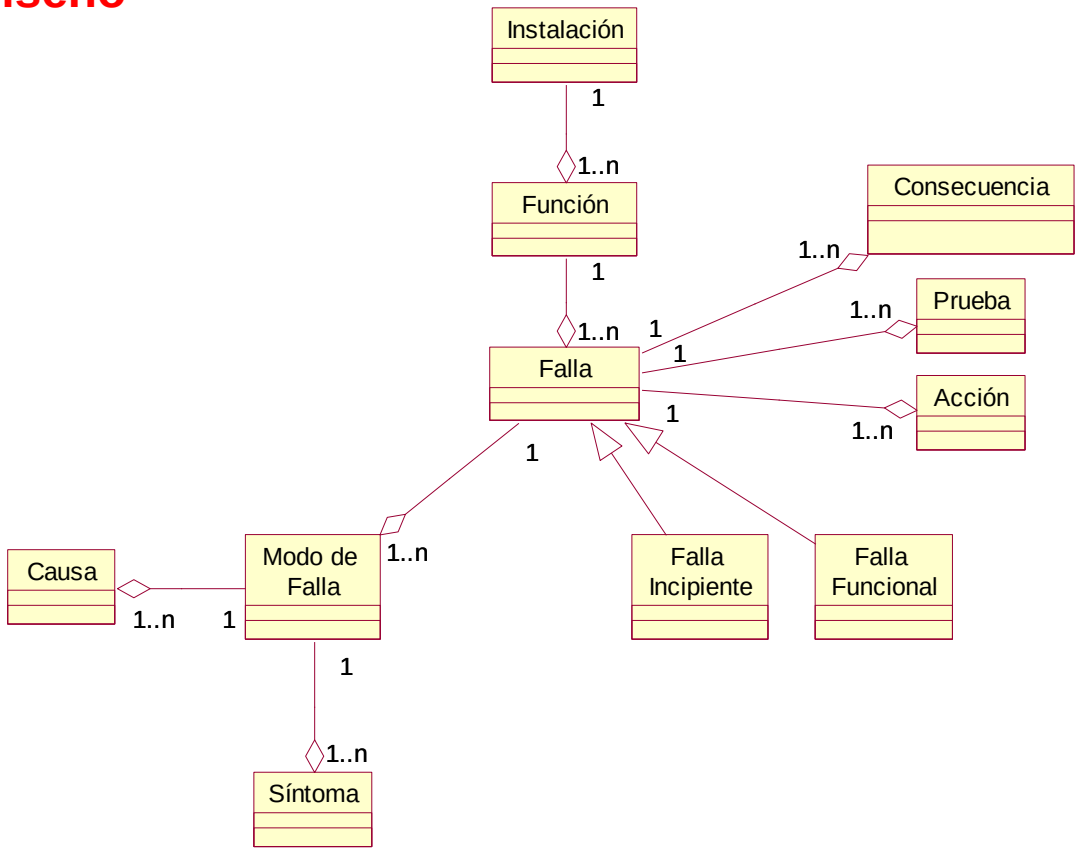
MASINA

- El producto de esta fase es un **documento que especifica**:
  - El universo de clases del sistema.
  - Para cada una de las clases descritas en el universo de clases su especificación formal.
  - Para cada uno de los métodos que componen las clases su especificación formal.

Nosotros particularmente usamos TDSO

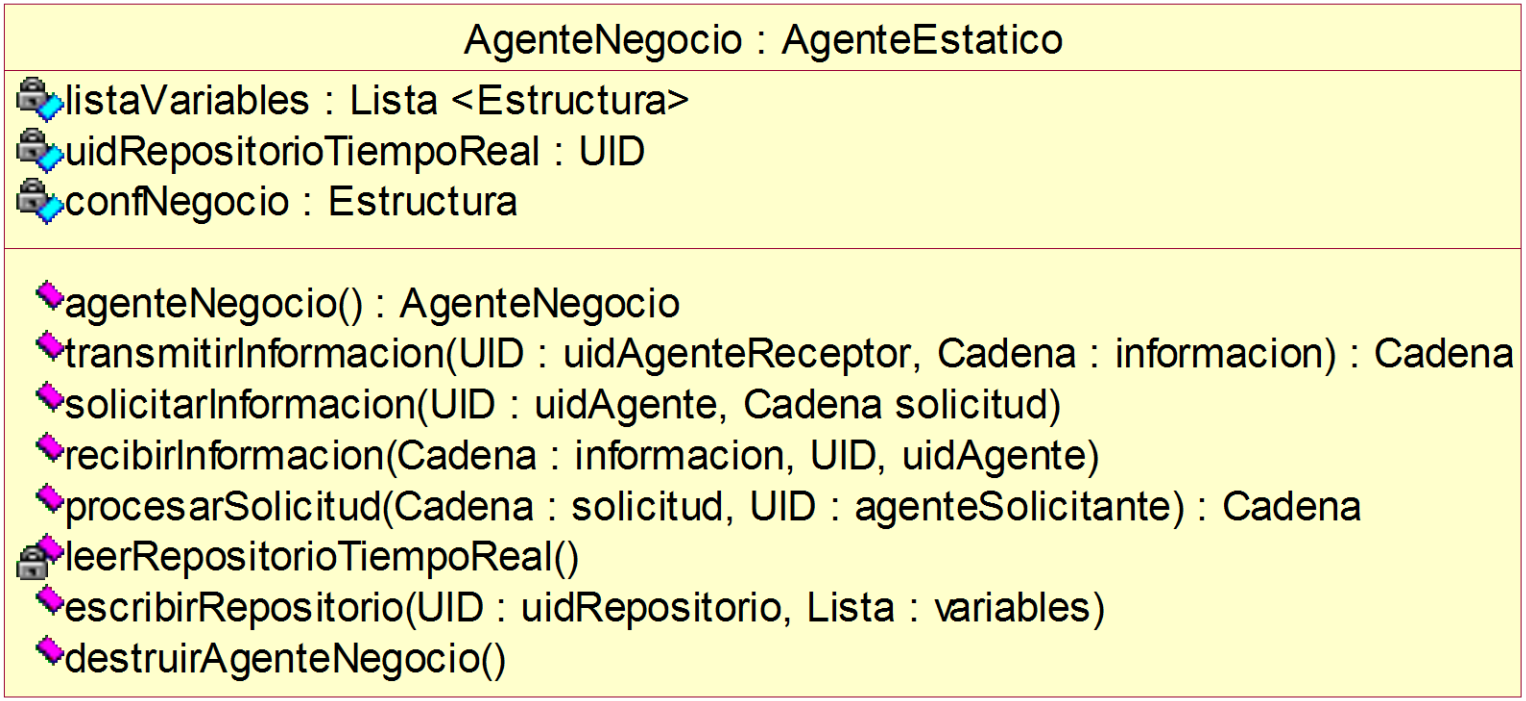
MASINA

Modelo de Diseño



UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

Diagrama de clases del Agente de Negocio



Definición del universo de clases y tipo de datos abstractos (TDAs)

20/11/07		Versión 1.0
Universo de clases y TDAs AgenteNegocio {Colección de clases y TDAs requerida para implantar el Agente de Negocio}		
1 2 3 4 5 6	Agente Negocio ( ) Cadena UID Logico TablaTiempoReal Estructura	*AgenteNegocio ( )*: clase que permite la creación de un Agente de Negocio. *Cadena*: TDA cadena de caracteres de longitud variable. *Entero*: valor entero. *UID*: tipo entero que representa un identificador único en el sistema multiagente. *Logico*: tipo lógico, conformado por los valores cierto y falso. *TablaTiempoReal:* TDA que contiene los datos del proceso real. *Estructura:* tipo de dato que contiene campos asociados a información configurada.

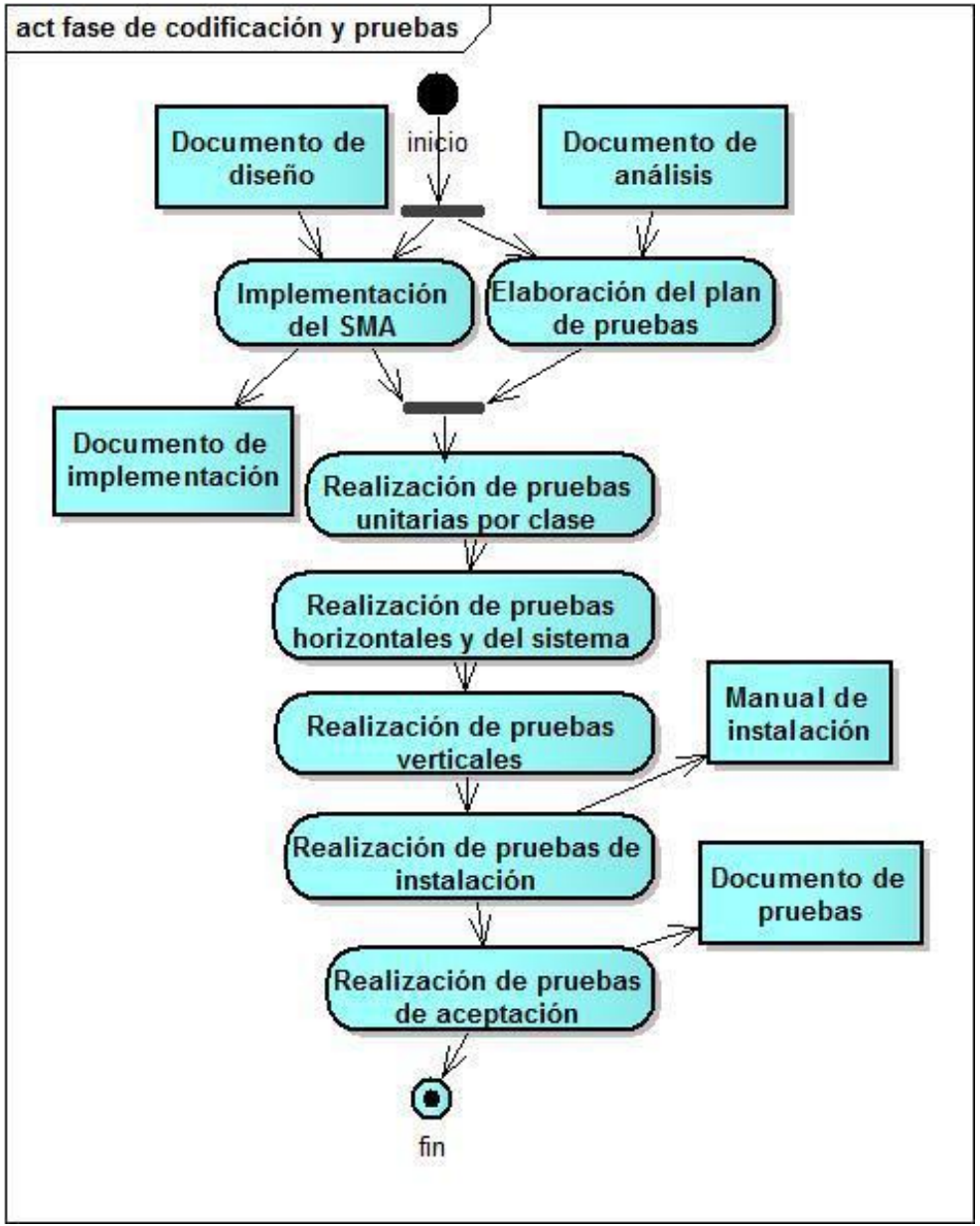
Definición del universo de clases y tipo de datos abstractos (TDAs)

20/11/07		Versión 1.0
1,1 (Constructor, Público) **agenteNegocio( ): AgenteNegocio** {Crea un Agente del tipo AgenteNegocio} {**pre:** existencia de memoria y Negocio.dat distinto de Null}		{**pos:** se crean componentes del agente o error}
1 2 3 4	abrirArchivoConf("Negocio.dat") Leer(Negocio.dat, ConfNegocio) uidRepositorioTiempoReal = asignaUid( ) listaVariables: Lista<TablaTiempoReal>	*asignaUid()*: método que asocia el AN con un único repositorio datos de tiempo real *listaVariables:* Lista que contiene los datos del proceso real
1 2	agenteNegocio x ⇒ se creó el AN agenteNegocio x ⇒ error	Se instancia el agente x, si éste se puede crear hay éxito, por el contrario hay error.

UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

MASINA

Fase de Codificación Y pruebas



UNIVERSIDAD DE LOS ANDES
MERIDA VENEZUELA

MASINA

- El producto principal de esta fase consiste en un sistema de ingeniería orientado a agentes

Resumen de otras metodologías

- ***GAIA*.** se centra en la idea de que la construcción de sistemas basados en agente es un proceso de diseño organizacional.
- ***DESIRE***. La principal contribución es que constituye un entorno lo suficientemente expresivo para permitir a los diseñadores de sistemas multiagente centrarse en el diseño conceptual y la especificación de su sistema.
- ***MASSIVE*** (Multi-Agent SystemS Iterative View Engineering) está constituido por un conjunto de vistas diferentes del sistema a construir donde el desarrollo que se sigue consiste en una visión iterativa del mismo.
- ***AUML*.** intenta adaptar herramientas de desarrollo ya existentes y que están teniendo éxito para aplicaciones industriales reales, como es el caso de UML, tratando de orientarlas hacia el campo de los agentes.

Resumen de otras metodologías

- ***Tropos*.** metodología de desarrollo de software basado en agentes mediante extensiones de UML y empleando un entorno de modelado denominado *i** . El concepto principal es el de actor, así como sus objetivos y posibles dependencias con otros actores.
- ***MaSE*** (Multiagent System Engineering). Es una metodología que parte de la especificación del mismo hasta su implementación. Lenguaje de especificación basado en UML+OCL
- ***MESSAGE*** (Methodology for Engineering Systems of Software Agents): incorpora técnicas de ingeniería del software cubriendo el análisis y diseño de sistemas multiagente. La metodología provee un lenguaje, un método y unas guías de cómo aplicar la metodología, centrándose en las fases de análisis y diseño

Gaia

- Su intención es la de permitir a un analista ir sistemáticamente **desde los requisitos a un diseño suficientemente detallado como para implementar directamente.**

- Los principales conceptos que aparecen en la metodología se dividen en dos:
  - abstractos y
  - concretos

- **Las entidades abstractas** son aquellas que son empleadas durante el análisis para la conceptualización del sistema.
- **Las entidades concretas** son empleadas en el proceso de diseño.

Gaia

- La entidad más abstracta de un sistema es la “sociedad” u “organización”.

- Idea: ver un sistema informático como un conjunto de roles teniendo en cuenta una visión organizacional del mundo.

- **Organización =** una colección de roles que toman parte en patrones sistemáticos de interacción con otros roles

- **Roles =** pueden ser instancias de agentes; no fijos necesariamente

- Roles consisten en 4 atributos: responsabilidades, permisos, actividades y protocolos

Gaia

- Un **rol** es asociado a:
  - **un conjunto de permisos**, derechos asociados al rol, identifican los recursos que están disponibles al rol para realizar sus responsabilidades.
  - **un conjunto de protocolos**, los cuales definen la manera de interactuar del rol con otros roles

- Las **responsabilidades** determinan la funcionalidad, podemos ver dos tipos:
  - **propiedades de viveza**: describen aquellos estados de los asuntos que un agente debe efectuar. Decir que "algo será hecho"
  - **propiedades de seguridad**: son invariantes. condiciones de seguridad

Gaia

- En el **Proceso de *Análisis***:
  - Encontrar los roles en el sistema
  - Modelar las interacciones entre los roles
  - Para cada papel identificar y documentar los protocolos asociados

Role Schema: CoffeeFiller			
Description: This role involves ensuring that the coffee pot is kept filled, and informing the workers when fresh coffee has been brewed.			
Protocols and Activities: Fill, InformWorkers, CheckStock, AwaitEmpty			
Permissions:	reads	supplied *coffeeMaker*	// *name of coffee maker*
		coffeeStatus	// *full or empty*
	changes	*coffeeStock*	// *stock level of coffee*
Responsibilities Liveness: CoffeeFiller = (Fill. InformWorkers. CheckStock. AwaitEmpty)$^{\omega}$ safety: • $coffeeStock > 0$			

- En el **Proceso de *Diseño***:
  - Mapear los roles en tipos de agentes
  - Crear el número correcto de instancias de agente de cada tipo
  - Modelar los servicios necesarios para desarrollar el rol (modelo de servicios), examinando protocolos y propiedades de viveza y seguridad
  - Crear el modelo de conocimiento

Gaia

- El *análisis* incorpora los siguientes modelos:
  - **El Modelo del Prototipo del Rol:** consiste en identificar los roles y dar una breve descripción de los mismos
  - **El Modelo de Interacción:** captura la interacción secuencial entre los roles
  - **El Modelo de Roles Elaborado:** documenta los roles del sistema, sus protocolos y actividades, permisos y responsabilidades
- El *diseño* incluye:
  - **El Modelo de Agente,** en el que se los roles son agregados en tipos de agentes
  - **El Modelo de Servicios,** en el que se identifican todos los servicios (funciones) asociados a cada rol de agente
  - **El Modelo de Interacción,** que simplemente define la interacción existente entre los agentes

AUML

AUML = AGENT UNIFIED MODELING LANGUAGE
(www.auml.org)

"Reutilizar UML sólo donde tenga sentido"

- **UML es insuficiente para modelar sistemas multiagentes**
  - Comparados con los objetos, los agentes son activos, ya que actúan por razones que emergen de ellos mismos
  - Sus actividades incluyen objetivos y condiciones que guían la ejecución de las tareas definidas.
  - Toman la responsabilidad de sus necesidades.
  - Los agentes pueden actúar de igual forma solos o con otros agentes.
  - Forman una comunidad social de miembros
  - interdependientes que actúan de forma autónoma.

AUML

- No se trata de una metodología, sino de un **lenguaje para la documentación de modelos de sistemas.**

- AUML sintetiza el interés por disponer de **metodologías de desarrollo orientadas a agentes** con la aceptación de UML

- Presenta en la actualidad un conjunto de extensiones de UML para:
  - la especificación de protocolos de interacción de Agentes
  - la representación de estructuras sociales y organizativas entre agentes

AUML

- Inicialmente se identifican dos áreas para el desarrollo detallado de especificaciones:
  - **Diagramas de clases**
    - Especifican el comportamiento interno de un agente y su relación con el exterior usando diagramas de clases UML adaptados
    - Las clases de UML tienen “una extensión” que son los *AgentifiedClass, AgentClass* y un AgentRoleClass
  - **Diagramas de interacción**
    - Término genérico que se aplica a diversos tipos de diagramas centrados en la interacción entre agentes
    - Similar a los diagramas de interacción usados en UML
    - Varios usados: Diagramas de secuencia, Diagramas de descripción de interacciones, Diagramas de colaboración, diagrama de comunicación, Diagramas de estado

AUML

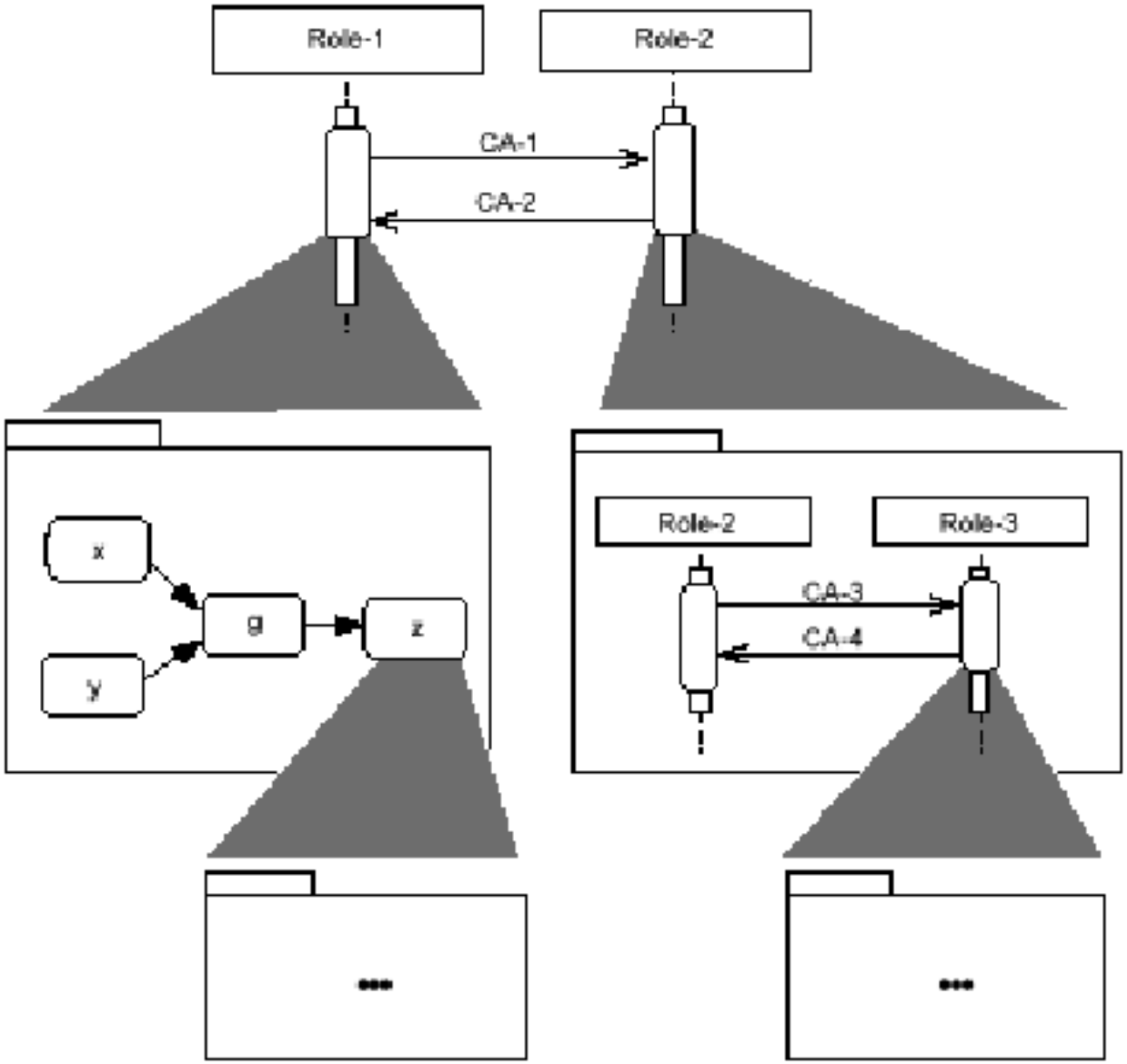
AUML adopta un enfoque por capas de protocolos:

Nivel 1 - Representar el protocolo general (diagramas de paquetes, etc)

Nivel 2- Representar las interacciones entre los agentes (diagramas de secuencia, colaboración)

Nivel 3- Representar la parte del Agente interna (diagramas de actividad y estados, de clases)

AUML: enfoque por capas de protocolos

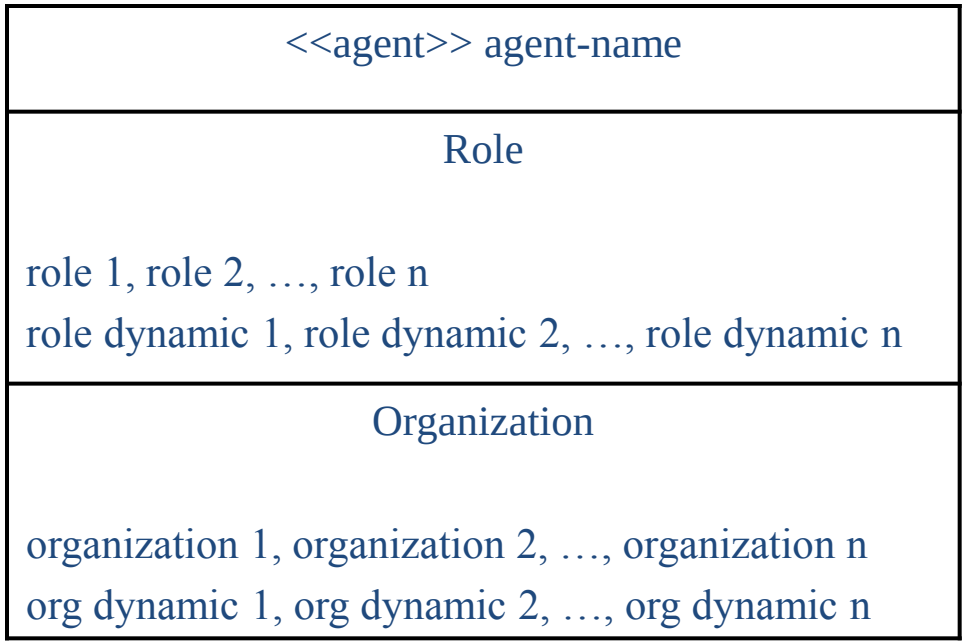


Representación de agentes y UML

- Identificador
  - Identifica a cada agente en un SMA
- Rol
  - define el comportamiento de un agente en la sociedad (es. Vendedor, etc.)
- Organización
  - Define las relaciones entre los roles (como en los humanos o animales: jerarquías, grupos de interés, etc.
- Capacidad
  - Especifica que un agente puede hacer y bajo que condiciones
  -
- Servicio
  - Describes la actividad que un agente púede realizar y proveer a otros agentes

Representación de agentes

Diagrama de clases UML se puede usar para representar parte estática de un agente.



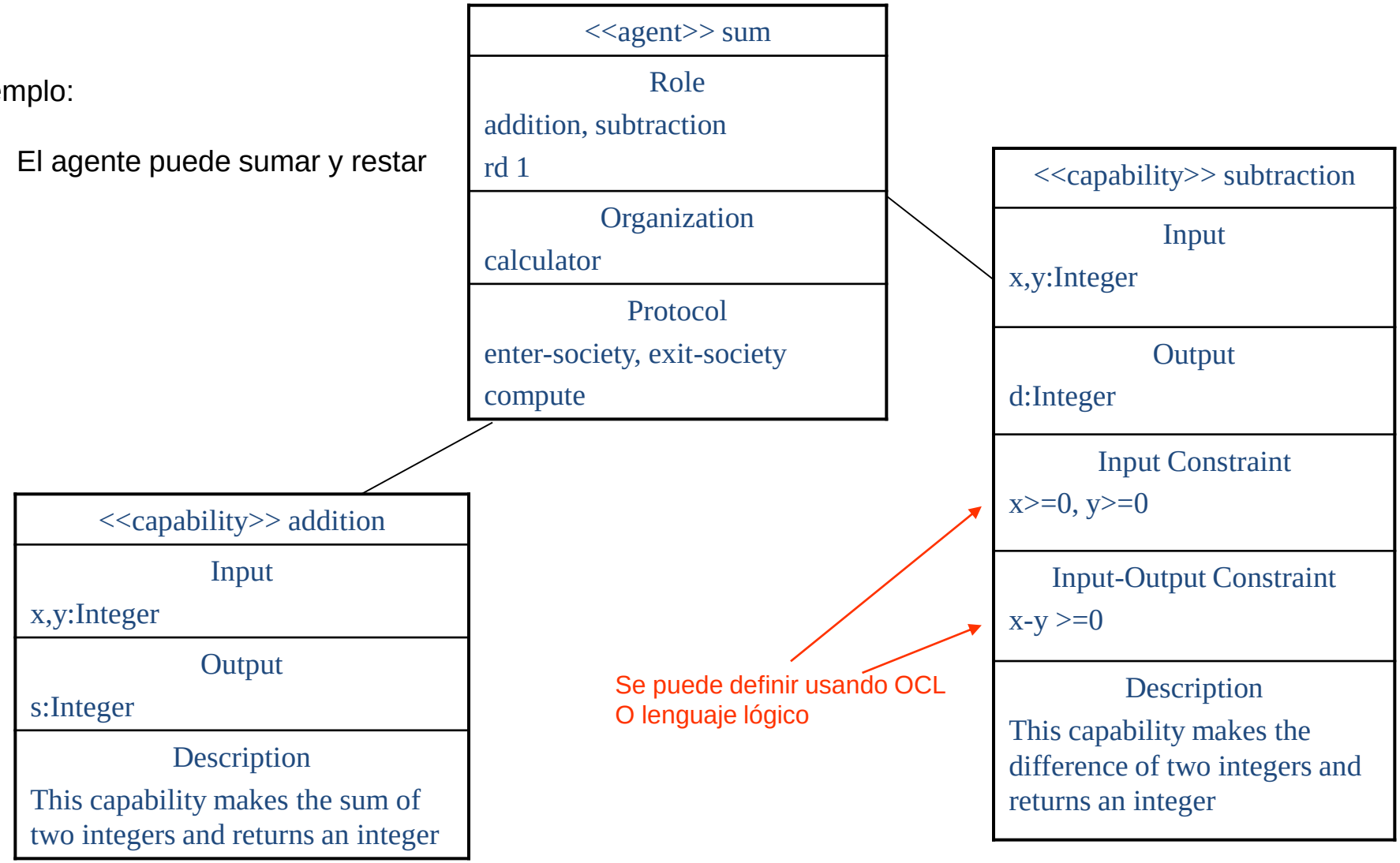
Capacidades

- Esta compuesta por:
  - Input
    - Entradas para realizar sus tareas
  - Output
    - Qué genera como resultados
  - Input constraints
    - Restricciones para poder realizar sus tareas
  - Output constraints
    - Restricciones para poder usar sus resultados
  - Input-output constraints
    - Restricciones que deben cumplirse en ambos casos
  - Description
    - Descripción en lenguaje natural de las capacidades

Capacidades

- Ejemplo:

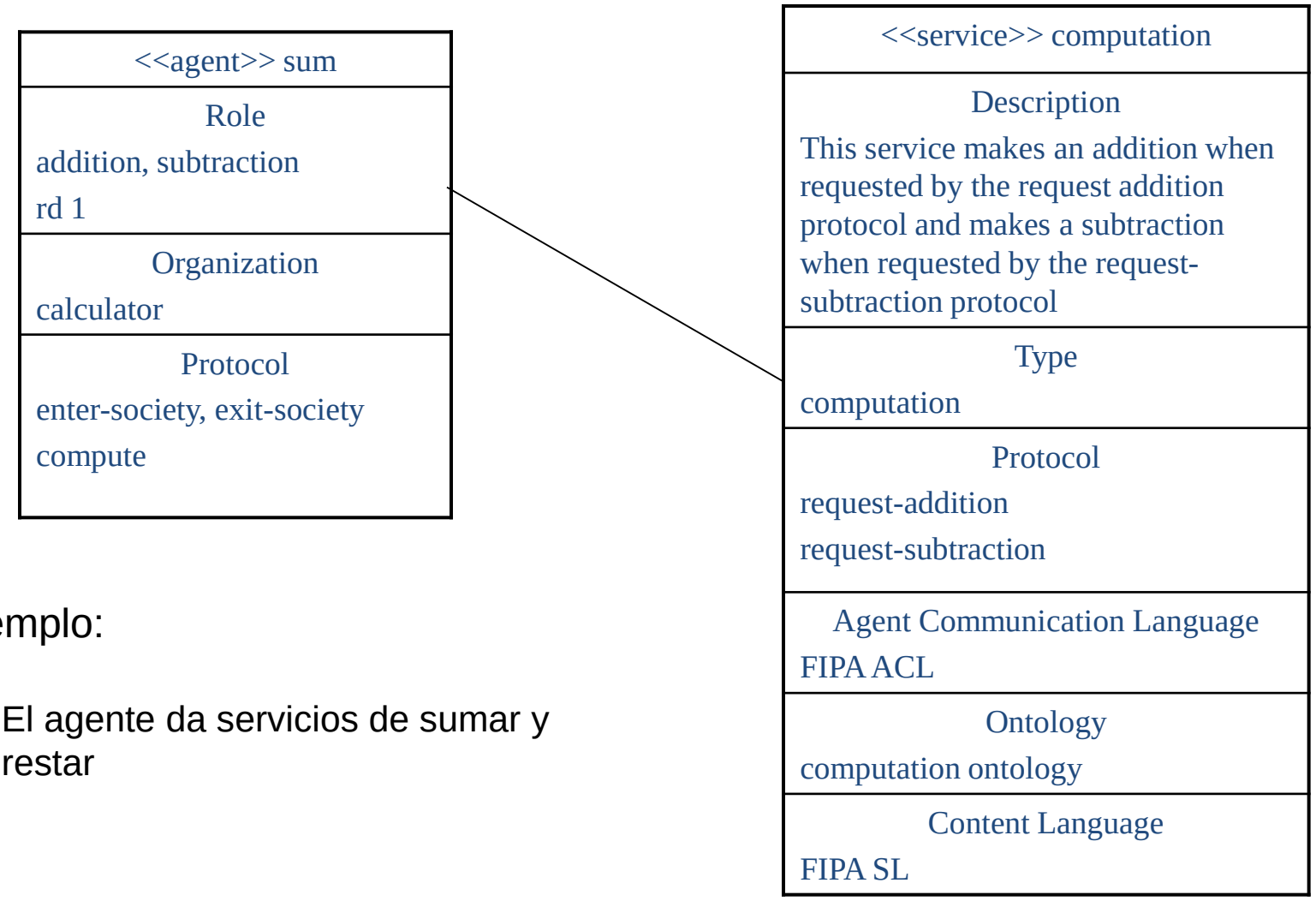
El agente puede sumar y restar



Servicio

- Esta compuesto por:
  - Name
    - Nombre del servicio
  - Description
    - La descripción
  - Type
    - El tipo
  - Protocol
    - Protocolos de interacción soportados por el servicio
  - Agent communication language
    - Lenguaje de comunicación usado
  - Ontology
    - Lista de ontologías usadas
  - Content language
    - Lenguaje de contenido
  - Properties
    - Propiedades del servicios

Servicio

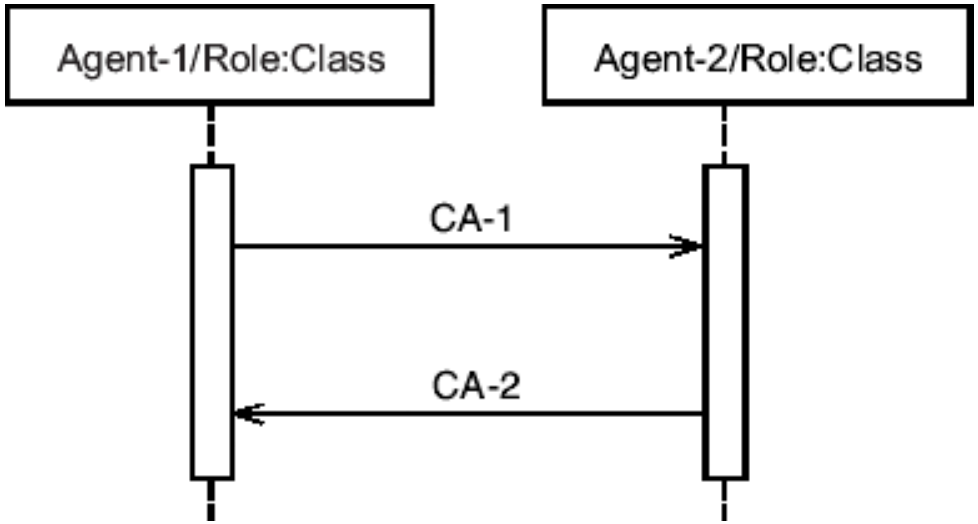


- Ejemplo:

  El agente da servicios de sumar y restar

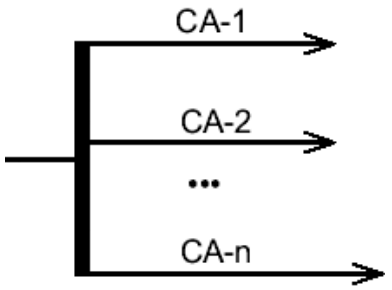
Interacciones

Interacciones entre agentes se define usando diagrama de secuencia de UML



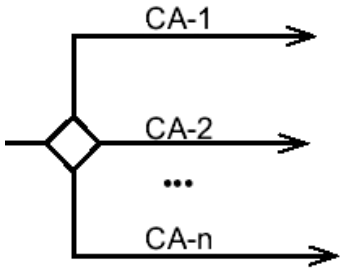
Interacciones Concurrentes

- A) Los actos de comunicación simultáneos procedentes de CA-1 a CA-n se envían en paralelo.



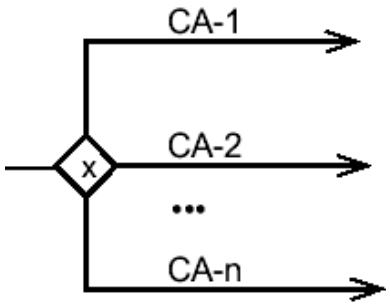
(a)

- b) Una selección de las n actos se envía en paralelo (cero o más).



(b)

- c) Elección exclusiva: sólo uno de los actos de comunicación se envía.

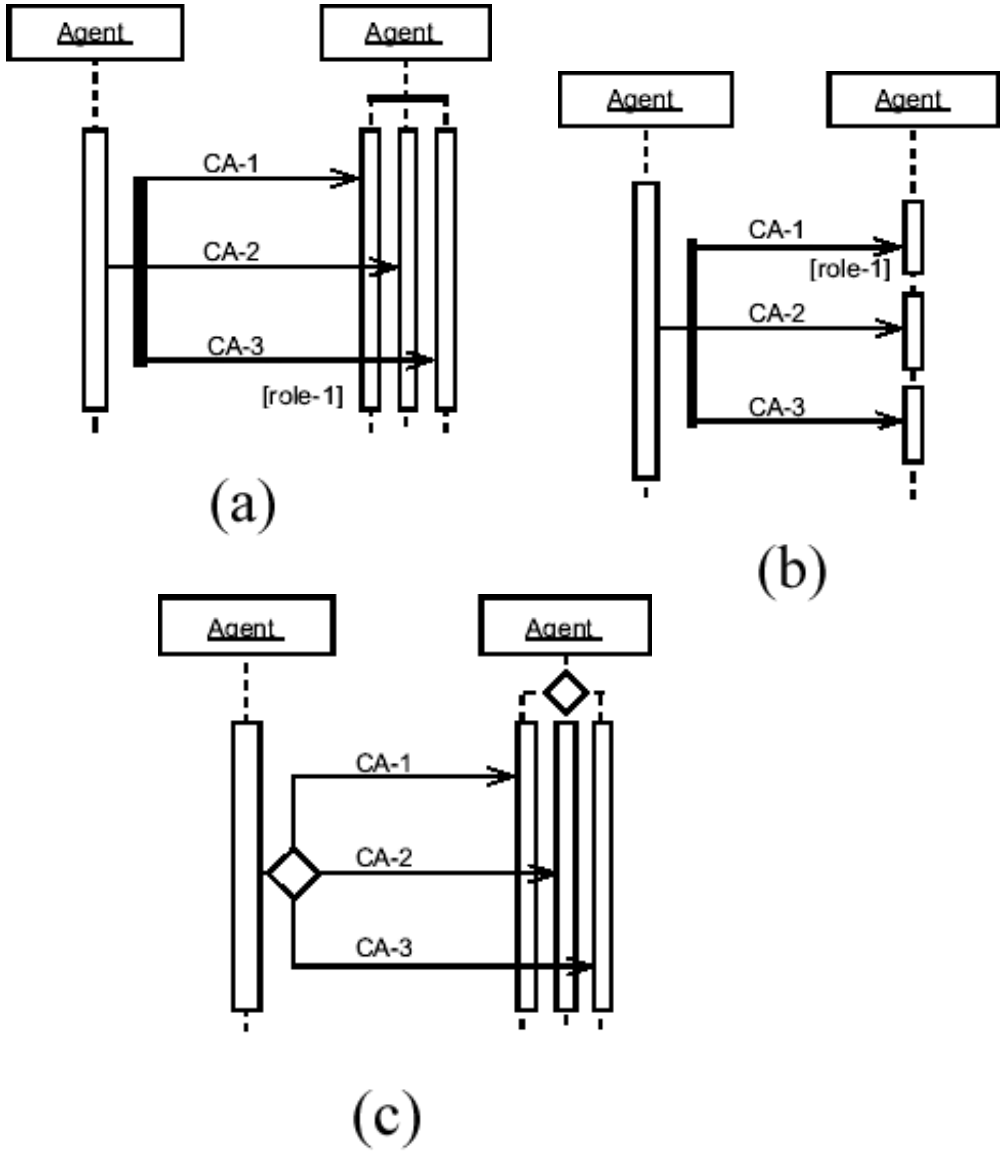


(c)

Interacciones Concurrentes

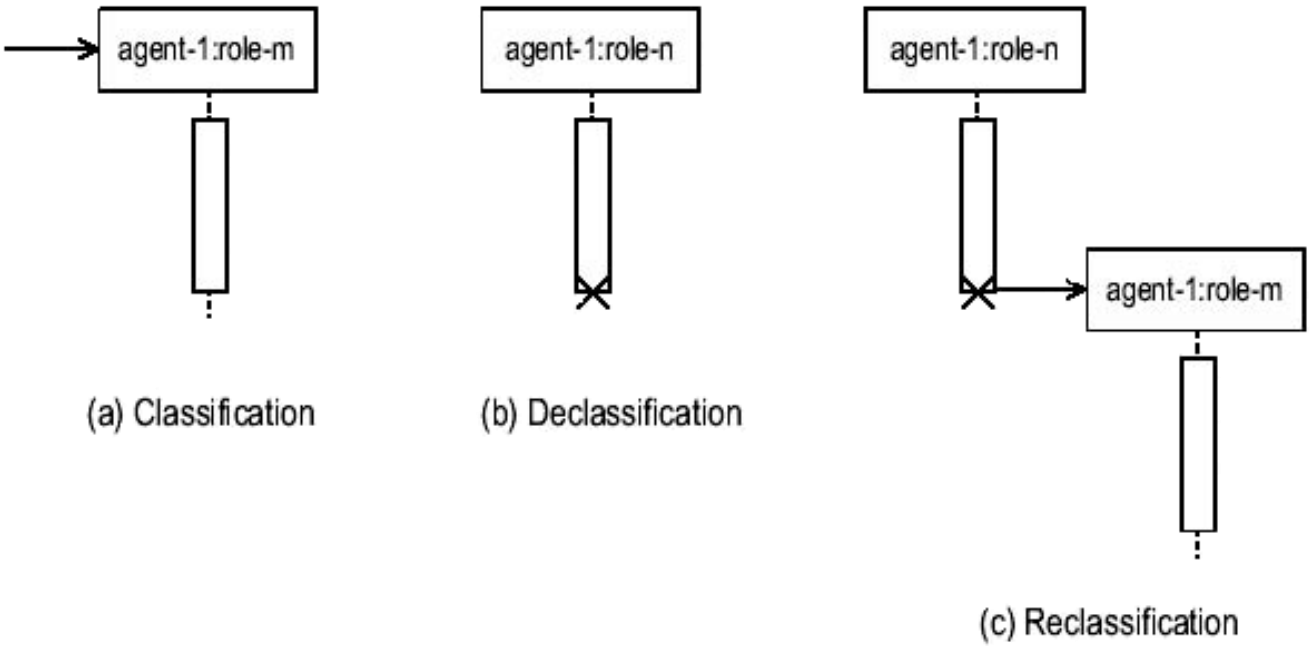
1. Un agente envía 3 CA concurrente a otro agente. El diagrama se puede interpretar de dos maneras diferentes:
    1. Cada CA es procesado desde el mismo agente/papel con un hilo diferente de la ejecución
    2. Cada CA es procesada por un papel diferente del agente (en este caso los mensajes pueden ser anotada especificando el papel)
2. La misma semántica de (a), pero con una notación más simple
3. Elección de tres actos de comunicación diferente recibido por tres hilos diferentes (o roles)

NOTA: cada CA concurrente puede ser enviado a diferentes agentes



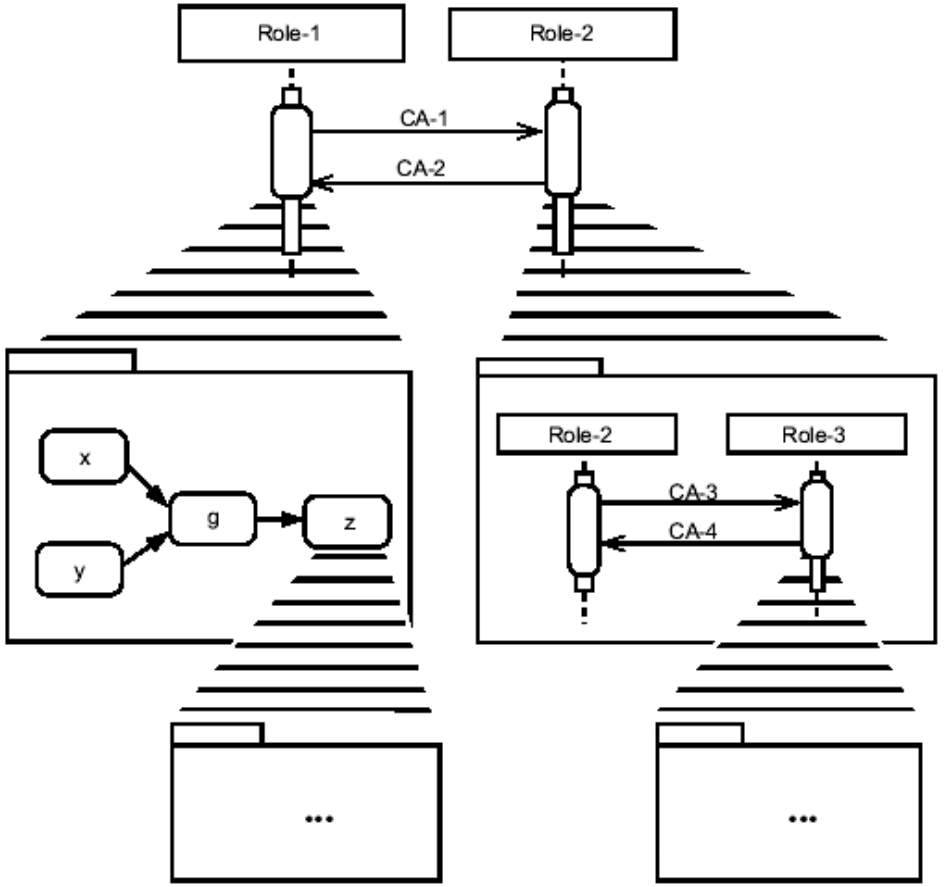
Roles

- diagrama de secuencia de UML para representar cambios en roles



Especificación

- Cualquier proceso de interacción se puede expresar en más detalle.
- La "nivelación" puede continuar hasta que el problema se ha especificado adecuadamente para generar código.
- También los diagramas de actividad y gráficos de estado pueden ser utilizados.



Diagramas de secuencia Extendidos

Expresan roles de los agentes

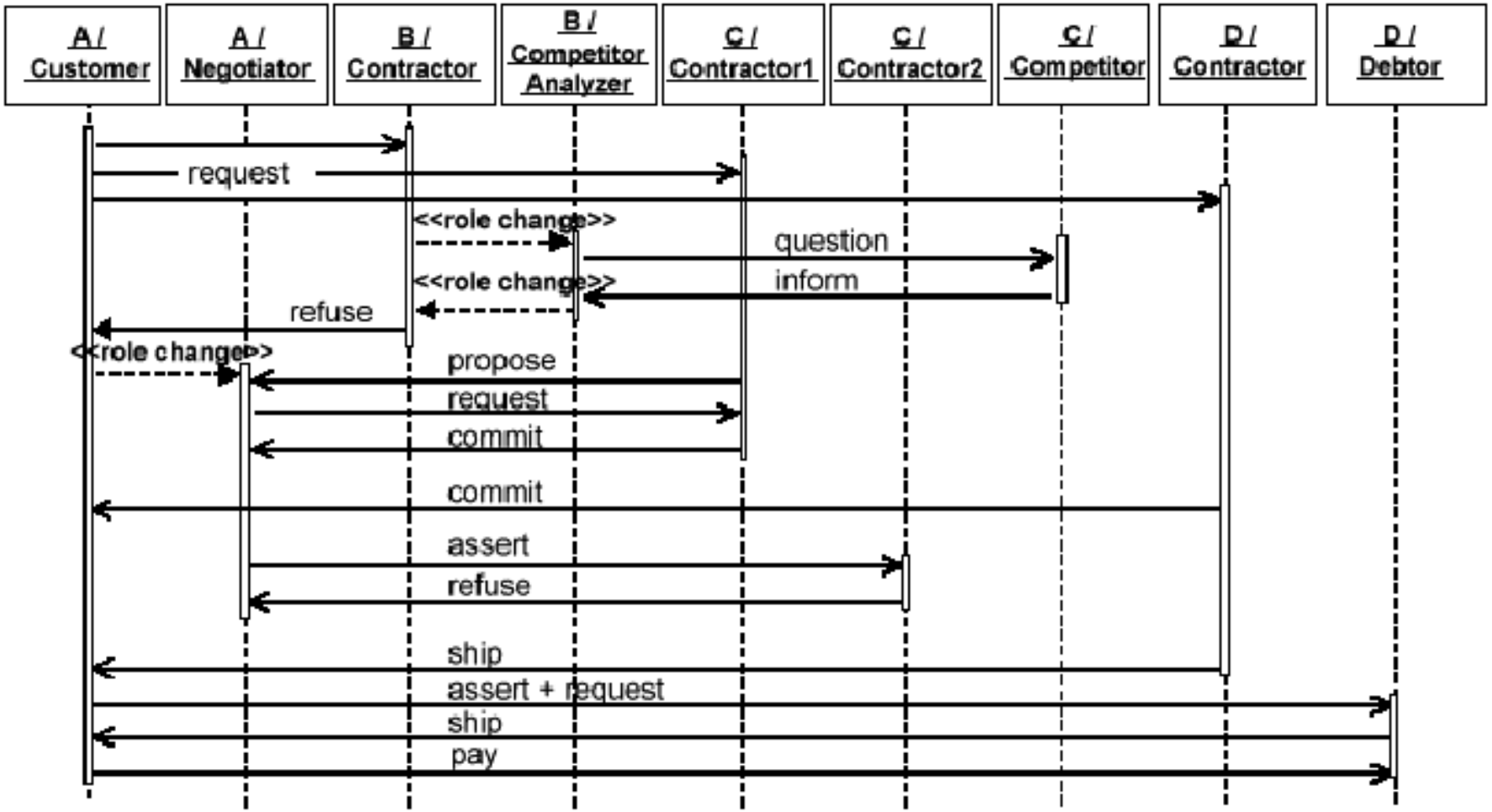


Diagrama de colaboración

otra forma de mostrar interacción entre agentes

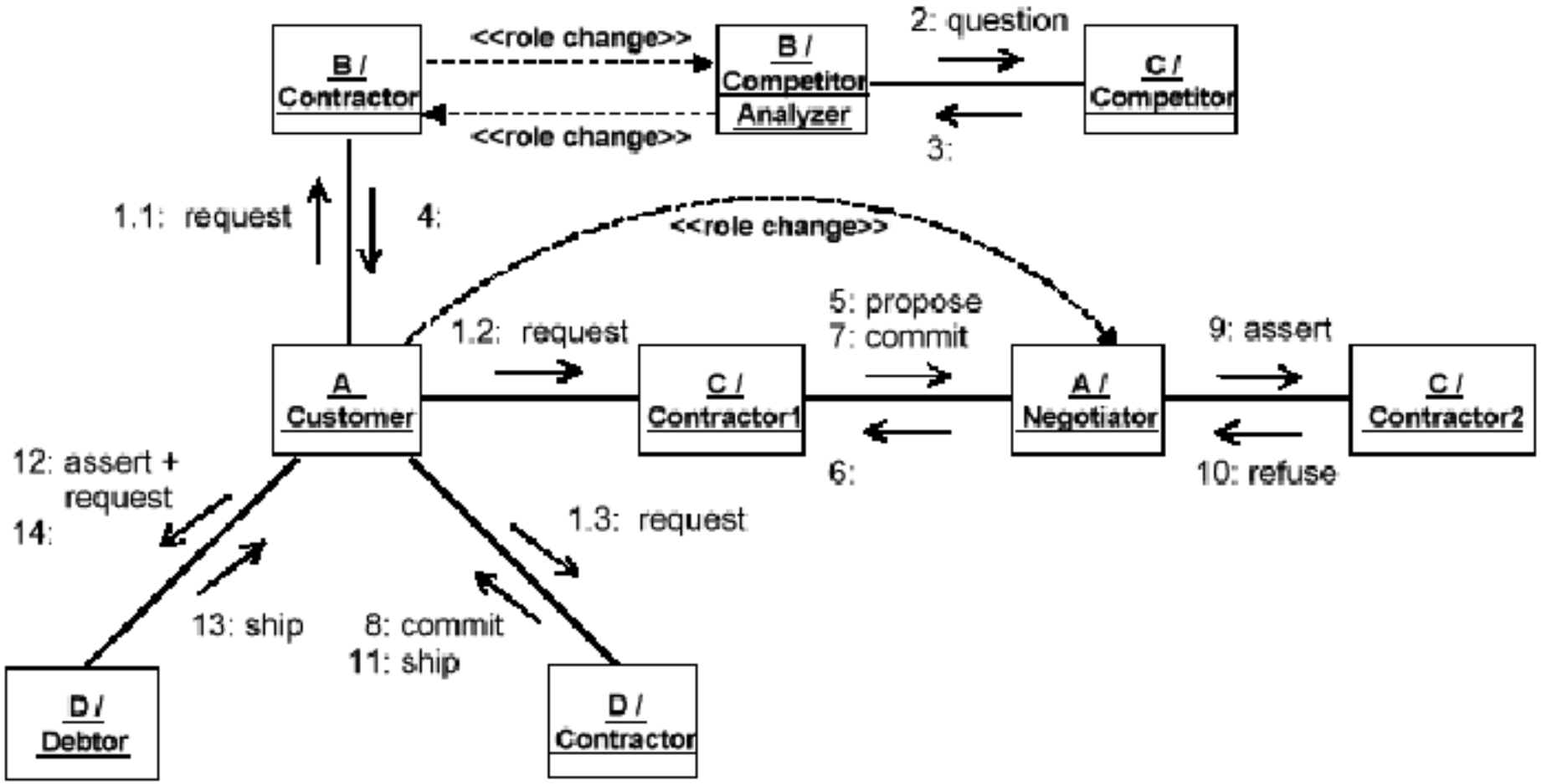
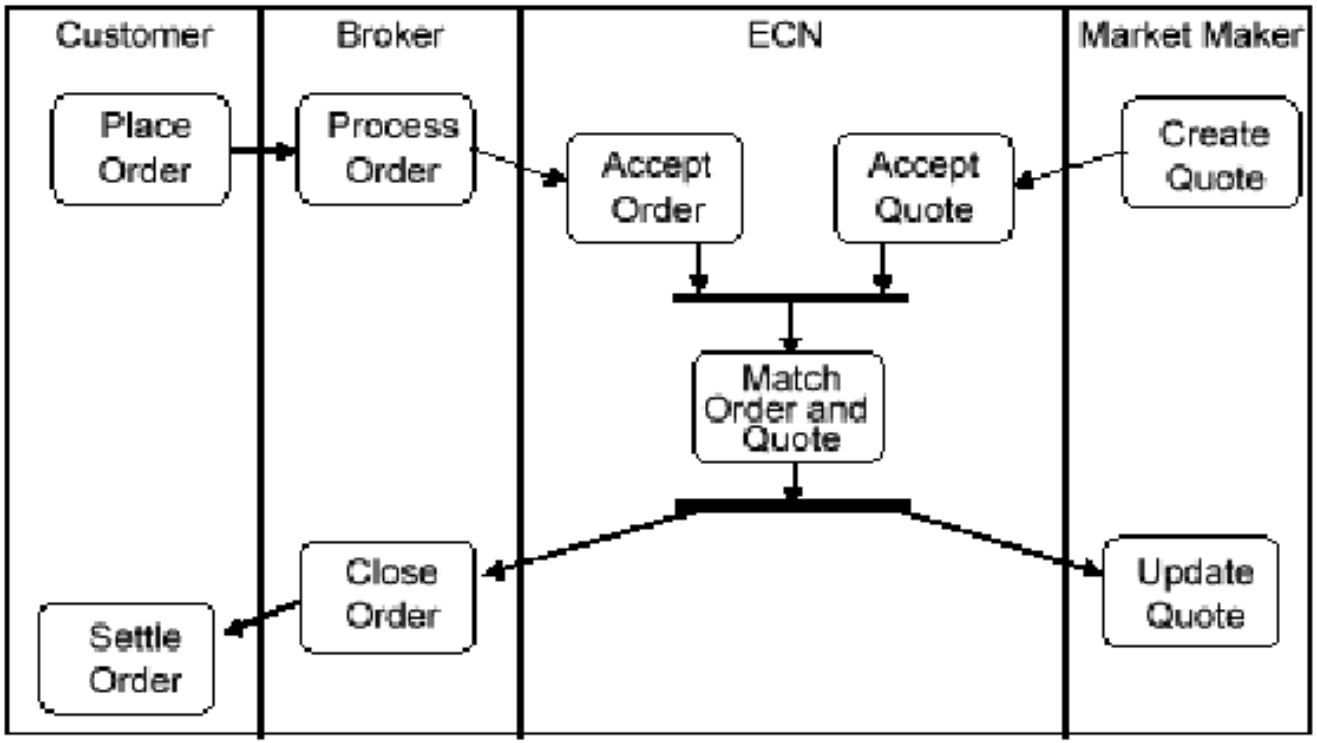


Diagram de Actividades Extendido

- Proporciona un hilo explícito de control?
- Útil para los protocolos de interacción complejos que involucran concurrencia



Multiagent System Engineering (MaSE)

- Diseñada para desarrollar multiagentes de propósito general

- **Análisis**
  - Definir metas a partir de los requerimientos
  - Definir roles necesarios para satisfacer las metas

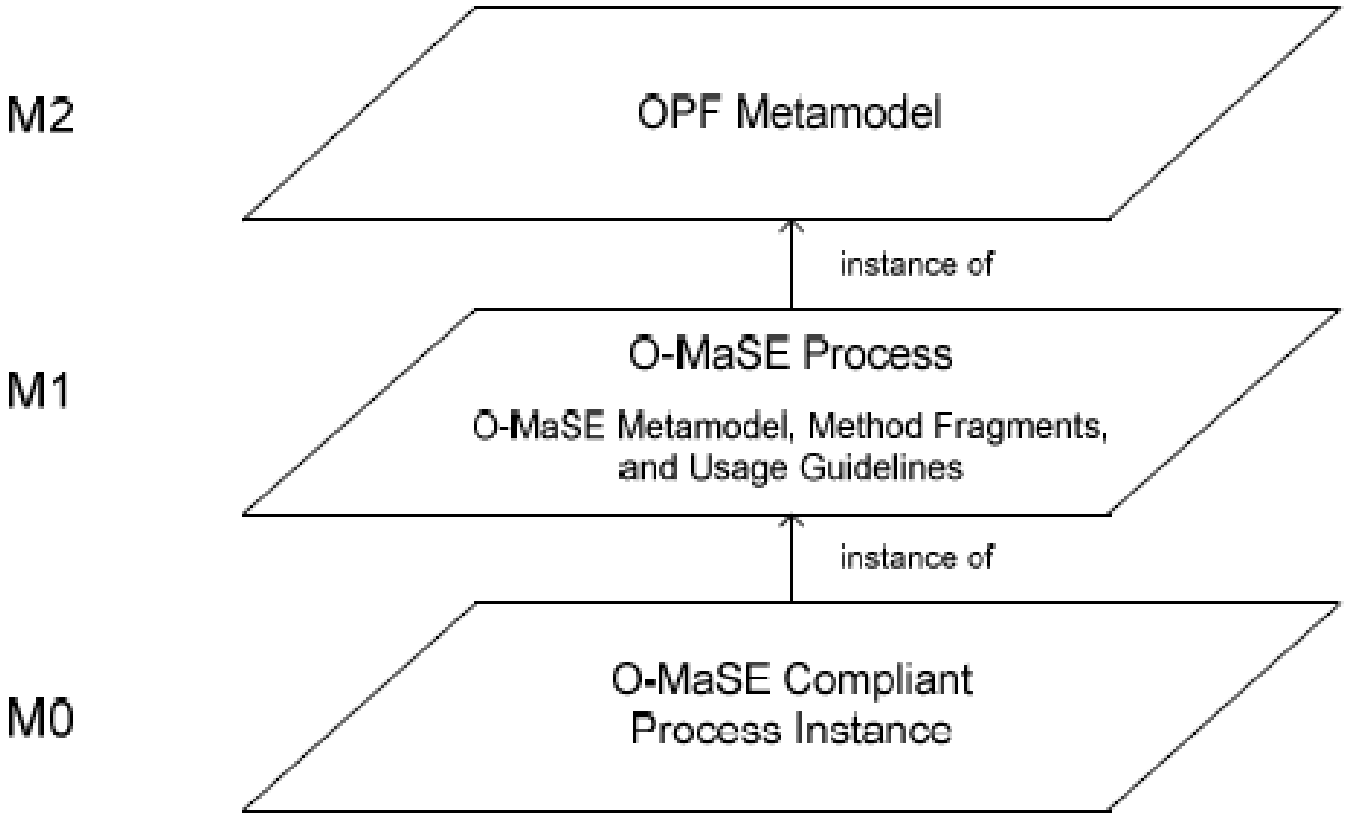
MaSE

- **Diseño**
  - Definir clases de agentes basado en los roles
    - Un agente puede ejecutar varios roles
    - Un rol puede ser dividido y ejecutado por varios agentes
  - Construir conversaciones
  - Ensamblar agentes
    - Definir la arquitectura
    - Definir los componentes de la arquitectura
  - Diseño del sistema

Problemas de MaSE

- No provee un mecanismo para modelar la interacción de los agentes con su ambiente
- MaSE produce un sistema multiagente con una organización definida. Estos sistemas deben ser capaces de diseñar y adaptar su organización dinámicamente
- MaSE no incluye el concepto de sub-equipos, todos los agentes pertenecen a una misma capa
- Las conversaciones entre agentes se diseñan a muy bajo nivel. Entender el proceso de comunicación entre agentes se vuelve complicado

O-MaSE Framework



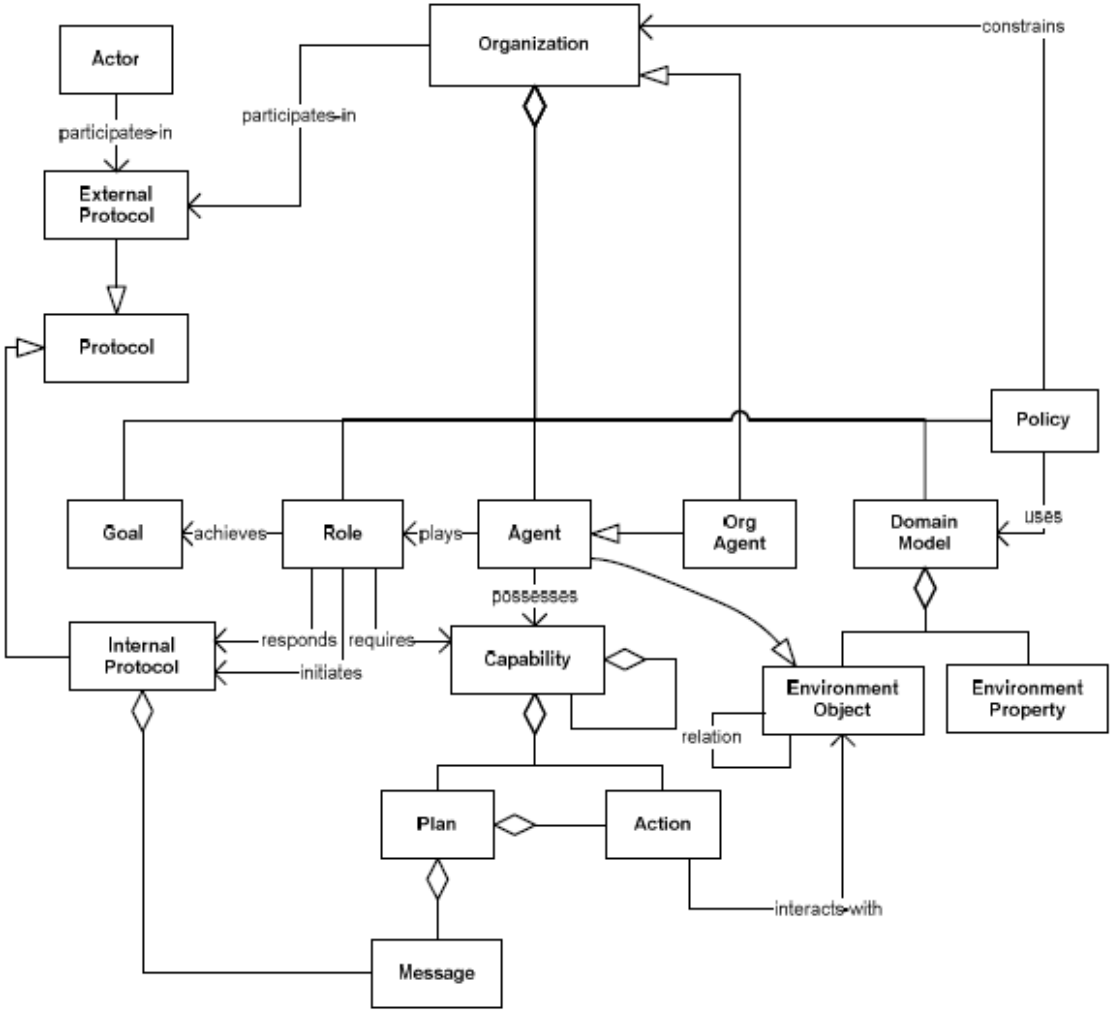
O-MaSE: Metamodelo

- Define los conceptos principales utilizados en los sistemas multiagente
- Basado en un enfoque organizacional
- Organización
  - Metas
  - Roles
  - Agentes
  - Modelo de dominio
  - Políticas

O-MaSE: Metamodelo (cont.)

- Meta: función u objetivo de la organización
- Rol: posición dentro de la organización que intenta alcanzar una meta
- Agente: percibe el ambiente y ejecuta acciones
  - Capacidades
    - Planes
    - Acciones
- Modelo del dominio: descripción del ambiente
- Políticas: reglas de la organización

O-MaSE: Metamodelo (cont.)



O-MaSE: Fragmentos de métodos

- Define actividades a realizar durante el proceso de desarrollo de software
- FIPA se encarga de desarrollar (agrupar) estos fragmentos
- Define Sistema Multiagente en términos de:
  - Escenarios
  - Unidades de trabajo
    - Actividades
    - Tareas
    - Técnicas
  - Productores
  - Productos
  - Lenguajes

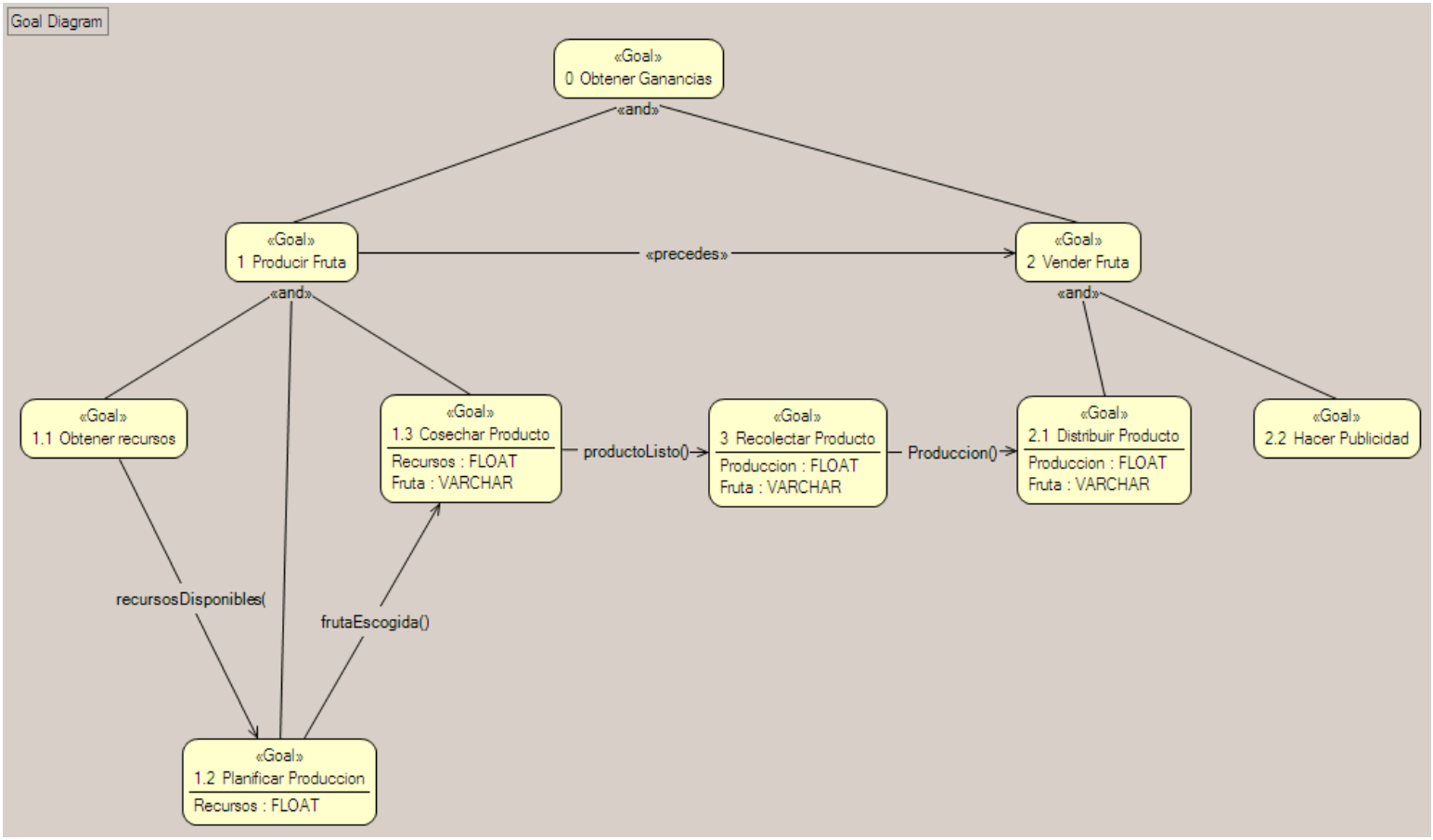
O-MaSE: Guías

- Combinar los fragmentos de métodos para obtener procesos
- Los procesos describen la metodología final que se va a utilizar
- Se especifican como (Entrada, Salida, Precondiciones, Postcondiciones)
  - **Entrada y salida**: un conjunto de productos
  - **Condiciones**: estado de productos y productores

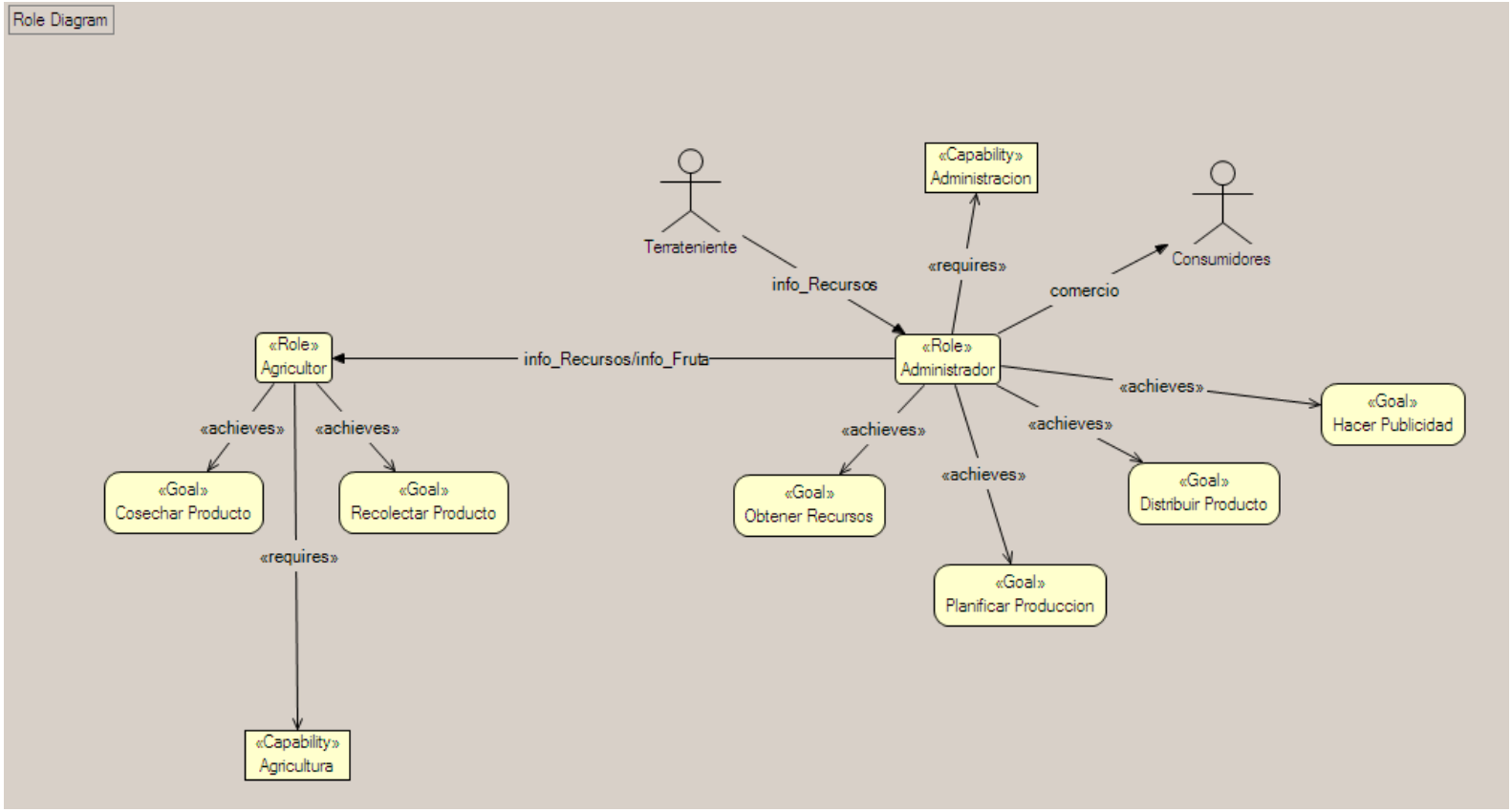
O-MaSE: tareas

- Modelar metas
- Refinar metas
- Modelar clases de agentes
- Modelar protocolo (interacción)
- Modelar plan

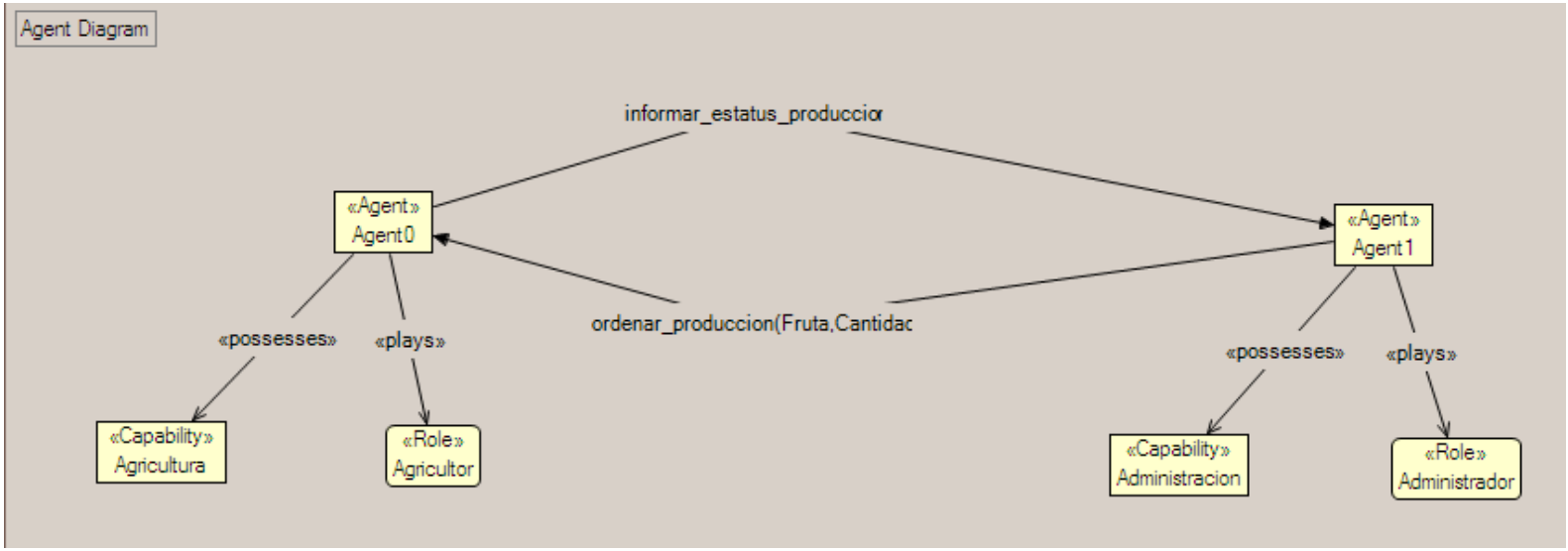
Modelo de Metas (con algo de refinamiento)



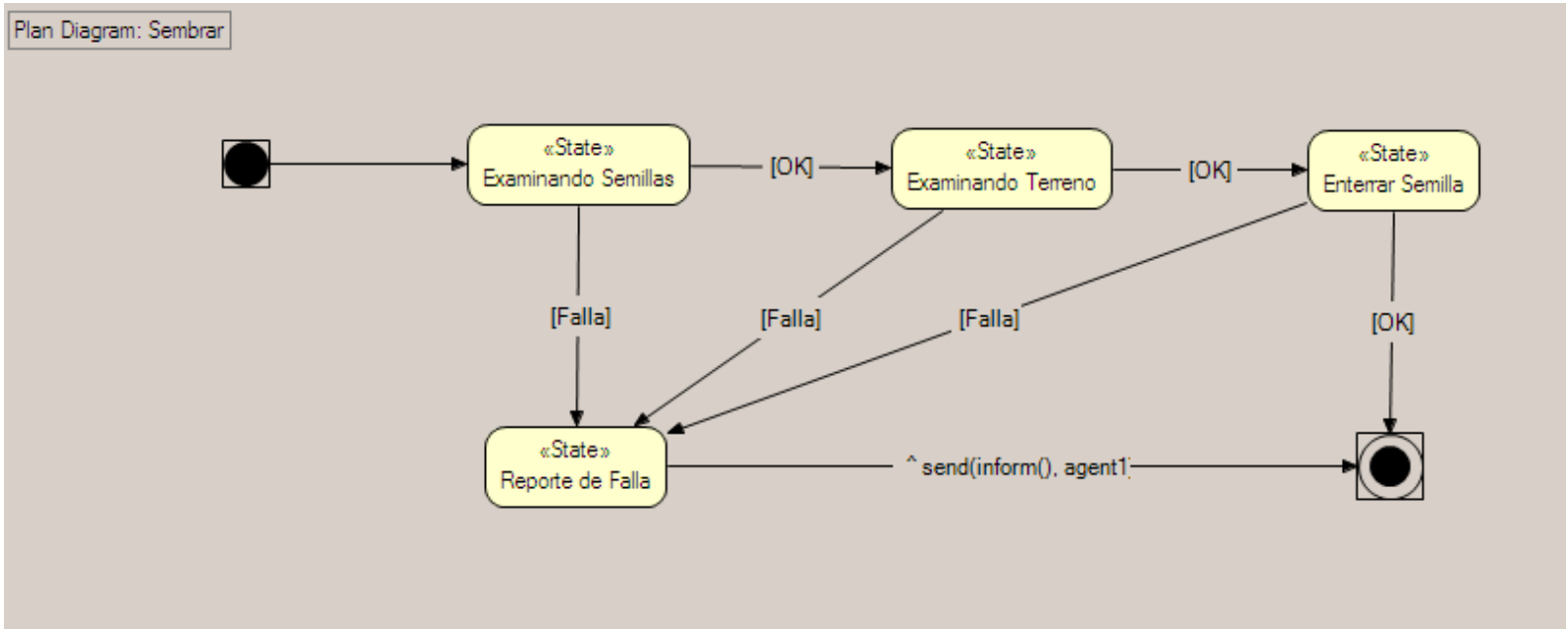
Modelo de Roles



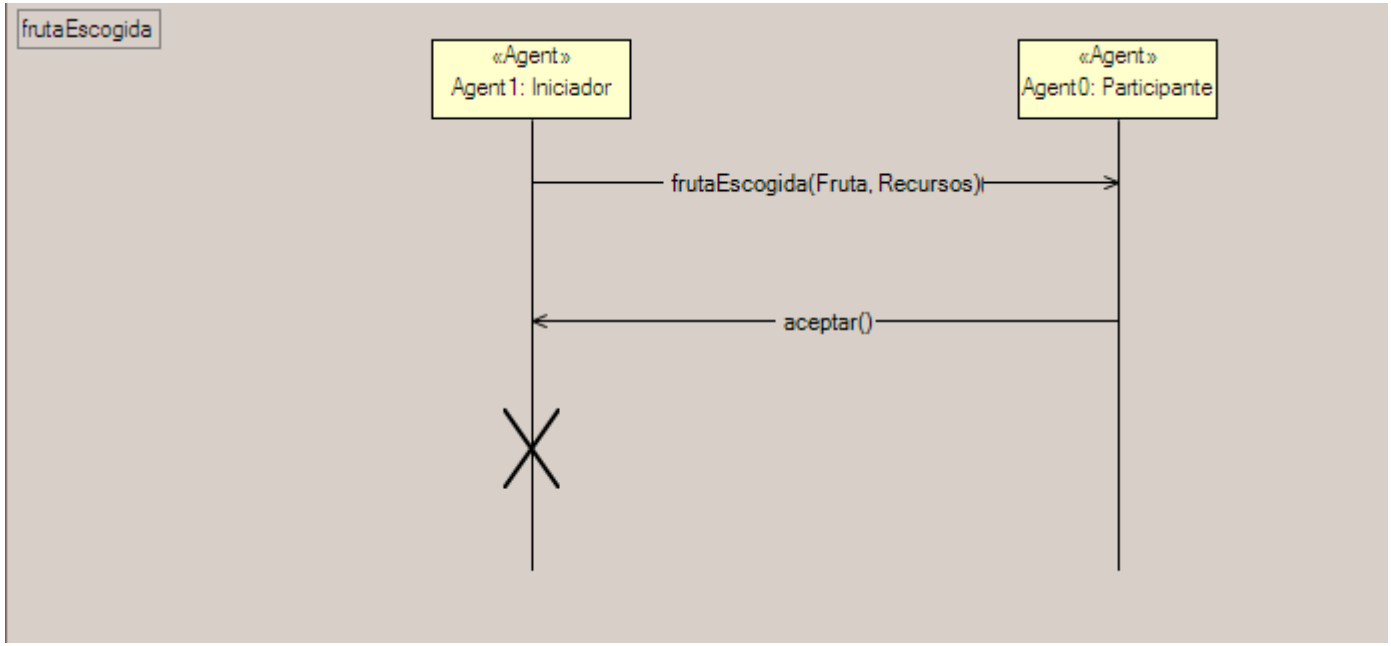
Modelo de Agentes



Modelo de Planes (Ej)



Modelo de Protocolos (Ej)



Comparación Metodologías

	ANÁLISIS	DISEÑO	UML	FASES
GAIA.	Modelo de Roles Modelo de Interacción	Modelo de Agente Modelo de Servicios Modelo de Conocimiento	-	AD
MASSIVE.	Vista de Tareas Vista de Entorno Vista de Sistema	Vista de Roles Vista de Interacción Vista de Sociedad Vista de Arquitectura Vista de Sistema	Si	ADI
AUML			Si	A

Comparación Metodologías

	ANÁLISIS	DISEÑO	UML	FASES
MaSE	Captura de Objetivos Aplicación casos de uso Transformación de Roles Creación clases de agente	Construcción de conversaciones Ensamblado clases de agente Diseño del sistema	Si	ADI
MASINA	Fase de Conceptuación Modelo de Organización Modelo de Agente Modelo de Tareas Modelo de Inteligencia M. Comunicación y Coord.	Diseño de la Red de agentes Diseño de Agentes Diseño de Plataforma	Si	ADI

Middleware (medio de gestión de servicios) o herramientas para el despliegue de los SMA

Middleware para SMA

Un cliente de un MGS para agentes aspira:

- **Mecanismos genéricos**: a nivel de protocolos de cooperación, de comunicación, de negociación, de coordinación, entre otros;
- **Capas de bajo nivel para los SMA**: primitivas de comunicación entre agentes (KQML, ACL, entre otros), mecanismos que permitan a los agentes entrar (registrarse) y salir (desregistrarse) del SMA, duplicarse (crear otras instancias), migrar (moverse a otro nodo), etc.
- **Motores que implementen el ciclo básico de comportamiento de los agentes**, incluyendo lenguajes de ontologías, sistemas de razonamiento, etc.
- **Acceso a los recursos disponibles**: mecanismos de comunicación de bajo nivel (sockets, RPC), mecanismos de procesamiento distribuido (hebras),
- etc.
- **Usar el concepto de agentes** como una abstracción.
- **Centrarse en los detalles del comportamiento de alto nivel del agente**.
- **Servicios de nombramiento, de transporte de mensajes y de paginas amarillas**.

Middleware para SMA

Algunos basados en FIPA son:

- JADE
- JIAC
- JACK
- Zeus
- Fipa-OS
- MGS

JADE

- JADE (*Java Agent DEvelopment Framework*) es un software para desarrollar aplicaciones basadas en agentes que sigue las especificaciones de FIPA,
- El objetivo de JADE es simplificar el desarrollo de agentes, garantizando seguir los estándares establecidos
- por la FIPA, a través de un conjunto de servicios ofrecidos por la plataforma.
  - Plataforma para la gestión de los agentes y
  - Un IDE (*Integrated Development Environment*) para el desarrollo de agentes.
  - Paquete basado en Java para desarrollar agentes.
  - Varias extensiones para ejecutarse en dispositivos con aplicaciones especificas (PDA, teléfonos inteligentes, etc.).

JADE

- La plataforma de agentes se puede distribuir en varios *hosts*. Es suficiente una Maquina Virtual Java (JVM) en cada host.

- La plataforma JADE soporta la coordinación de múltiples agentes FIPA y proporciona una implementación estándar del lenguaje de comunicación FIPA-ACL,

- Para la implantación de SMA, JADE proporciona:
  - Un entorno de ejecución en el que los agentes se ejecutan.
  - Bibliotecas de clases para la creación de agentes mediante la herencia y la redefinición de comportamientos.
  - Un conjunto de herramientas gráficas para el monitoreo y la administración de la plataforma.

JADE

JADE tiene un contenedor principal que tiene dos agentes especiales:

- *DF (Directory Facilitator):*
- *AMS (Agent Management System): register() takedown()*, y
- *deregister()* del AMS.

JADE define la Clase *Agent*, la cual es una superclase que permite a los usuarios crear agentes

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

- Pasar mensajes utilizando objetos ACLMessage
- Dar soporte al ciclo de vida de un agente.
- Planificar y ejecutar m´ múltiples actividades al mismo tiempo.

JADE

JADE tiene un contenedor principal que tiene dos agentes especiales:

- *DF (Directory Facilitator):*
- *AMS (Agent Management System): register() takedown()*, y
- *deregister()* del AMS.

JADE define la Clase *Agent*, la cual es una superclase que permite a los usuarios crear agentes

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

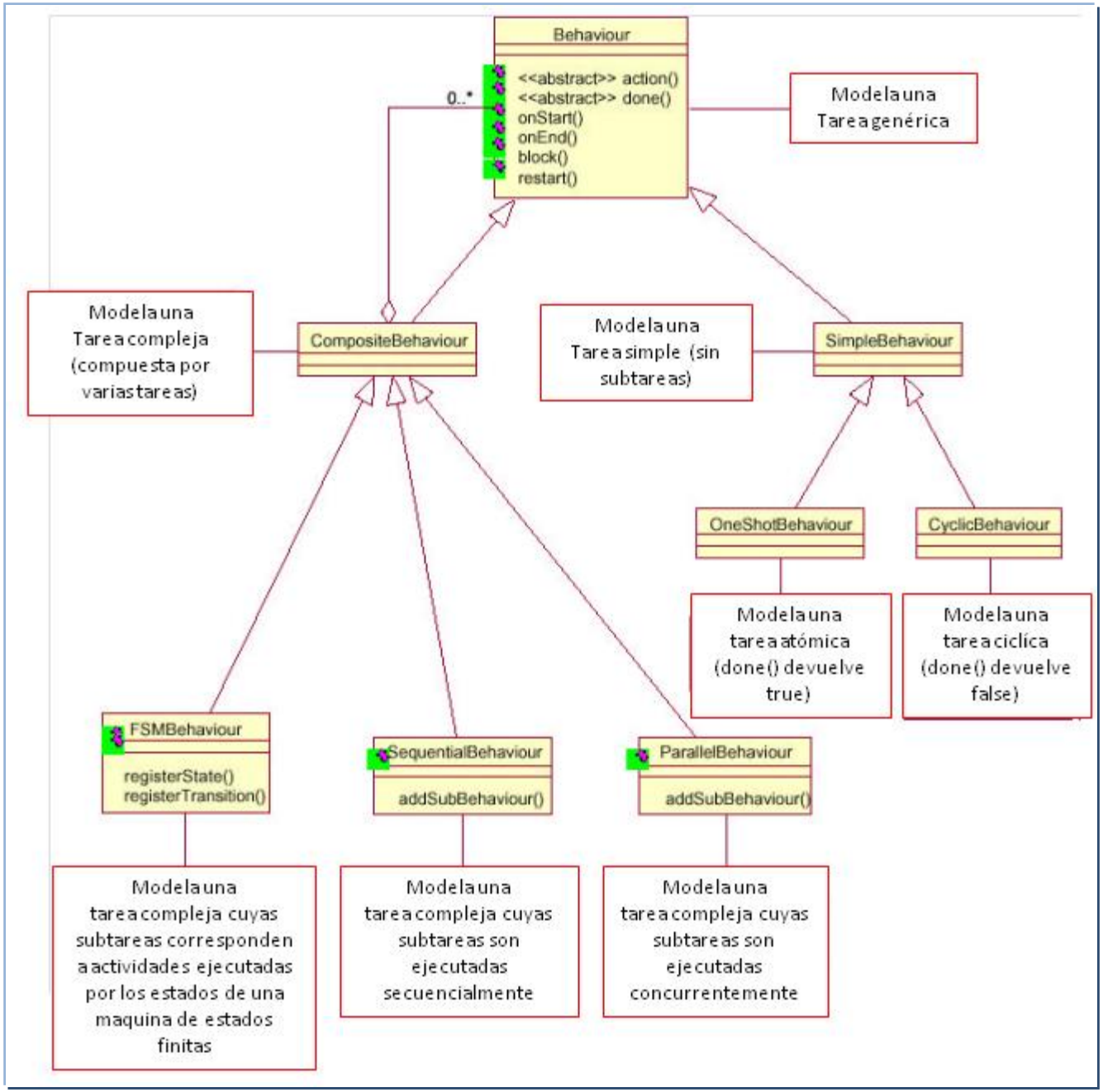
- Pasar mensajes utilizando objetos ACLMessage
- Dar soporte al ciclo de vida de un agente.
- Planificar y ejecutar m´ múltiples actividades al mismo tiempo.

JADE

- El “comportamiento” de un agente define las acciones a realizar bajo un determinado evento y hereda de la clase agentes los métodos: *addBehaviour, removeBehaviour*.
- Los diferentes comportamientos que el agente adopta se definen a partir de la clase abstracta *Behaviour*, para lo cual usa el método *action(), done(), reset(),*
- JADE incluye prototipos de comportamientos listos para utilizarse, tales como los protocolos de interacción FIPA, despertar bajo una cierta condición, etc.

- los agentes se implementan como un hilo por agente.
- Además de la solución multi-hilo, ofrecidos directamente por Java, JADE también admite la programación de comportamientos cooperativos

JADE


Behaviour
<<abstract>> action()
<<abstract>> done()
onStart()
onEnd()
block()
restart()
0..*
Modela una Tarea genérica
Modela una Tarea compleja (compuesta por varias tareas)
CompositeBehaviour
Modela una Tarea simple (sin subtareas)
SimpleBehaviour
OneShotBehaviour
CyclicBehaviour
Modela una tarea atómica (done() devuelve true)
Modela una tarea cíclica (done() devuelve false)
FSMBehaviour
registerState()
registerTransition()
SequentialBehaviour
addSubBehaviour()
ParallelBehaviour
addSubBehaviour()
Modela una tarea compleja cuyas subtareas corresponden a actividades ejecutadas por los estados de una maquina de estados finitas
Modela una tarea compleja cuyas subtareas son ejecutadas secuencialmente
Modela una tarea compleja cuyas subtareas son ejecutadas concurrentemente

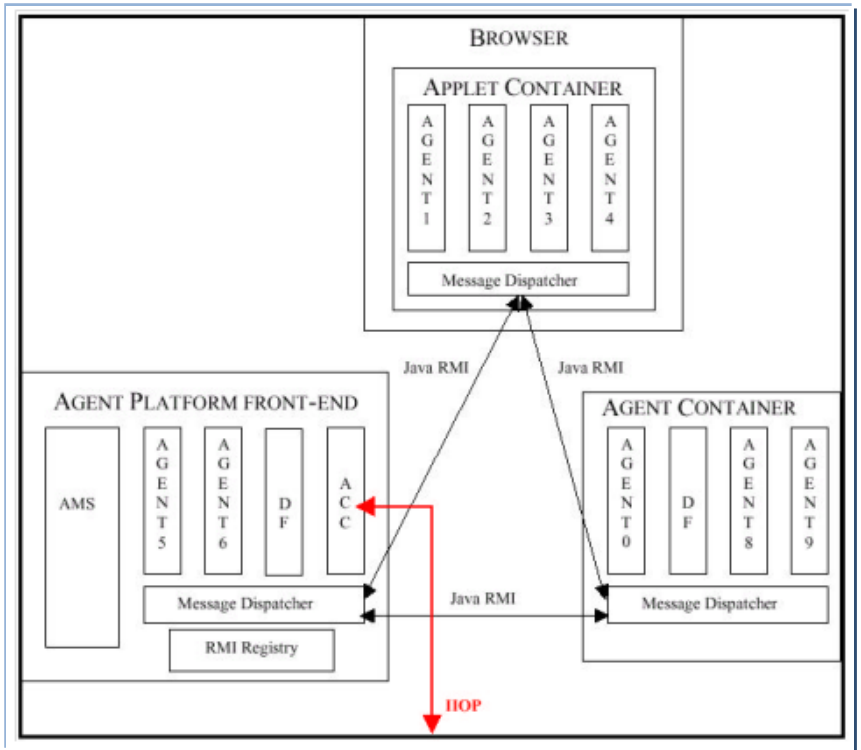
JADE

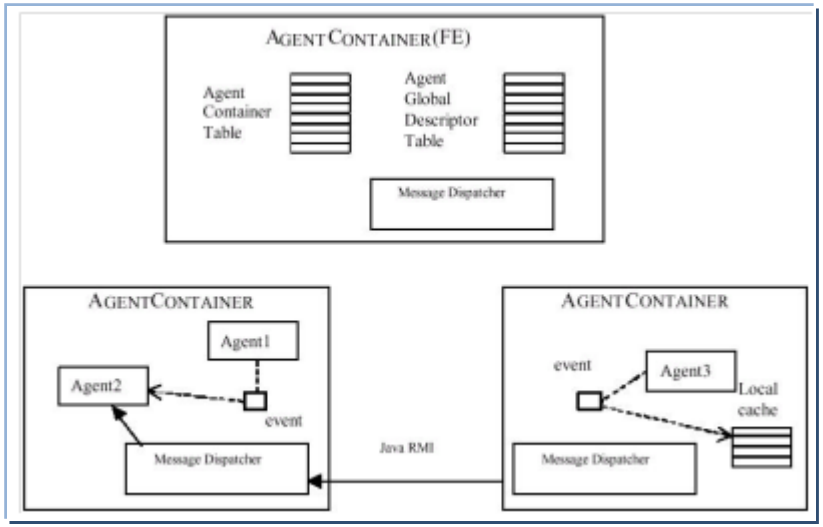
- En el caso de las ontologías, un sistema de gestión de ontología de agentes ha sido desarrollado, así como el soporte a lenguajes y ontologías que pueden ser implementadas y registradas con los agentes.
- JADE ofrece una llamada a *JessBehaviour* que permite la integración con JESS, un entorno para la programación de reglas con su motor de razonamiento

- Cada agente vive en un contenedor, que es una maquina virtual de Java, el cual provee un completo ambiente de ejecución para los agentes, permitiendo que varios de ellos puedan ejecutarse concurrentemente, controlando sus ciclos de vida (crearlos, suspenderlos, eliminarlos, etc.) y comunicaciones (envió y enrutamiento de mensajes).

JADE

- Existe un contenedor especial, ***llamado front-end (FE),*** que ejecuta los agentes de administración
- Existe otro contenedor especial para mostrar la interfaz Web. Este contenedor ejecuta un agente llamado ***Remote Management Agent (RMA)***

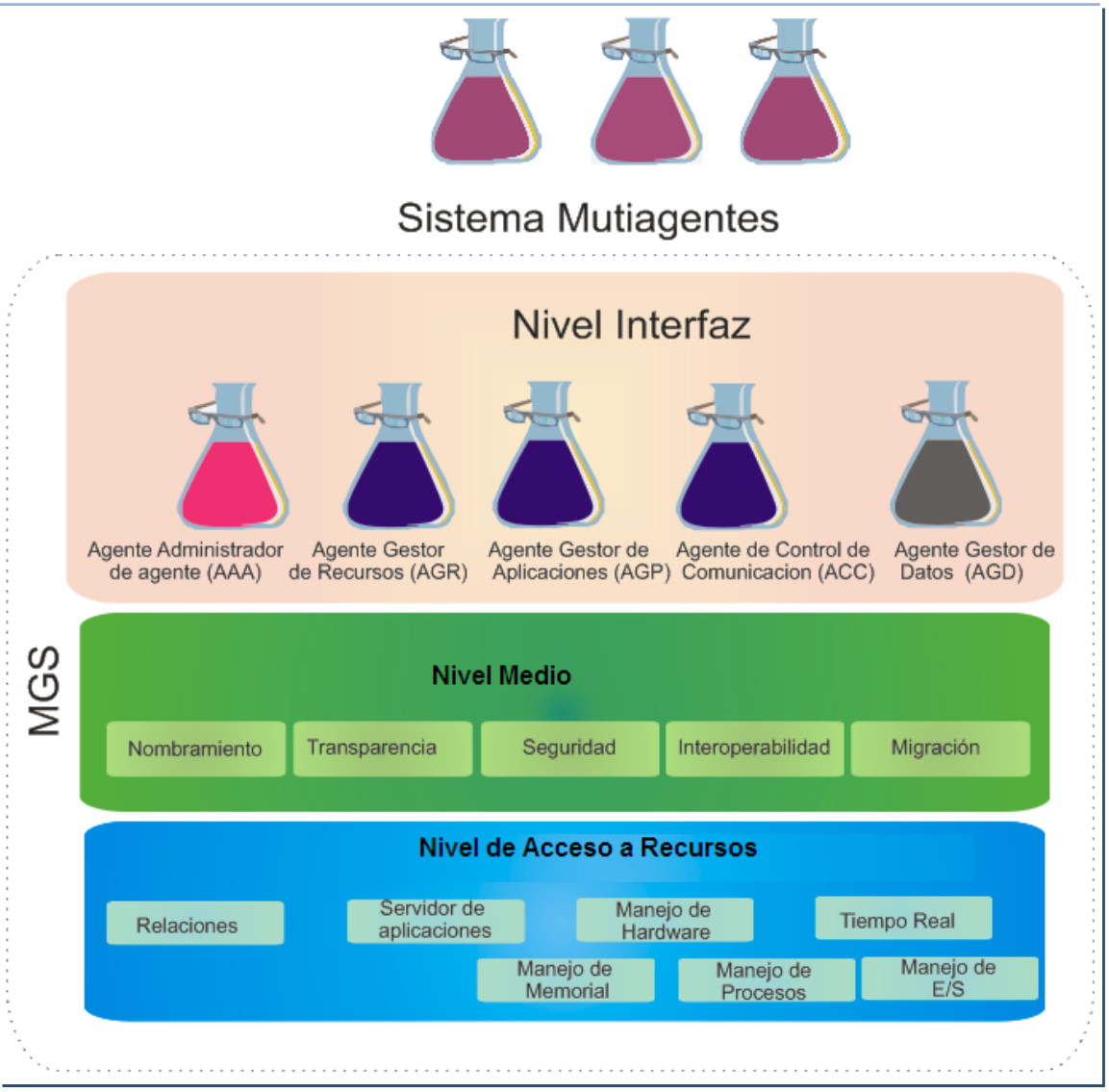




JADE

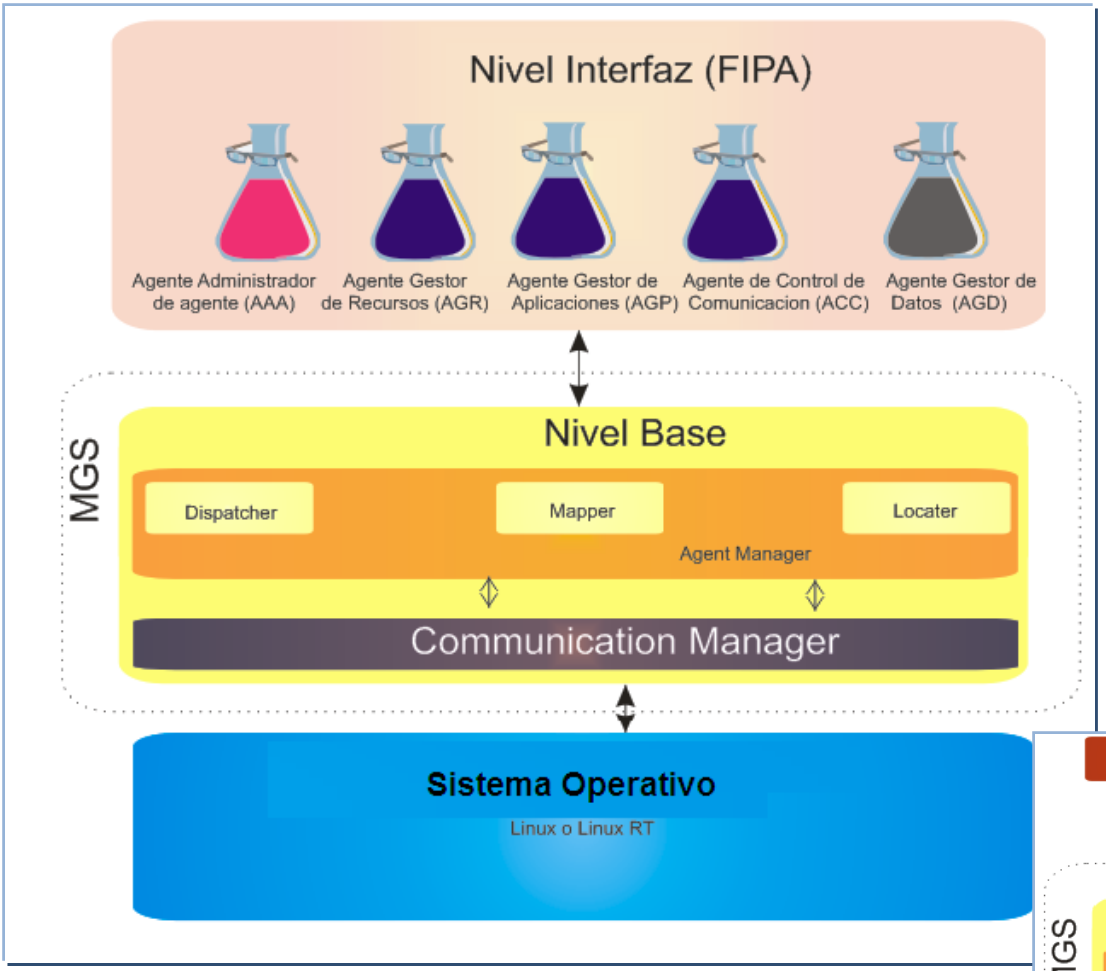
- **El agente *dummy*** es una herramienta para la inspección de los intercambios de mensajes entre los agentes. Este agente facilita la prueba y validación de la interfaz de los agentes antes de su integración en un SMA.
- **El agente *sniffer***, a través de su IGU, permite el seguimiento de los mensajes intercambiados en una plataforma de agentes JADE. Cuando el usuario decide rastrear un agente o un grupo de agentes, cada mensaje dirigido a, o procedente de, el agente o grupo de agentes, se sigue y se muestra en la ventana del *sniffer*, utilizando una notación similar a los diagramas de secuencia de UML.
- **El agente *introspector*** permite monitorizar y controlar el ciclo de vida de un agente en ejecución, y sus mensajes intercambiados, en particular, su cola de mensajes enviados y recibidos.

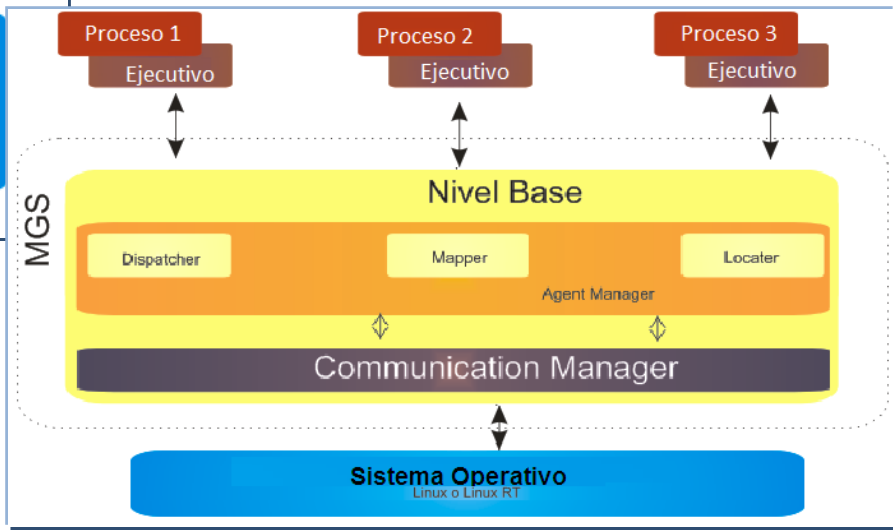
MGS

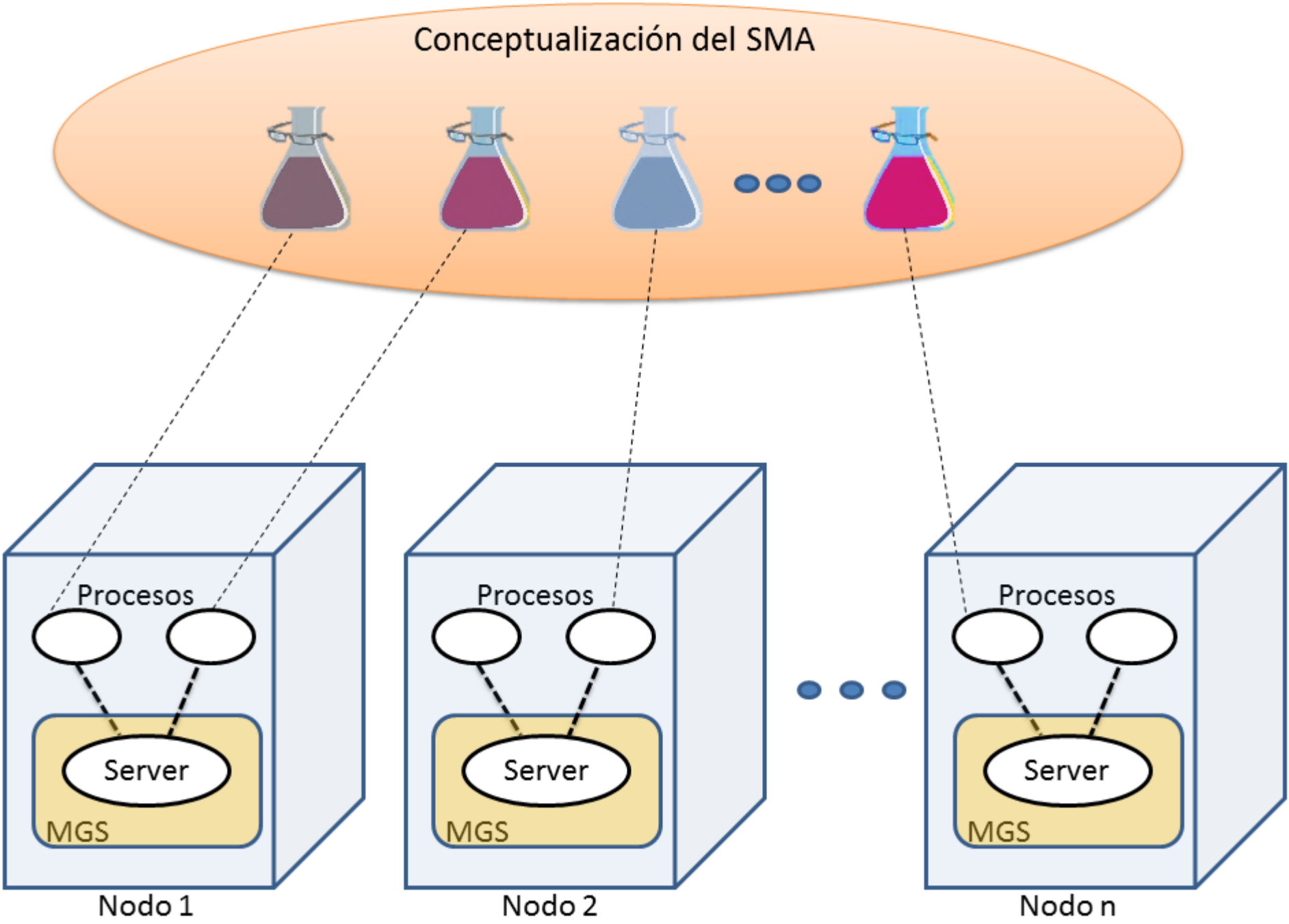


- Proveer mecanismos que permitan a una (comunidad de agentes:
- Conocer los servicios disponibles agrupados por niveles, por tipos de servicio, o alguna otra clasificación
- Conocer los agentes que prestan un determinado servicio y donde están
- Conocer las formas para acceder a esos servicios. entre otros.
- La movilidad,
- Comunicarse los agentes
- Su activación y desactivación, la posibilidad de invocarlos, etc.

Middleware para SMA






Conceptualización del SMA
Procesos
Server
MGS
Nodo 1
Procesos
Server
MGS
Nodo 2
Procesos
Server
MGS
Nodo n

Comparación Plataformas Despliegue

- Qué lenguaje de comunicación de agentes (ACL) soportan: FIPA ACL y/o KQML
- Si soportan movilidad de código,
- Arquitectura base de la plataforma,
- De qué tipo son los agentes soportados,
- Cuáles son los lenguajes en los que se pueden desarrollar los agentes,
- Cuáles son las licencias de los paquetes de software,
- Si están disponibles en Internet,
- Cuán dificultosa es la instalación de dichos paquetes,
- Qué tan completa es la documentación
- Qué tipo de interface posee.
- Si tienen un EDI

Herramientas para el desarrollo o IDE para agentes

IDE para SMA

- un IDE es un sistema informático compuesto por un conjunto de herramientas de programación.

- Un IDE puede dedicarse, en exclusiva, a un solo lenguaje de programación o bien puede utilizarse para varios.

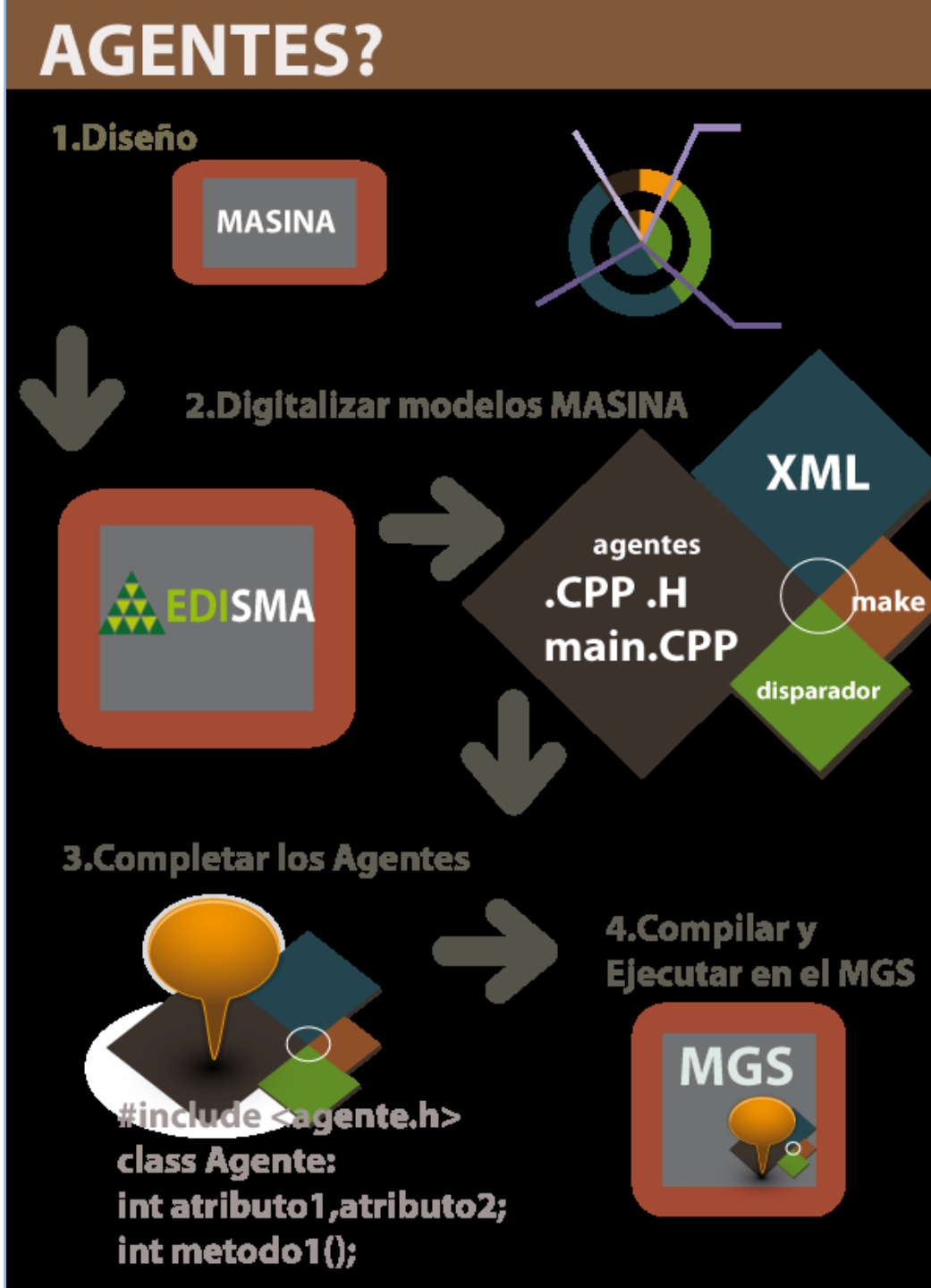
- Por lo general, un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación; es decir, consiste en un editor de código, un compilador, un depurador y un constructor de IGU.

EDISMA

Entorno de Desarrollo Integrado para la construcción de SMA.

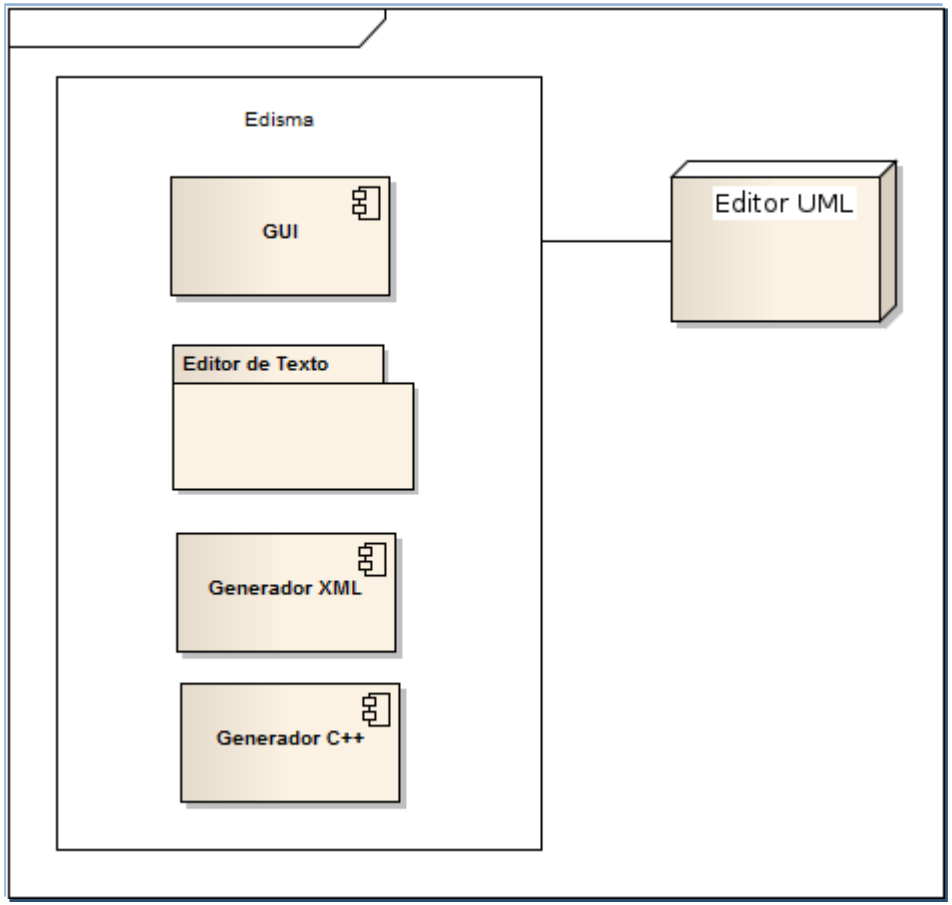
EDISMA pretende facilitar la creación de agentes modelados a través de MASINA.

Pero además, EDISMA despliega los agentes sobre MGS



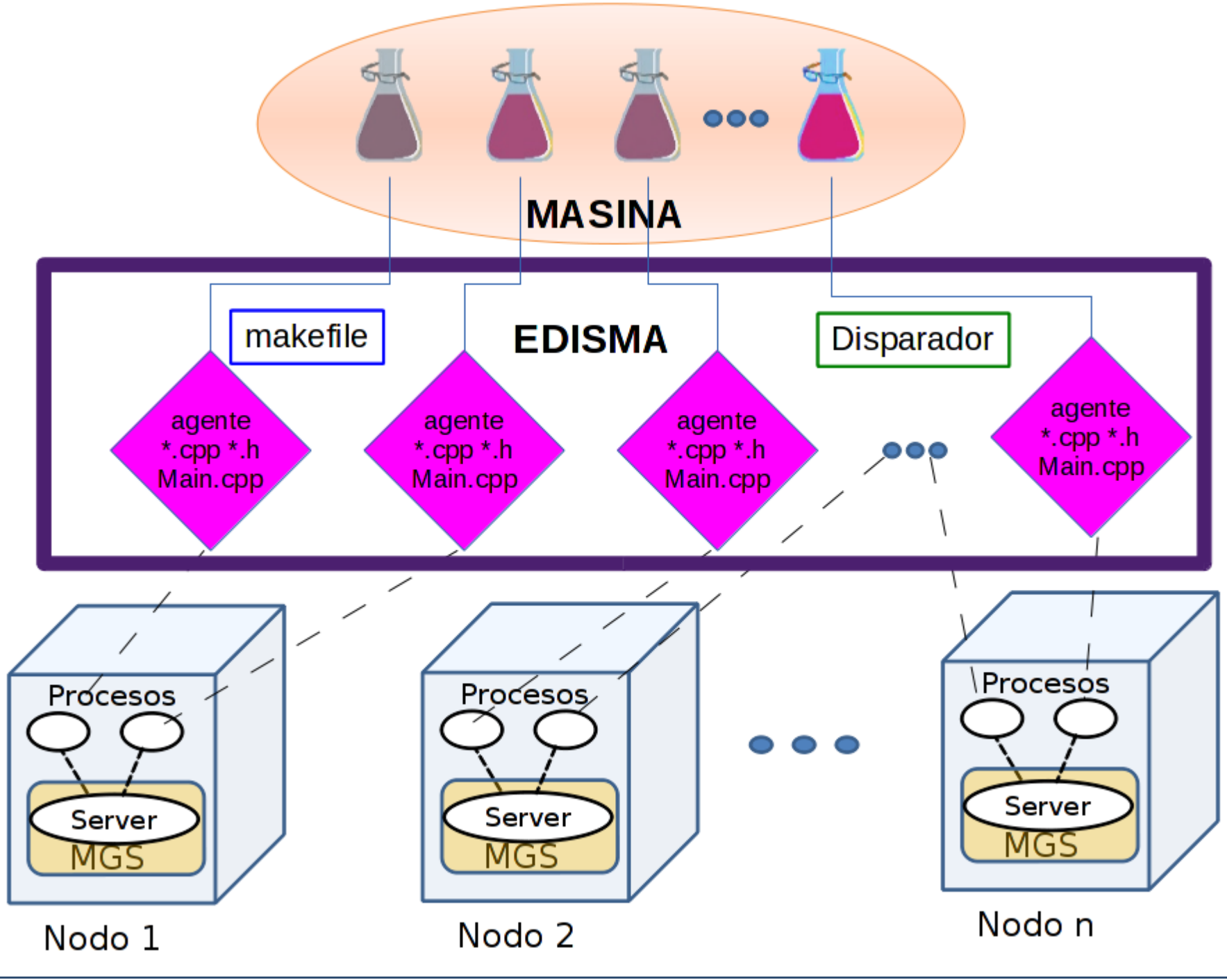
EDISMA.

- Proveer una IGU para modelar los agentes, seg´un MASINA.
- Implementar un componente para especificar los modelos. Almacena los modelos en un formato digital estándar, para ser usado tanto en la documentación del SMA, como en la generación de código fuente.
- Generar la estructura del SMA, integrando aspectos vinculados a la comunicación y coordinación entre los agentes,
- Proveer un mecanismo que permita modificar los códigos generados.

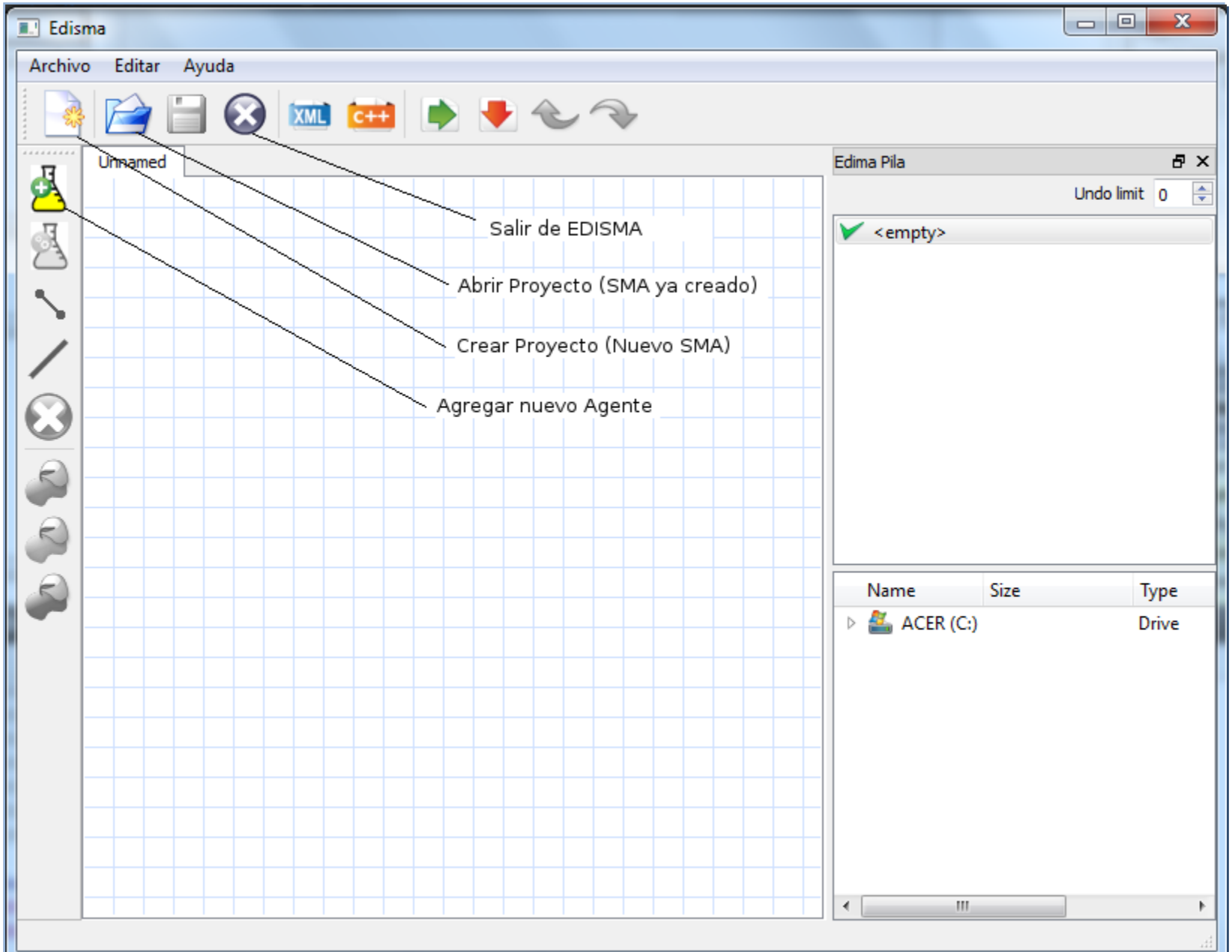


Servicios EDISMA

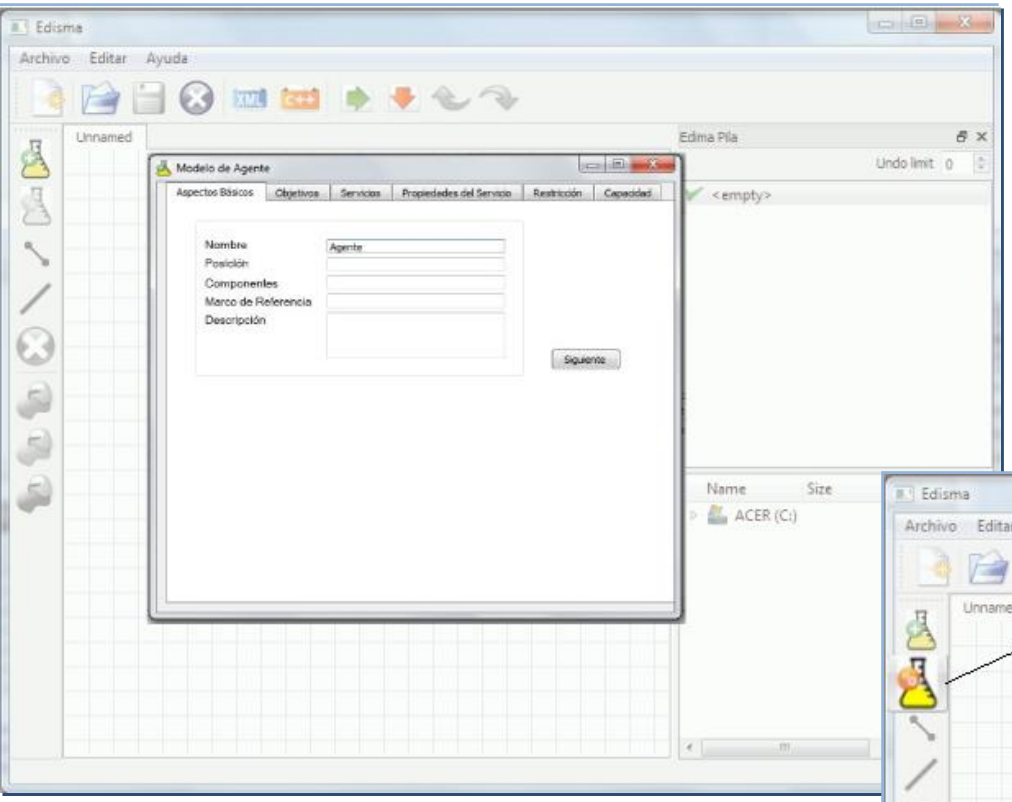
- Crear un SMA. En el contexto de la IGU, se asocia un SMA con un proyecto de software.
- Crear un agente.
- Crear un modelo de tarea a un agente.
- Crear un modelo de conversación entre agentes.
- Crear un modelo de comunicación entre agentes.
- Abrir un SMA elaborado creado previamente.
- Crear los archivos que permiten la implantación de los agentes en el MGS.
- Editar los archivos creados por EDISMA.


MASINA
EDISMA
makefile
Disparador
agente
*.cpp *.h
Main.cpp
agente
*.cpp *.h
Main.cpp
agente
*.cpp *.h
Main.cpp
agente
*.cpp *.h
Main.cpp
Procesos
Server
MGS
Procesos
Server
MGS
Procesos
Server
MGS
Nodo 1
Nodo 2
Nodo n

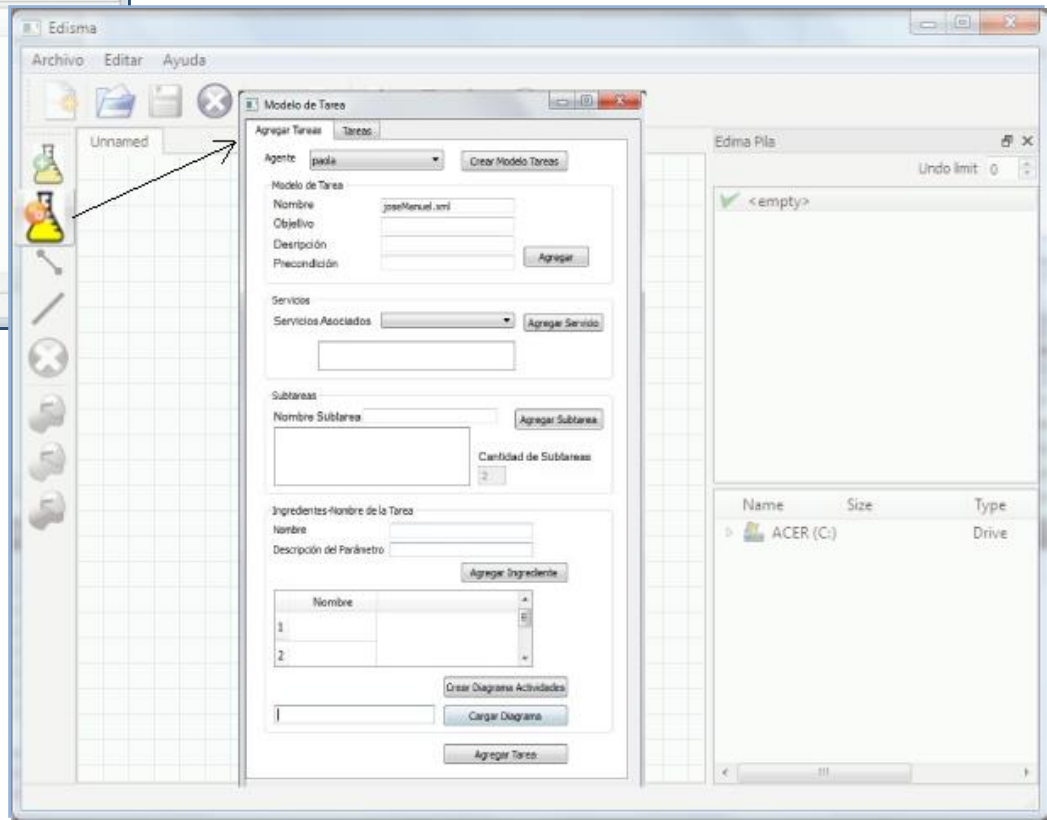
EDISMA


Edisma
Archivo
Editar
Ayuda
XML
C++
Unnamed
Salir de EDISMA
Abrir Proyecto (SMA ya creado)
Crear Proyecto (Nuevo SMA)
Agregar nuevo Agente
Edima Pila
Undo limit
0
<empty>
Name
Size
Type
ACER (C:)
Drive

EDISMA



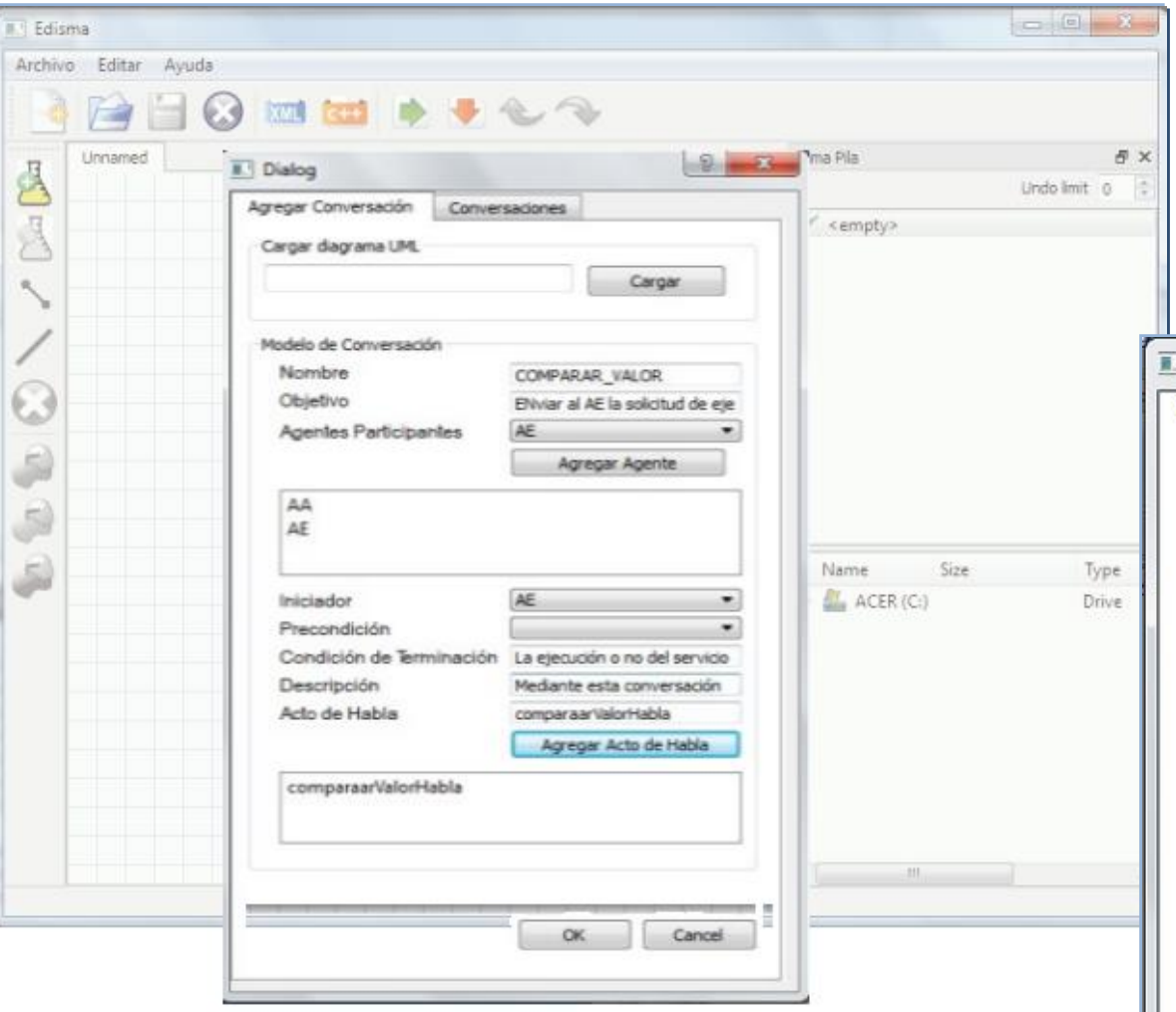
Modelo de agentes

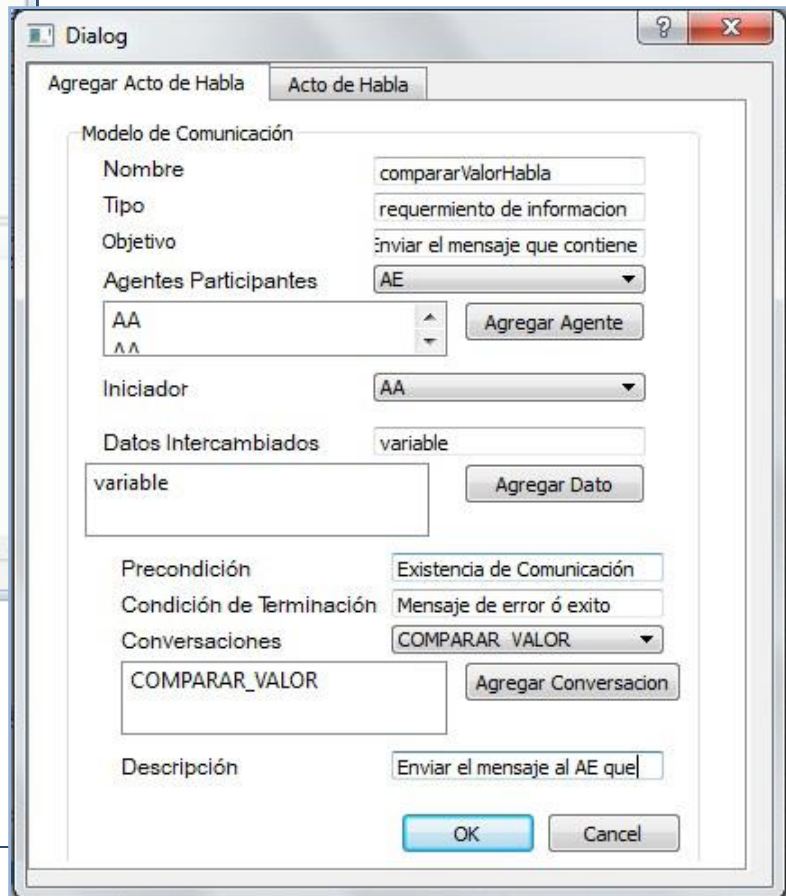


Modelo de tareas

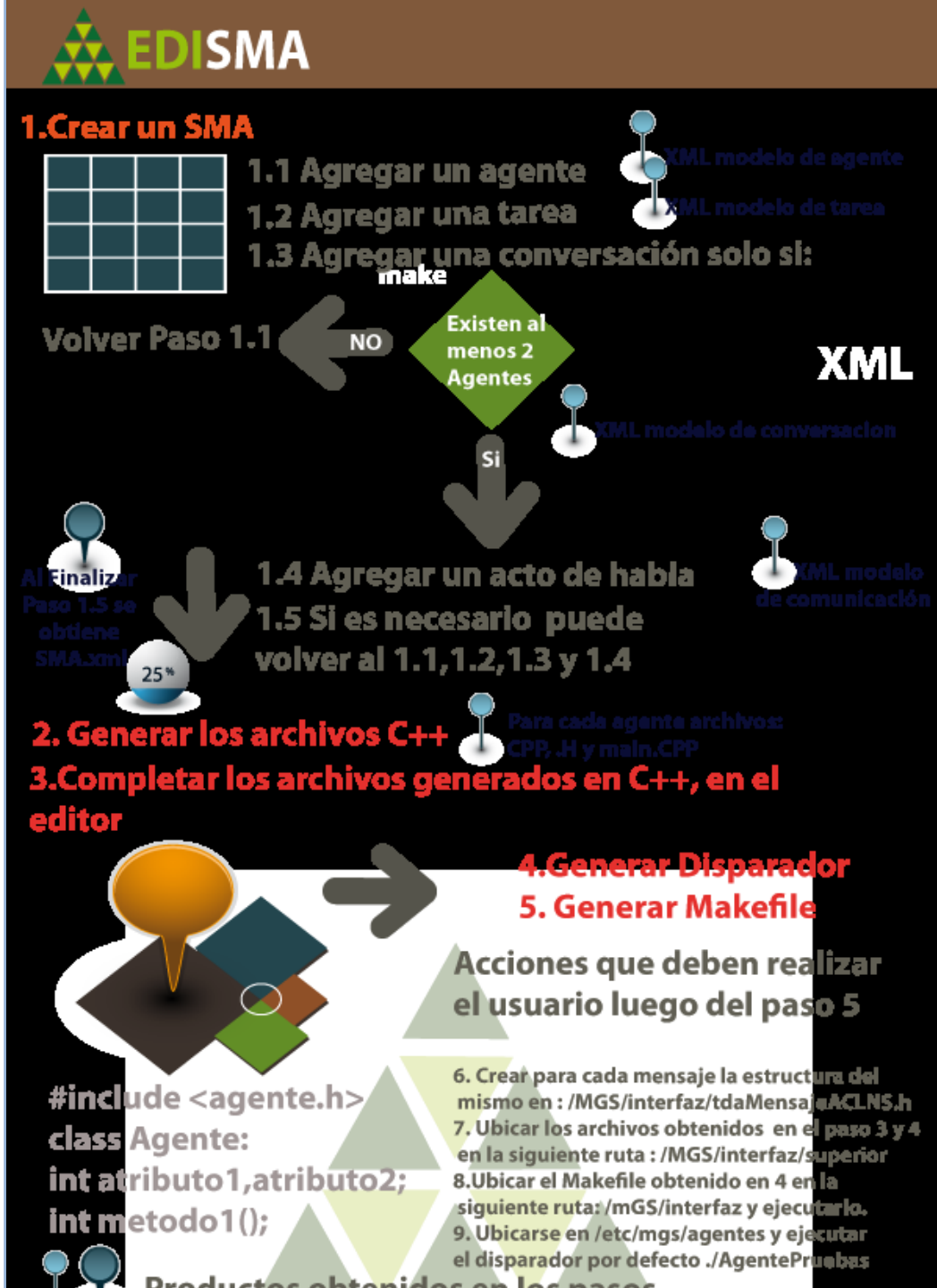
EDISMA

Modelo de conversación





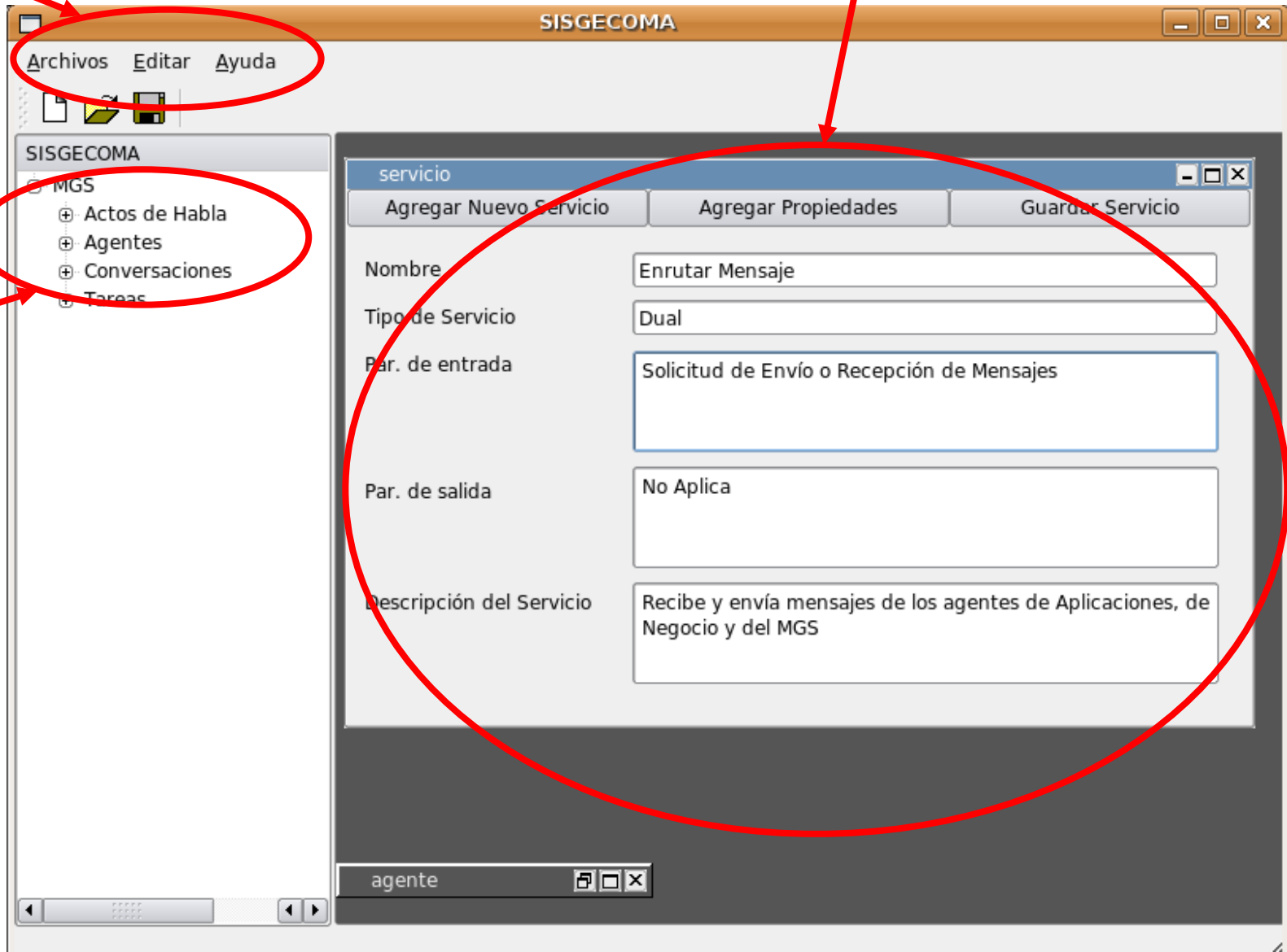
EDISMA


EDISMA
1.Crear un SMA
1.1 Agregar un agente
1.2 Agregar una tarea
1.3 Agregar una conversación solo si:
XML modelo de agente
XML modelo de tarea
make
Volver Paso 1.1
NO
Existen al menos 2 Agentes
XML
XML modelo de conversacion
Si
Al Finalizar Paso 1.5 se obtiene SMA.xml
25%
1.4 Agregar un acto de habla
1.5 Si es necesario puede volver al 1.1,1.2,1.3 y 1.4
XML modelo de comunicación
2. Generar los archivos C++
Para cada agente archivos: CPP, .H y main.CPP
3.Completar los archivos generados en C++, en el editor
4.Generar Disparador
5. Generar Makefile
Acciones que deben realizar el usuario luego del paso 5
#include <agente.h>
class Agente:
int atributo1,atributo2;
int metodo1();
6. Crear para cada mensaje la estructura del mismo en : /MGS/interfaz/tdaMensajeACLNS.h
7. Ubicar los archivos obtenidos en el paso 3 y 4 en la siguiente ruta : /MGS/interfaz/superior
8.Ubicar el Makefile obtenido en 4 en la siguiente ruta: /mGS/interfaz y ejecutarlo.
9. Ubicarse en /etc/mgs/agentes y ejecutar el disparador por defecto ./AgentePruebas

Sistema Generador de Código para la Metodología MASINA

Menú

Área de Trabajo



Árbol de Búsqueda

SISGECOMA

cemisid
centro de estudios en microelectrónica y sistemas distribuidos

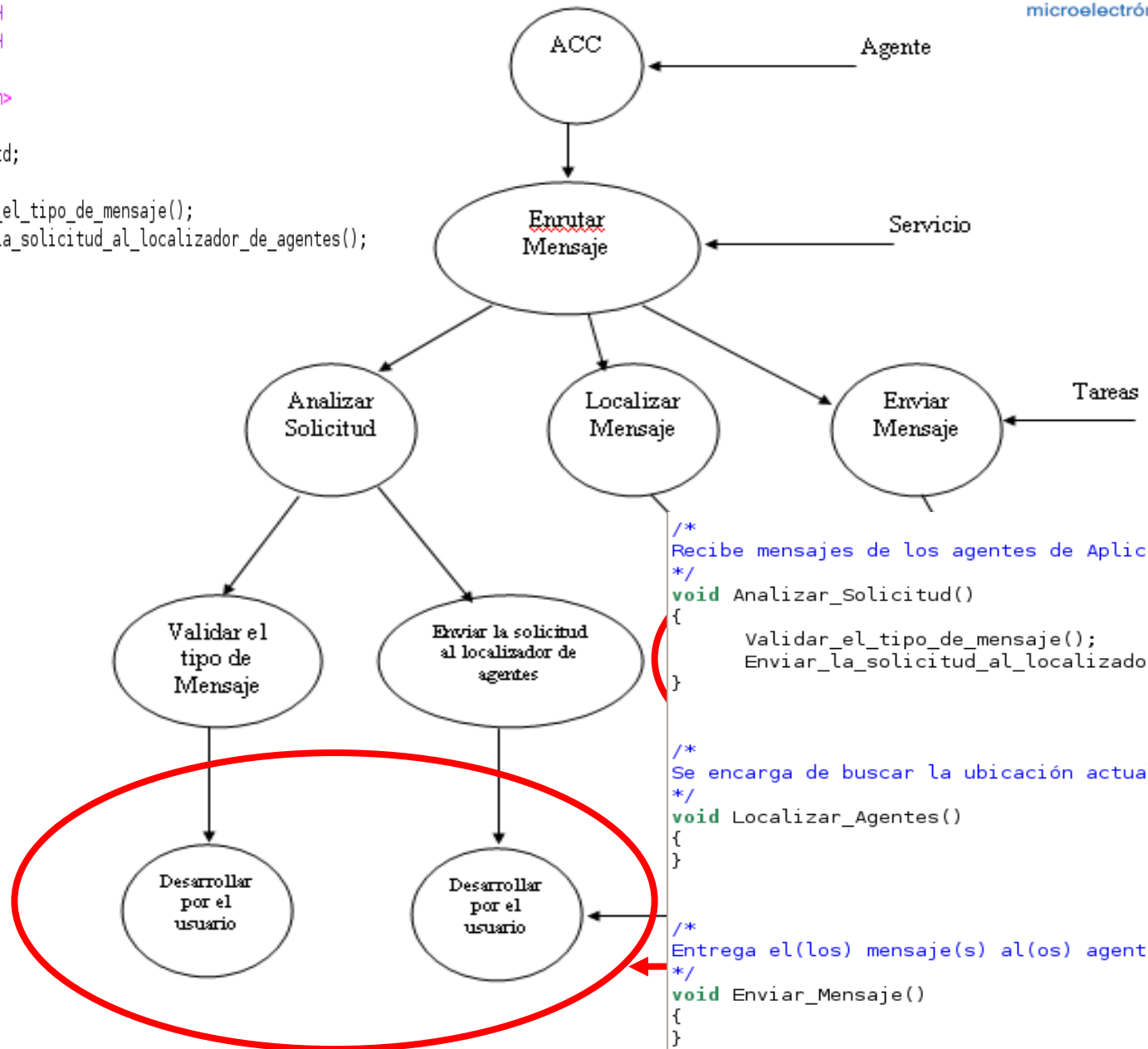
```
#ifndef SUBTAREA_H
#define SUBTAREA_H

#include <iostream>

using namespace std;

     void Validar_el_tipo_de_mensaje();
     void Enviar_la_solicitud_al_localizador_de_agentes();

#endif
```




```
/*
Recibe mensajes de los agentes de Aplicaciones, de Negoci
*/
void Analizar_Solicitud()
{
      Validar_el_tipo_de_mensaje();
      Enviar_la_solicitud_al_localizador_de_agentes();
}


/*
Se encarga de buscar la ubicación actual del(os) agente(s
*/
void Localizar_Agentes()
{
}


/*
Entrega el(los) mensaje(s) al(os) agente(s)
*/
void Enviar_Mensaje()
{
}
```

esclavo

SISGECOMA

cemisid
centro de estudios en microelectrónica y sistemas distribuidos

Servicios	Tareas
obedecer	T1 Reporte T2 Caminante

Servicios	Tareas
ordenar	T1 Manda

TAREA: Caminante	
Nombre	Caminante
Objetivo	Caminar
Descripción	Camina
Servicios asociados	obedecer
Precondición	
Sub-tareas	for(int i=1; i<11; i++) cout<<"estoy dando el paso"<<i<<endl;

jefe

```
void manda()
{
      cout<<"ponte a caminar";
}


void reporte()
{
      cout<<"hola, ya llegue"<<endl;
}


void caminante()
{
      for(int i =1; i<11;i++)
        cout<<"estoy dando el paso "<<i<<endl;
}
```