Estructuras de comunicación para la resolución de problemas de manera distribuida en la ingeniería concurrente

Leonid B. Sheremetov*

Resumen

Se presenta el problema de la comunicación en la resolución de problemas distribuidos. Se analizan las siguientes tecnologías de la comunicación: actores, pizarrones, las comunicaciones homogéneas y heterogéneas en un ambiente de multiagentes. Se ilustra la discusión con ejemplos del ambiente para el Diseño de Objetos Estructurados (Design of Structured Objects - DESO). *El prototipo actual integra algunos sistemas preexistentes de ingeniería que inicialmente utilizaban diferentes técnicas de comunicación en un marco común de ambiente de varios agentes que abarca múltiples sitios, subsistemas y disciplinas. La arquitectura general de sistemas está basada en los principios de la ingeniería concurrente. Se utiliza cada sistema individual para modelar diferentes etapas del desarrollo del sistema de producción y para razonar sobre ellos desde diferentes perspectivas de la ingeniería. Los sistemas interactúan por medio de lenguajes y servicios de comunicación basados en el conocimiento.*

Abstract

The author introduces the problem of communication in distributed problem solving. The following communication technologies are analyzed: actors, blackboards, homogeneous and heterogeneous communications in a multiagent environment for the Design of Structured Objects (DESO). The present prototype integrates some pre-existing engineering systems which initially used different communication techniques in a common framework of a multiple agent environment which covers multiple sites, subsystems and disciplines. The general architecture of systems is based on the principles of concurrent engineering. Each individual system is used to model different stages of the development of the production system, and to reason about them from different engineering points of view. The systems interact by means of languages and communication services based on Knowledge.

1. Introducción

Durante muchos años la investigación sobre la inteligencia artificial se ha orientado hacia las aplicaciones independientes con un conocimiento determinado y con una meta específica. Las aplicaciones se han desarrollado en un ambiente estático y sus actividades principales han sido las siguientes: recopilar información, planear y ejecutar algún plan para lograr su meta. Esta metodología ha resultado insuficiente debido a la inevitable presencia de un número de entidades que cooperan en el mundo real. La Inteligencia Artificial Distribuida *(Distributed Artificial Intelligence - DAI)* es un subcampo de la AI que intenta construir un modelo del mundo poblado por entidades inteligentes que interactúan por medio de la cooperación, la coexistencia, o por la competición. En primer lugar, los sistemas reales normalmente son tan complicados y contienen tanta información que es mejor reducirla a diferentes entidades cooperativas para lograr mayor eficiencia (por ejemplo modularidad, flexibilidad, y un tiempo más rápido de respuesta). De hecho, la construcción de una solución distribuida muchas veces se relaciona con la naturaleza del problema en cuestión. El primer camino está vinculado con la necesidad de tratar el conocimiento distribuido en las aplicaciones que están separadas geográficamente, tales como las redes sensoriales, el control de tráfico aéreo, o la cooperación entre robots [Davis 1983; Fehling 1980; Smith 1980, 1985; Lesser 1987]. La razón principal es que estas aplicaciones requieren de una interpretación y de una planeación distribuidas por medio de diferentes

* *Profesor-investigador de tiempo completo del Departamento de Posgrado de la UTM.*

censores inteligentes. Además de las aplicaciones que tienen que ver con las redes sensoriales, normalmente las tecnologías de procesamiento del conocimiento de manera distribuida se aplican a la automatización de sistemas, tales como los sistemas de manufacturas flexibles *(Flexible Manufacturing Systems - FMS)* [Parunak 1987], y a la cooperación entre los sistemas expertos en la ingeniería [Bond y Gasser 1988, Pavlin 1988, Klein 1991, Roth 1991]. Estas aplicaciones están motivadas por los aspectos tradicionalmente positivos de los sistemas de procesamiento distribuido (por ejemplo, el desempeño, la confiabilidad y el compartir recursos). Además, los tipos de campos de investigación como bases de datos distribuidas, la computación distribuida y paralela, trabajo cooperativo apoyado por la computadora, el diseño y la producción apoyados en la computación, ingeniería concurrente, así como la toma de decisiones distribuida, también tienen que ver con el proceso del tratamiento de la información distribuida.

Con la finalidad de mostrar las formas particulares para resolver los problemas que surgen en un ambiente distribuido vamos a ilustrarlo por medio de un ejemplo de la Ingeniería Concurrente (CE) [Serrano 1991, Smirnov 1994, Cutkosky *et al.* 1993]. Hemos explorado problemas que tienen que ver con la construcción de un marco amplio sobre las siguientes dimensiones: el desarrollo de interfaces, protocolos y arquitectura; compartir el conocimiento entre los sistemas que mantienen sus propias bases de conocimiento especializado y mecanismos de razonamiento, y el apoyo basado en la computación para la negociación y toma de decisiones que son características de la CE. Si bien el eje central para el desarrollo de ambiente DESO es la metodología de los agentes interactivos (los programas que abarcan las herramientas de la ingeniería), ahora comentaremos otras posibilidades para la comunicación basada en las estructuras tales como actores y pizarrones. En este documento se revisan nuestros experimentos iniciales en la simulación distribuida y en el rediseño creciente, se describe la arquitectura basada en agentes de DESO y se discuten tendencias futuras. Si bien hemos utilizado aplicaciones que nosotros diseñamos estamos conscientes del problema de la integración con otros paquetes, de tal manera que las normas aplicadas para la comunicación de agentes funcionan muy bien en este caso particular tal como se mostró en Cutkosky *et al.* [1993].

2. Una presentación del problema de diseño basado en el conocimiento distribuido

Si bien la ingeniería concurrente se recomienda casi universalmente hoy en día, es difícil de lograrla cuando se trata de grandes proyectos multi-disciplinarios. Consideremos el contenido de un problema de diseño. Tenemos una meta principal y un número de especificaciones y requerimientos que el sistema a diseñar debe comprender. Estos requerimientos descritos en términos de variables cuantitativas y cualitativas (figura 1), son planeados sobre los atributos de las entidades del sistema a lograr (*A, B* y etc.) dependiendo de un aspecto del diseño. Un aspecto define los atributos de una entidad particular que deben tomarse en cuenta de acuerdo al punto de vista del diseñador sobre un objeto a diseñar. Se describe cada entidad por medio de un conjunto de atributos y relaciones funcionales, las cuales conectan atributos dentro de una entidad (f3 y f4) y atributos de entidades de diferentes tipos (f1 y f2). Hacemos una distinción entre diferentes tipos de relaciones funcionales representando reglas y expresiones algebraicas. Se puede formular el problema de diseño de la siguiente manera: integrar el sistema a partir de un número de soluciones de patrón para cubrir los requisitos exteriores. Por lo tanto, cada entidad o componente de objeto diseñado puede obtenerse como un elemento a partir de un conjunto de soluciones de patrón. El problema puede ser solucionado de manera cooperativa por un equipo de diseño, siendo cada miembro responsable de un componente particular o de un conjunto de componentes a partir del punto de vista que depende de un aspecto de diseño.

En cualquier momento, los miembros del equipo pueden estar trabajando en diferentes niveles de detalle, cada

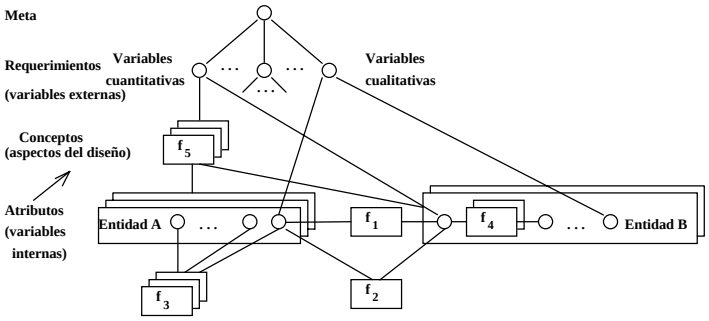


Figura 1. La descomposición del problema de diseño

uno empleando sus propias representaciones de artefactos físicos, modelos de ingeniería y conocimiento. A pesar de sus diferencias de perspectiva los especialistas comparten considerable información. Al aplicar tecnología de la información para apoyar los proyectos de ingeniería tales como éste, buscamos herramientas que ayuden a los miembros del equipo a compartir conocimientos y mantenerse enterados de las necesidades, restricciones, decisiones y suposiciones de cada uno.

Un diseño incorpora una gran cantidad de restricciones que se tienen que cubrir con el conjunto de componentes que conforman el diseño. Para asegurar que se cubran estas restricciones hay que organizar las descripciones de los componentes dentro de un marco representativo del diseño en desarrollo. Este marco es el modelo de diseño. Las descripciones en el modelo se componen a partir de las ontologías compartidas; es decir, los conjuntos de términos preestablecidos y significados con una descripción formal del ambiente de diseño. El modelo de diseño forma una base para compartir conocimientos entre diversos sistemas. Cualquier situación en donde se comparte debe tomar en cuenta el hecho de que estas herramientas no comparten el mismo modelo interno. Además, diseñar es un proceso de negociación: se toman y cambian las decisiones frecuentemente conforme se cambien las especificaciones y se planteen nuevas ideas. Por lo tanto podemos sacar conclusiones sobre la factibilidad de diseño por medio de la interacción de componentes distribuidos basados en el conocimiento, cada uno de ellos con su propia base local de conocimiento y mecanismos para razonar, y con facilidades de comunicación avanzada.

Se puede considerar a la distribución de componentes como una de las características claves de los sistemas basados en el conocimiento de este tipo, la cual influye enormemente sobre su eficiencia y se puede discutir desde los siguientes puntos de vista:

Distribución física. Se pueden realizar los componentes del sistema local por procesos separados (distribución lógica), por procesadores íntimamente acoplados de la arquitectura paralela, o por estaciones acopladas en forma flexible de redes globales o locales. El nivel y la organización de determinados sistemas dependen mucho de las metas de la arquitectura del sistema.

Distribución de conocimientos y de datos. Durante el proceso de la toma de decisiones en conjunto cada componente debe obtener información acerca del ambiente externo y las conclusiones y decisiones derivadas por otros componentes. La arquitectura en la cual todos los componentes obtienen información similar se ve sospechosamente útil ya que contiene operaciones superfluas en cada nodo. Las conclusiones en cada nodo dependen muchísimo de la disponibilidad de hechos y fuentes de conocimiento, estrategias aplicadas de control y estrategias de intercambio de información entre componentes. La distribución de fuentes de conocimiento está afectada también por la disponibilidad de conocimiento en diferentes nodos de la red. En general las fuentes de conocimiento que ya existen, con dificultad están disponibles para la operación distribuida.

Distribución de tareas. Cada componente como una unidad operacional tiene la capacidad para procesar varias tareas generalmente definidas por las metas. Debido a que cada nodo es capaz de procesar un número limitado de tareas (con frecuencia sólo una), la selección de métodos de evaluación de tareas y de emprendimiento tiene el impacto primario en la eficiencia de las acciones del componente.

Estrategia de intercambio de información. Cada componente puede generalmente apoyar su propia estrategia. Para el nodo transmisor ésta define cuándo, qué y en dónde se debe transferir la información. Para el receptor se aplican condiciones, tipos, fuentes y criterios

de estimación de información. La elección de una estrategia particular para cada nodo se define por la estructura organizacional.

Estructura organizacional. Define las reglas de preferencia para cada nodo que son independientes de las condiciones actuales de la solución de problemas generales. Junto con la estrategia de intercambio de información definen la estrategia para la resolución de problemas.

La solución de un problema de colaboración en el diseño distribuido parece ser inalcanzable sin la implementación de una estrategia de comunicación avanzada, cuyos tipos principales se van a ver en la siguiente sección.

3. Estructuras de comunicación para CE: observaciones generales sobre las metodologías

Las arquitecturas específicas de comunicación que se proponen para apoyar la toma de decisiones en forma distribuida se dividen en dos clases con respecto a la estructura de comunicación:

Sistemas de pizarrón. La comunicación entre diferentes fuentes de conocimiento o unidades de procesamiento de conocimientos se lleva a cabo a través de una estructura de conocimiento compartido llamada pizarrón por medio de la colocación y lectura de información en el pizarrón.

Sistemas de transferencia de mensajes. Un módulo de razonamiento envía recados (solicitudes o respuestas) a uno o más módulos cuyos nombres normalmente ya se conocen de antemano.

Si bien estas técnicas parecen ser diferentes, en la práctica ofrecen un apoyo similar para el razonamiento distribuido y pueden realizar acciones semejantes de comunicación. Se puede decir que los sistemas de pizarrón y los parámetros de actores basados en mensajes [Hewitt 1977, Erman 1980, Nii 1986a, Nii 1986b] estuvieron entre los primeros conceptos de DAI que construyeron un sistema modular con instalaciones implantadas para la colaboración.

3.1 Marco de comunicación por pizarrones

Se puede considerar al marco de pizarrón como una posible marca para la integración de un sistema de procesamiento de conocimiento distribuido, el cual está formado por componentes con representación del conocimiento cambiable. Se puede utilizar esta información sobre diferentes niveles de abstracción integrando fuentes de conocimiento con diferentes modelos de cálculo e inferencia. La estructura de pizarrón para la resolución de problemas surgió del discurso de Hearsay sobre el sistema de comprensión [Erman 80, Lesser 75, Nii 86a], cuando lo que ahora consideramos como una arquitectura de pizarrón estándar fue implementada en el proyecto Hearsay-II. La idea en donde se apoya la metodología de pizarrón es sencilla: un conjunto de módulos independientes llamados fuentes de conocimiento *(knowledge sources - KS)*, que contienen conocimiento específico del dominio; un pizarrón a través del cual los KS se comunican entre ellos, y un sistema de control o monitoreo de pizarrón que determina la secuencia en la cual los KS operan en el pizarrón (figura 2).

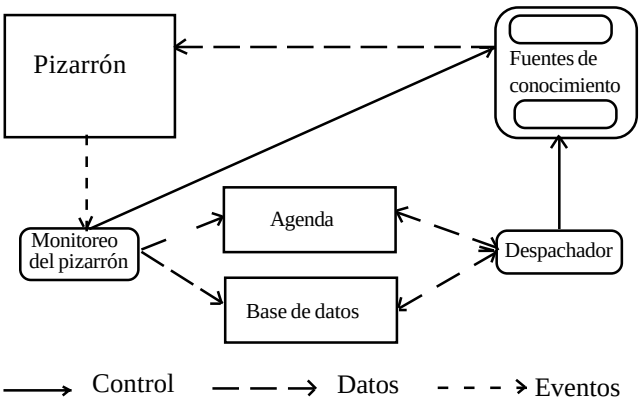


Figura 2. La arquitectura estándar del pizarrón

Como un ejemplo de la arquitectura de comunicación centralizada, apoya la estructura del sistema modular, pero desarrolla una comunicación intensa a través del pizarrón y por lo tanto propicia un embotellamiento del sistema. Otro aspecto del problema recae en el control de pizarrón. El monitoreo de pizarrón basado en temarios para identificar correctamente a las KS que se deberían accionar debe contener toda la información sobre las clases de

eventos que le interesan a cada agente. Una de las posibles soluciones del problema se puede plantear en pizarrón integrado y el intercambio de mensajes especialmente en estructuras orientados a objetos. Para hacerlo, por un lado tenemos que mejorar el paradigma de Programación Orientada a Objetos *(Object Oriented Programming - OOP)* aumentando la autonomía del objeto, incluyendo semántica en los mensajes y finalmente presentando las capacidades de aprendizaje y toma de decisiones. En la escala completa vamos a reconsiderar esta cuestión más tarde cuando hablemos sobre las arquitecturas basadas en agentes. El otro aspecto de esta integración es mejorar el control de pizarrón y las estrategias de comunicación empleando un nivel horizontal (basado en el intercambio de mensajes) de comunicación entre los KS.

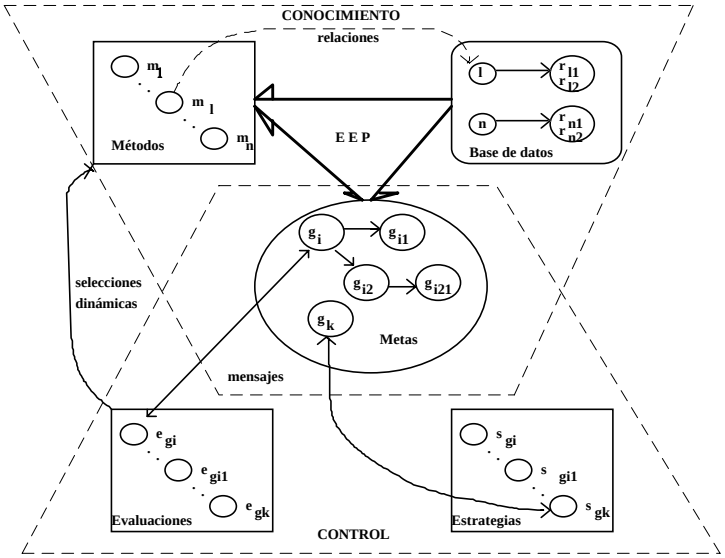


Figura 3. La arquitectura del pizarrón con una implicada estrategia de redes de metas dinámicas

Con el fin de ejemplificar esta estrategia consideremos una estructura representada en la figura 3, la cual apoya el dominio vertical, usualmente utilizado en pizarrones, y la colaboración horizontal por medio de la transmisión de mensajes. Los objetos son entidades tridimensionales compuestos de apuntadores, componentes (metas) y métodos. Los apuntadores se utilizan para definir las relaciones entre las fuentes de conocimiento multiexperto. Los componentes, para la representación de objetos complejos o la descomposición de metas; los métodos, para una elección adaptativa de la estrategia para la resolución de un problema.

Las metas se representan por medio de clases de objetos con herencia de métodos y cada uno de ellos define la estrategia para lograr una meta. Estos métodos aplican una planeación global parcial (PGP), búsqueda al azar y estrategias del proceso de emancipación de entidades (EEP). La EEP proporciona una oportunidad para aplicar dos estrategias diferentes, así como hacer inferencias a partir de datos y métodos hacia las metas, y de datos y metas hacia métodos. Los métodos y las metas pueden conectarse con entidades en la base de datos *(Data Base - DB)* (por ejemplo n y l en la figura 3), o pueden ser genéricos. Cada objeto es capaz de generar automáticamente metas y submetas relacionadas, constituyendo una red de metas dinámicas, que pueden lograrse por medio de diferentes componentes del ambiente. Cada componente define su capacidad para resolver un problema dentro de la estrategia general de solución distribuida.

Los mensajes se clasifican en 3 grupos: apuntadores, que definen los mensajes entre objetos creados por diferentes expertos; composiciones, mensajes para un objeto complejo o para una de sus partes; elecciones, mensajes para una instancia de un objeto. Se utilizan para implementar el proceso de colaboración horizontal, involucrando múltiples fuentes de conocimiento y estrategias de inferencia.

Si bien esta metodología maneja el problema de la distribución de control, tiene restricciones particulares sobre los esquemas de representación del conocimiento y se puede aplicar a los sistemas homogéneos basados en el conocimiento. Desde el punto de vista de la implementación, se puede emplear con éxito solamente en situaciones donde las fuentes o componentes del conocimiento se desarrollen dentro del mismo marco de aplicación. Se puede encontrar una discusión más completa sobre el modelo de pizarrón en Carver 1992, Engelmore 1988.

3.2 La comunicación de componentes utilizando estructura de actores

Los actores representan uno de los modelos convencionales para la computación paralela, basado en la transmisión de mensajes asincrónicos [Agha 1986, Agha 1988, Agha *et al.* 1993]. Los actores son objetos activos que procesan mensajes, crean nuevos actores o los activan por medio de la transmisión de mensajes como resultado del procesamiento de estos últimos. Se define la reacción particular del actor por su memoria local y contenido del mensaje. Los mensajes apoyan un modo asincrónico de transferencia entre los actores, lo cual significa que un actor al enviar un mensaje pasa a procesar el siguiente sin esperar respuesta. Los receptores del mensaje actual pueden ser aquellos actores que tienen relaciones con el actor-fuente o pueden estar incluidos como parámetros de los mensajes.

Las acciones del actor se definen por su repertorio (escenario) único que contiene una descripción de su comportamiento que abarca las siguientes alternativas de actividades: modificación de estado, actualización de la lista de destinatarios, el envío de mensajes y la creación de nuevos actores. El proceso de comunicación está basado en nombres no repetidos con que cuentan los actores de acuerdo con los papeles que desempeñan. Esto significa que si el mismo actor A resulta destinatario de un número de actores o mensajes $X_1,...,X_n$, cada uno de los X_i le asignará un nombre único. O sea, A desempeña su papel en el repertorio de X_i. Un sistema de actores contiene conjuntos finitos fijos de estados de actores, significados de mensajes, nombres de actores y nombres de comunicación.

Una definición formal del sistema de actores incluye el vocabulario de actores (una representación formal de todos los estados de actores, todos los mensajes y todos los nombres en el sistema), grafo de configuración (representación formal del estado instantáneo del sistema en términos de vocabulario), gramática de actores (un caso especial de gramática de grafos para las transformaciones de grafo de configuración). El modelo bien definido para los sistemas de actores con cálculos paralelos tiene claras desventajas cuando es aplicado al procesamiento de conocimiento de manera distribuida. Una de ellas es la semántica indefinida. Para resolverlo se introduce el cálculo de procesos de CE y el álgebra de procesos concurrentes, basada en la gramática de grafos sensible al contexto, los cuales describen los mecanismos de coordinación y de comunicación de los procesos de CE [Koulinitch 1995] (véase la sección 4.2).

3.3 Tecnología de agentes inteligentes autónomos

Para alcanzar sus metas, los componentes distribuidos deben contar con suficientes medios para resolver los problemas de comunicación que surgen en la toma de decisiones distribuida. Cuando analizamos los marcos de comunicación por pizarrón y basada en actores, mencionamos la necesidad de aumentar sus capacidades para resolver problemas. En realidad, para cooperar eficientemente en el proceso de la toma de decisiones en CE, los componentes deben tener las siguientes propiedades:

- Ejecutar acciones basadas en el análisis de su estado interno, metas, hipótesis y el conocimiento;
- Planear sus acciones de acuerdo a su conocimiento del estado interno, metas, hipótesis y conocimiento de otros objetos;
- Planear su acción con base en el procesamiento de mensajes de otros objetos, tomando en consideración el conocimiento sobre sus calificaciones y compatibilidades;
- Participar en el proceso de colaboración con otros objetos en la toma de decisiones y en la planeación de acciones con base en el consenso logrado;
- Otorgar la posibilidad de crear copias de objetos con diferente repertorio (funcionalidad);
- Otorgar la posibilidad de crear un espacio virtual de inferencia deductiva.

Para manejar estas propiedades se introdujo la noción del agente inteligente autónomo *(Intelligent Autonomous Agents - IAA).*

3.3.1 Un concepto del agente

El concepto de agencia se puede considerar como uno de los más importantes y emocionantes en los 90's, tanto en la Inteligencia Artificial como en las tendencias principales de la Ciencia de la Computación. La tecnología de agentes está a punto de alterar radicalmente no sólo la forma en la que interactuamos con las computadoras, sino también en la manera en que se conceptualizan y se construyen los sistemas complejos [Maes 1990a, Wooldridge and Jennings 1994]. La interacción de los agentes está basada en los conceptos y la terminología compartidos para comunicar el conocimiento entre diferentes disciplinas, y un lenguaje de comunicación y de control que permita a los agentes solicitar información y servicios. Esta tecnología habilita a los agentes que trabajan en diferentes aspectos de un diseño para interactuar a nivel de conocimientos: compartiendo e intercambiando información acerca del diseño independiente del formato en que se codifique internamente la información.

Quizá la manera más general en que se utiliza el término agente es para denotar un hardware, o (con más frecuencia) un sistema de software que goza de las siguientes propiedades:

Autonomía: los agentes operan sin la intervención directa de los seres humanos u otros, y tienen alguna clase de control sobre sus acciones y el estado interno;

Capacidad social: los agentes interactúan con otros agentes (y posiblemente con los seres humanos) a través de un tipo de lenguaje de comunicación entre agentes [Genesereth y Ketchpel 1994];

Reactividad: los agentes perciben su ambiente (el cual puede ser el mundo físico, un usuario a través de un interfaz gráfico, una colección de otros agentes, la INTERNET, o quizás todas estas combinadas) y responden de una manera oportuna frente a los cambios que ocurren en él;

Pro-actividad: los agentes no simplemente actúan como respuesta a su ambiente, sino que son capaces de exhibir un comportamiento dirigido a metas por medio de la toma de iniciativa.

Existe un número de conceptos sobre un agente dependiendo del punto de vista sobre el procesamiento del conocimiento de manera distribuida. Desde el punto de vista del procesamiento distribuido se define al agente como *un proceso de software autosuficiente ejecutando en forma concurrente, que encapsula algún estado y tiene la capacidad para comunicarse con otros agentes por medio de la transmisión de mensajes.* En este sentido se le puede considerar como un desarrollo natural del paradigma de programación concurrente basada en objetos [Agha *et al.* 1993, Genesereth y Ketchpel 1994].

La idea de despachar un programa en una computadora remota es algo viejo. A principio de los años 60's se empleaba esta idea para que las mini-computadoras enviaran trabajos por lotes *(batch-jobs)* sobre unidades centrales *(mainframes)* y recibieran los resultados para procesamiento local. Más tarde, en los 70's, se despachaban escritos ejecutables entre redes de mini-computadoras para permitir el procesamiento distribuido en tiempo real. Ahora se desarrolla esta idea como la base de una tecnología de agentes móviles o ambulantes, con la aplicación de escritos móviles tales como Telescript, en donde normalmente se escribe el contenido programático del agente [Chess 1994, White 1994]. De los marcos de programación distribuida y orientada a objetos muchas veces se puede reducir enormemente la necesidad de transportar los métodos, cuando se puede suponer que los métodos comunes estén lejanamente disponibles por medio de transporte de un conjunto de referencias de objeto, un conjunto de datos como muestra y un estado de proceso. Otro ejemplo de ese tipo es un softbot o robot de software *(software robot)*. Un softbot es un agente que interactúa con un ambiente de software dando órdenes e interpretando la retroalimentación del ambiente. Los ejecutores de un softbot son órdenes destinadas a cambiar el estado externo del ambiente. Los censores de un softbot son instrucciones orientadas a proveer información [Etzioni *et al.* 1994].

Otra preocupación se asocia con el punto de vista del dominio de AI, donde el término "agente" tiene un

significado más fuerte y específico que el mencionado anteriormente. De acuerdo a esto un agente es un sistema computacional que, además de tener las propiedades identificadas arriba, se conceptualiza o implementa por medio del empleo de conceptos que se aplican con más frecuencia a los seres humanos. Por ejemplo, es bastante común que la AI caracterice a un agente utilizando conceptos mentales, tales como el conocimiento, la creencia, intención y obligación [Shoham 1993]. Algunos investigadores en AI han ido más allá y los han considerado como agentes emocionales [Bates 1994, Bates *et al.* 1992a, Maes 1994a]. Por razones obvias, tales agentes son de especial importancia para aquellos que se interesan por las interfaces de computadora-seres humanos.

Otros atributos se discuten algunas veces en el contexto de agencia. Por ejemplo, como ya mencionamos, la movilidad es la capacidad con que cuenta un agente para desplazarse dentro de una red electrónica [White 1994]; la veracidad es la suposición de que un agente no comunicará, a sabiendas, falsa información [Galliers 1988a]; la benevolencia es la suposición de que los agentes no tienen metas en conflicto, y que cada agente por lo tanto siempre trata de hacer lo que se le pide [Rosenschein y Genesereth 1985], y la racionalidad es (en forma vulgar) la suposición de que un agente actuará para lograr sus metas, y no actuará para obstaculizar su logro -por lo menos en la medida en que sus creencias lo permitan- [Galliers 1988b].

3.3.2 Definición del sistema de programación orientada a agentes

Se puede considerar al sistema de programación orientada a agentes *(Agent Oriented Programming System - AOPS)* como una especialización de la OOP [Shoham 1990, Shoham 1992]. Los agentes son objetos que tienen estados mentales (como creencias, deseos e intenciones en PLACA [Thomas 1993], o creencias, capacidades y compromisos [Davies 1991]) y un concepto de tiempo. Los compromisos son programados utilizando reglas de compromiso. Éstas son reglas sencillas con encadenamiento hacia adelante que conectan los mensajes, el estado interno del agente y sus acciones. Las creencias son afirmaciones lógicas acerca del mundo que el agente considera como falsas o verdaderas. Las capacidades son acciones que el agente es capaz de ejecutar. Los compromisos son garantías de que un agente llevará a cabo una acción en un momento determinado.

Para hacer realidad la estructura de compromiso se proporcionan tres tipos básicos de mensaje: *informar* (inform), *solicitar* (request) y *cancelar la solicitud* (unrequest). *Informar* permite a los agentes comunicar estados mentales con otros agentes, mientras que *solicitar* permite a un agente pedir a otro agente que ejecute una acción, y *cancelar la solicitud* cancela una solicitud. Las reglas de compromiso consisten en las condiciones antecedentes que se combinan con los mensajes que se van recibiendo y los estados internos del agente (por ejemplo las creencias). Si se acciona una regla el agente generará un compromiso para: hacer una acción privada, abstenerse de una acción, o enviar un mensaje (es decir *tell*, etc.). Luego se ejecutan estos compromisos en el momento especificado por el agente que haya enviado la solicitud. Se describe el AOPS por un conjunto de fuentes autónomas de conocimiento, cada una de ellas representa una tupla de componentes:

$$AOPS = (M', K', A', I', L', S', F', G', R', C'), \qquad (1)$$

donde M' es un conjunto de posibles acciones de IAA, definidas por mensajes que forman un vocabulario de agentes;

K' es un conjunto de conocimientos que pertenece a un agente en forma de reglas y hechos acerca de cualquier evento descrito en su base de conocimientos local *(Local Knowledge Base - LKB)*. De no ser así, un agente no puede ejecutar una acción, definida por un mensaje M', que no está descrita en su LKB;

A' es un conjunto de atributos de los agentes, definidos en una LKB;

I' es un conjunto de procedimientos de inferencia, definidos por este agente, por ejemplo, en el caso del

razonamiento basado en restricciones, consiste en un mecanismo de satisfacción de restricciones;

L' es un conjunto de lenguajes para describir los estados internos de los métodos, de los atributos y de los conocimientos, así como los lenguajes para la comunicación con el ambiente externo (otros agentes);

S' es un conjunto de operadoras para el envío de mensajes que están disponibles para la interacción con otros agentes;

R' es un conjunto de mecanismos para recibir mensajes basados en un patrón, que realiza un esquema de cálculo secuencial con retrasos para la espera de respuestas para sus propios mensajes y la contestación de los mensajes de otros agentes;

F' es un conjunto de mecanismos funcionales de herencia;

G' es un conjunto de conocimiento global compartido por un grupo de agentes. Se puede realizar por medio de un agente de repositorio especial que posee este tipo de conocimiento y lo distribuye directamente entre otros agentes por medio del procesamiento de sus mensajes;

C' es un conjunto de procedimientos de bajo nivel, que incluye las utilidades del sistema, las utilidades de la transmisión de sus mensajes, etc.

Cada agente de AOPS puede definirse de la siguiente manera:

$$agent_i = (m_i, k_i, a_i, i_i, l_i, s_i, r_i, g_i). \qquad (2)$$

Los mecanismos de herencia (F') y las llamadas de sistema (C') no pertenecen a la descripción de un agente, como las características de AOPS. AOPS provee una herencia funcional por medio de un mecanismo de reproducción contrario a las jerarquías de clase/instancia como se usa tradicionalmente en OOP. Cada agente tiene la capacidad de crear copias completamente funcionales de sí mismos capaces de modificar su propio repertorio de comportamiento, sin cambiar el de su clase de origen. La herencia múltiple hace posible crear muestras de agentes mezclando el repertorio de comportamiento, el modelo de herencia y los atributos de las clases de origen.

3.3.3 Lenguajes de programación en la estructura general del agente

Un agente definido anteriormente, puede ser implementado por medio de los lenguajes de programación *(L')* de las dos formas siguientes. La primera define el contenido programático de un agente e incluye el lenguaje de programación (definido por las funcionalidades *(M')* y *(A')*) y el lenguaje de representación del conocimiento (el cual describe las funcionalidades *(K')*, *(I')* y *(G')*, proporcionando los medios para expresar las metas, tareas, preferencias, creencias y vocabularios apropiados para el dominio de un problema). El segundo tipo define las capacidades de comunicación de un agente (las funcionalidades *(S')* y *(R')*), proporcionando interacción dentro del ambiente distribuido (figura 4).

La estructura del agente articula las características particulares que los agentes tienen que cubrir, por lo tanto los lenguajes del agente tienen que reflexionar sobre ellas. Por ejemplo los lenguajes de programación del agente tienen que proporcionar una capacidad de abstracción del objeto para el control de acceso y la movilidad de datos, repartir su procesamiento entre múltiples reproducciones de ellos mismos, etcétera. Por otro lado, un papel de un agente tiene que definir no sólo el aspecto conductual de un agente, sino también la interacción con otros agentes. Los lenguajes modernos de programación de los agentes (llamados *script-lenguages*), por ejemplo, *Save-Tcl* y *Telescript*, tienen sin embargo capacidades limitadas mientras describen el aspecto cooperativo del trabajo de un agente. Es por esto la necesidad para que un lenguaje de comunicación de agente sea considerado como un nivel separado de la descripción del agente.

Dependiendo del concepto de agencia que se insinúa, un agente puede tener todas estas capacidades o sólo algunas de ellas. Tomando en consideración el significado de las características de comunicación de la agencia [Singh 1991] y las metas de este artículo, vamos a concentrarnos en las siguientes secciones sobre la especificación de los niveles de comunicación de los agentes.

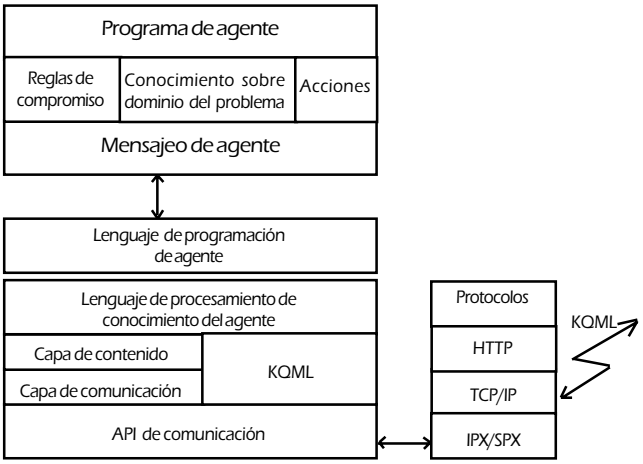


Figura 4. La arquitectura general del agente

3.3.4. Interacción de agentes en la capa de la sesión

Todas las interacciones de los agentes están basadas en el concepto de la transmisión de mensajes. Este mecanismo consta de los siguientes componentes: destino, operador de envío de mensajes, tipo de mensaje y mecanismos de recepción.

De acuerdo con la clasificación de mensajes se pueden considerar los siguientes tipos: mensaje individual, conjunto de mensajes, mensaje modificado con modificadores conjuntivos, disyuntivos y "¿quién es el remitente?". Se pueden realizar estos mensajes por los siguientes tipos de direcciones:

- a un agente particular,
- el mensaje transmitido "a cada agente",
- el mensaje transmitido "al primer agente",
- el mensaje transmitido "recoge los resultados",
- el mensaje transmitido "consenso",
- el mensaje transmitido "a los agentes enlistados",
- el mensaje de transmisión general.

El destinatario de un mensaje puede ser un agente, un conjunto de agentes, modificado por el funcionario que define la manera de interoperar. Por un agente individual es la manera más sencilla de dirigir un mensaje:

msg(agent1,get_ancestors(AllAncestors).

Dependiendo del lenguaje de programación del agente se le puede tratar a este código, por ejemplo, como una cláusula CLP(R) (solicitud) acerca de un *AllAncestors* (todos los ancestros) de una variable lógica, delimitado por el agente 1 y devuelto al remitente automáticamente. Este código puede ser también parte de un método que necesita la información del agente 1 acerca de los ancestros para continuar con los cálculos.

En caso de que un agente tenga una lista de destinatarios se utiliza un tipo de mensaje transmitido. Normalmente se utiliza este tipo de mensaje para obtener una confirmación del grupo de agentes al terminar una acción:

msg(agent1, agent2, agent5, clear_ kb).

En este caso el mensaje se les envía solamente a los destinatarios de la lista. En muchos casos es necesario recibir sólo la primera respuesta de un grupo de agentes. Para hacerlo se utiliza un mensaje de transmisión restringida:

msg(first_one(agent1, agent2, agent5 , Handler, diagnose (TaskList, Diagnose)).

EL primero de los agentes capaz de manejar el mensaje *diagnose* (diagnosticar) delimita los variables *Handler* (operador) y *Diagnose* (diagnosticar). El tipo OR de interacción entre agentes se realiza de esa manera.

Para recolectar resultados de un grupo de agentes se utiliza el siguiente mensaje:

msg(collect (agent1, agent2, Variable, ResultList), get_ancestors(Ancestors)).

Aquí *Variable* define una variable que necesita resultados, y se forma la lista de resultados por medio de dos sublistas que incluyen pares: la lista *agent_name/ancestor* para cada agente (*agente1, agente2*) consecutivamente.

El mensaje emitido *consensus* (consenso) sirve para definir el grado de conformación entre las respuestas:

msg(consensus(agent1, agent2 , get_parent (name)).

Este mensaje define si el *agente 1* y el *agente 2* tienen el mismo nombre de padre. En CLP(R) es posible ir hacia atrás para responder con un *falso* o *verdadero* en caso de

que se logre o no el consenso. En este caso la unidad de propagación de restricciones tiene el papel de moderador (o mediador) y los agentes no se interactúan directamente. Algunas veces en vez de obtener una respuesta acerca del consenso es necesario recibir la lista de agentes que dan respuestas afirmativas o negativas. Aquí se utiliza el tipo de mensaje de transmisión *enumerado*:

msg(consensus(agent1, agent2, YesAgents) query_kb (President(name))).

Este mensaje entrega las listas de agentes que tienen la capacidad para adoptar o negar esta afirmación utilizando su LKB.

Se utiliza el mensaje de transmisión general cuando algún agente no sabe exactamente con quién comunicarse para la información que se necesite:

msg(broadcast(All)diagnose(TaskList, Diagnose)).

Se le asignará a la variable *All* un valor del nombre del primer agente que responde, y a *Diagnose* un valor asignado por el agente. En caso de que se use vuelta atrás se pueden recibir valores de otros agentes (si no nos importa el tiempo y los recursos).

Los siguientes tipos de mensajes utilizan modificadores de conjunción y disyunción:

msg(agent1, and(executeTest1, executeTest2, execute Test3)

msg(agent1, or([executeTest1, executeTest2, executeTest3]))

En el segundo caso el proceso de ejecución se detiene cuando se cumpla la primera operación anotada. El uso de modificadores es de gran importancia para aumentar la funcionalidad de comunicación entre los agentes. Se puede utilizar el último tipo de mensaje para propósitos de seguridad:

msg(agent1, wholsHost(agent2, solve(problem 1,Solution))).

Es necesario un lenguaje expresivo y accesible para especificar el conocimiento y las intensiones de los agentes para que se comuniquen con éxito, de tal manera que puedan ser entendidos correctamente por otros.

3.3.5 KQML: especificación de capas de contenido y comunicación

En esta sección se proporciona un breve resumen de los lenguajes de comunicación entre los agentes utilizados en la arquitectura de IAA. Representan la funcionalidad de S' y R' del modelo descrito anteriormente. El sistema IAA, como muchos otros sistemas computacionales, está estructurado como colecciones de procesos independientes, con frecuencia distribuidos en múltiples anfitriones enlazados por una red. Además, en los sistemas de redes modernos debería ser posible construir nuevos programas extendiendo los sistemas existentes; cada nuevo proceso debería tener la capacidad de enlazarse sin mayores problemas a las fuentes de información y herramientas existentes, tales como los sistemas basados en conocimientos o filtros. Esto es extremadamente importante para las aplicaciones de CE, las cuales con regularidad tienen que integrar a los módulos existentes. Una interacción compleja del software, como la interacción basada en el conocimiento (por ejemplo, acertar, retractar, evaluar), y la ubicación de servicio que se basa en la funcionalidad, se sujetan a los acuerdos *ad hoc* que establecen entre las comunidades individuales de programadores. En esta situación la interoperación está limitada a las colecciones de programas diseñados para interactuar entre ellos mismos.

Por lo tanto el bajo nivel de comunicación está bien diseñado y permite integrar sin dificultad y flexiblemente las diversas plataformas (sistemas operativos, mecanismos de control, formatos de datos/conocimiento y las presuposiciones ambientales). La integración está basada en las arquitecturas de Procesamiento Distribuido Abierto *(Open Distributed Processing - ODP),* por ejemplo, el mecanismo de Intercambio Dinámico de Datos *(Dynamic Data Exchange - DDE)* para Windows, el cual provee formatos comunes de datos y mecanismos para la mensajería de interaplicación. Aunque las arquitecturas ODP definen la forma de la interoperación del software, dicen muy poco sobre el contenido de la información compartida.

Las normas y las metodologías de intercomunicación, como el OMG/CORBA (Object Management Group´s/ *Common Object Request Broker Architecture),* ILU, OpenDoc, OLE, etc. [OMG 1991, Curtkosky *et al.* 1993] son intentos que se promulgan con frecuencia como soluciones para el problema de comunicación entre agentes. Impulsar tal trabajo es una de las dificultades para ejecutar las aplicaciones en ambientes distribuidos y dinámicos. La preocupación principal de estas tecnologías es asegurar que las aplicaciones puedan intercambiar estructuras de datos e invocar métodos remotos a través de plataformas disparadas. Aunque los resultados de estos esfuerzos estándar serán útiles en el desarrollo de los agentes de software, no proporcionan respuestas completas a los problemas de comunicación entre agentes. A fin de cuentas los agentes inteligentes son más que colecciones de estructuras de datos y métodos sobre ellos. Por lo tanto los estándares y protocolos se entienden mejor como un substrato sobre el cual se podrían construir los lenguajes de agentes. Para solucionar el problema anterior se pueden aplicar las especificaciones del Lenguaje de Solicitud y de Manipulación de Conocimientos *(Knowledge Query and Manipulation Language - KQML)* del Lenguaje de Comunicación entre Agentes *(Agent Communication Language - ACL).*

KQLM es parte del Esfuerzo por Compartir Conocimiento *(Knowledge Sharing Effort - KSE)* [Patil 1992]. El KSE es un conjunto de iniciativas para proveer bases de conocimiento abiertas y re-usables. KQML proporciona un lenguaje estándar para la comunicación hacia y entre los sistemas basados en el conocimiento. Otros componentes del KSE incluyen el Formato de Intercambio de Conocimiento *(Knowledge Interchange Format - KIF)* y *Ontolingua* [Patil 1992, Genesereth 1992]. El KIF es un lenguaje estándar para la representación del conocimiento, mientras que Ontolingua es un lenguaje para especificar ontologías estándar.

KQML está basado en la Teoría del Acto-Discurso, es decir un mensaje es un ejecutor indicando lo que se espera que el receptor haga con el mensaje. Por ejemplo un mensaje sencillo *tell* indica que el receptor está obligado a creer en la veracidad de los hechos proporcionados. Un mensaje *ask* espera una respuesta a una pregunta.

KQML es un lenguaje de comunicación entre los agentes en capas [Finin *et al.* 1994, Finin *et al.* 1995, Mayfield *et al.* 1996]. Se puede considerar el lenguaje KQML dividido en dos capas: la capa del contenido y la del mensaje o la capa de la comunicación. La primera es el contenido actual del mensaje en el lenguaje de representación del agente. KQML puede llevar cualquier lenguaje de representación, incluyendo el lenguaje expresado como cadenas de ASCII y aquellas expresadas mediante una anotación binaria. Todas las implementaciones de KQML ignoran la porción de contenido del mensaje o solamente cuando tienen que determinar hasta dónde termina. El nivel de comunicación codifica un conjunto de características en el mensaje, las cuales describen los parámetros de comunicación de más bajo nivel, como la identidad del remitente y el receptor, y un identificador único asociado con la comunicación. También determinan los tipos de interacciones que se puede tener con un agente hablante de KQML. La función primaria de la capa de comunicación es identificar el protocolo que se usará para transmitir el mensaje y para suministrar un acto de discurso o ejecutor que el remitente anexa al contenido. El ejecutor significa que el contenido es una afirmación, una pregunta, una orden, o cualquiera de un conjunto de ejecutores conocidos. Por ser el contenido opaco al KQML, esta capa también incluye características opcionales que describen el contenido, por ejemplo su lenguaje. Conceptualmente un mensaje de KQML consiste en un ejecutor, los argumentos asociados que involucran el contenido real del mensaje y un conjunto de argumentos opcionales que describen el contenido de una manera que sea independiente de la sintaxis del lenguaje del contenido.

KQML proporciona una extensa lista de performativos que tienen que ver con una revisión de creencias, preguntas, mantenimiento de la base de conocimiento, acciones y servicios.

4. Arquitectura de sistema DESO: ejemplos de estructuras de comunicación

El sistema DESO, el cual se está desarrollando, busca la automatización de las siguientes etapas de re-ingeniería de FMS [Sheremetov 1993, Smirnov *et al.* 1995b, Smirnov *et al.* 1995c, Smirnov *et al.* 1995d]:

- especificación de los requisitos,
- diseño de la configuración,
- diseño de distribución del equipo,
- análisis de desempeño y simulación,
- análisis de costos y beneficios.

La arquitectura DESO se representa en la figura 5. Consta de un número de componentes que fueron desarrollados inicialmente utilizando diferentes plataformas de operación y modos de interacción. El uso de estas plataformas y herramientas saca provecho de la distribución del proceso de re-ingeniería entre diferentes usuarios, lo cual conduce a la implementación del principio fundamental para CE -su naturaleza cooperativa basada en la distribución de procedimientos de diseño entre los diferentes usuarios, considerando diferentes aspectos en la manera concurrente sobre el espacio de conocimiento compartido (diseño de múltiples aspectos). En este caso es más natural representar el conocimiento de entidades racionales conceptuales como un conjunto de agentes autónomos interactuantes que procesan bases locales del conocimiento y utilizan los principios de AOPS como la base para la integración general del sistema [Freeman 1990, Serrano 1991, Cutkosky *et al.* 1993, Smirnov 1994].

Las metodologías convencionales para la integración de las herramientas de ingeniería dependen de estructuras de datos estandarizados y de un modelo de diseño unificado, los cuales requieren compromisos substanciales de los diseñadores de herramientas. DESO se aparta de estas metodologías de dos maneras fundamentales. Primero, los datos y los modelos de herramientas son encapsulados en vez de ser estandarizados o unificados. Cada herramienta es por lo tanto libre de usar las representaciones internas más apropiadas y los modelos para su tarea. En segundo lugar, los agentes para encapsular

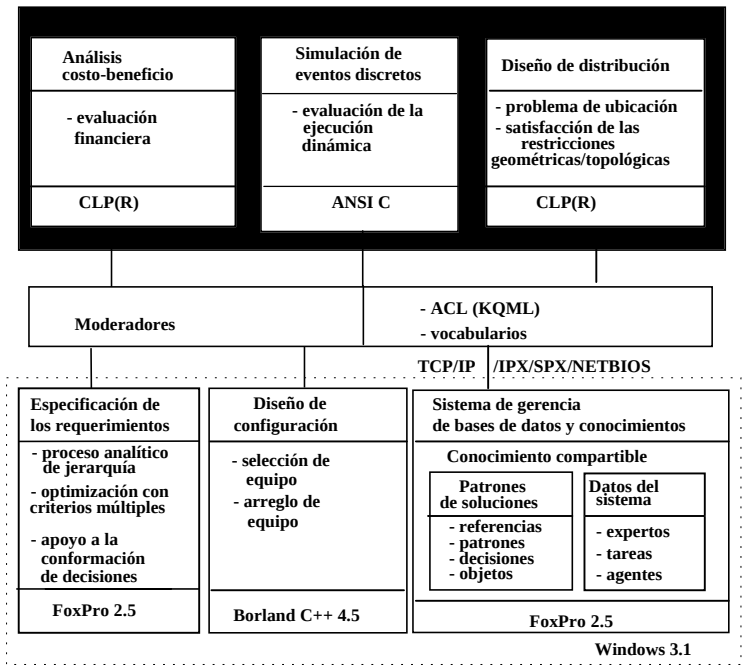


Figura 5. Estructura general del ambiente de software DESO

se comuniquen entre ellos mismos y con objetos en otros procesos. En el sistema clave consideramos a los objetos como procesos separados, y se enlazan procesos entre sí como un conjunto de procesos cooperativos paralelos con la transmisión de mensajes (figura 6). Este caso es similar al mecanismo descrito en Shapiro 1989 y Kaiser 1989.

El problema para notificar los cálculos que reflejan los cambios en datos apropiados se soluciona por la metodología de encuestas y los mensajes enviados y recibidos. La idea clave de esta metodología es checar una y otra vez si se ha cambiado un elemento de los datos que nos interesan. Se parece a la arquitectura de pizarrón, pero se realiza por medio de pruebas de mensajes en regiones específicas de la RAM, o en los archivos terminales específicos para cada objeto, dependiendo del modelo computacional. Otra posible metodología de interacción llamada *"valor activo" (active value)* que con frecuencia se apoya en los lenguajes OOP y representa la propagación del cambio a cualquier valor para todas las partes que interesan, es algo difícil de llevar a cabo en los modos concurrentes y distribuidos. Otra función muy importante para simulación, íntimamente

conectada a lo mencionado, es la sincronización de transacciones entre los objetos de simulación. En este caso se utiliza el método de reloj checador cuando a cada mensaje se le asigna un registro único del tiempo local. Después se salvan temporalmente las transacciones en archivos terminales apropiados para realizar el mecanismo de adquisición de historia del proceso.

La distribución del modelo de simulación en objetos se realiza en base a concurrencia interna del sistema objetivo. También se puede representar a los objetos, o por módulos reales de software que implementen los procesos, o por módulos que simulen sus actividades y recojan las estadísticas.

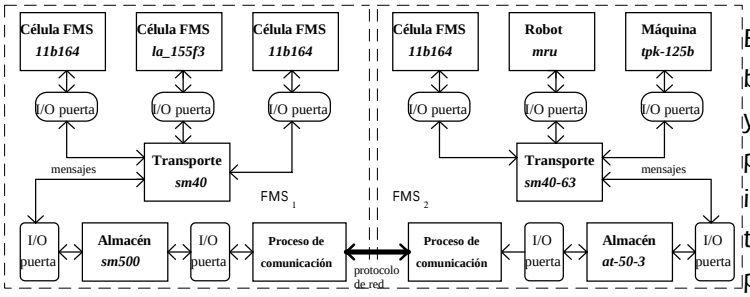
En una versión posterior el gerente de los objetos controla la operación general del sistema y la asignación de recursos. Cuando el objeto llama a otro objeto que reside en otro proceso se suspende el uso del tubo o archivo terminal. Mientras tanto, el gerente selecciona la mesa de unificación apropiada y envía un mensaje al otro proceso, indicando los parámetros, el código de identificación del objeto que llama y los parámetros temporales cuando sean necesarios:

msg (Receiver, Content, Arguments)

Por ejemplo:

msg (fms, simulate, sim, fms, 10, 8)

Los objetos de comunicación sirven como moderadores en el modo distribuido de computación (véase la sección 4.4) y son responsables de convertir el formato de un mensaje a partir de/a un formato estándar aceptado por la red. Para controlar el sistema el gerente utiliza tecnología sustituta *(proxy objects)* y contiene una ranura especial de objetos residentes que tienen las formas de los componentes del ambiente de software.



Tanto las llamadas sincrónicas como las asincrónicas son posibles. En el caso anterior el mecanismo de *rendezvous* se lleva a cabo por medio de un bloqueo por parte de la afirmación de recepción sobre el proceso que llama. En el caso posterior el proceso de llamado no espera respuesta y continúa la ejecución.

Se distribuye en la red una simulación del FMS entero para probar los resultados del diseño de distribución del equipo, así como las estrategias de control aplicadas, con cada uno de los sistemas, simulando sus respectivos componentes y comunicando sus resultados a los otros. Se puede efectuar la simulación a través de un lazo sencillo en donde cada sistema envía órdenes o datos al siguiente.
De hecho, primero se desarrolló la simulación como una función de presión para que todos los componentes del sistema se comuniquen por medio de la transmisión de mensajes, y para probar la eficiencia de sus modelos. Aunque esta parte de la demostración sirvió para iniciar los experimentos en interoperación, se necesitaban largas negociaciones entre los diseñadores del sistema para decidir sobre la secuencia de control y los formatos de datos. En este sentido se hizo la integración de sistemas para la simulación distribuida en la manera tradicional *ad hoc.*

4.2 Arquitectura del sistema de diseño de configuración: la metodología del proceso-CE

El concepto del proceso-CE o agente compuesto es la base para la metodología propuesta en Koulinitch 1995 y Koulinitch *et al.* 1996. El diseño de configuración de un proyecto se considera como una colección de procesos interactivos paralelos integrados como unidades que tienen una meta en común. Cada proceso captura una parte seleccionada del proyecto para protegerse de las posibles actualizaciones no coordinadas por otros

módulos. Una parte capturada de un proyecto se llama fragmento. Cada agente puede crear nuevas formas de datos como modelos de partes del proyecto y sus muestras, obtener datos de DB y el usuario, transformar los datos de acuerdo a fórmulas particulares, manipular el conocimiento y salvar los resultados en DB. La pantalla de interfase del usuario se presenta en la figura 7.

Cada agente puede usar todo el DB & KB, pero las operaciones de actualización pueden llevarse a cabo solamente sobre una parte capturada del proyecto. Por lo tanto todas las acciones que empiezan desde la captura de una parte de un proyecto, hasta su desarrollo, se cubren por un solo proceso llamado proceso-CE. Durante el desarrollo del proyecto se pueden engendrar los procesos "niño" destinados a lograr una meta común. Este mecanismo está basado en una noción de actor - uno de los modelos convencionales para la computación paralela, basado en el paso de mensajes asincrónico [Agha 1986, Agha 1988, Agha *et al.* 1993]. Los actores son objetos activos que procesan mensajes, crean nuevos actores o los activan por medio del paso del mensaje como resultado del procesamiento del mensaje.

Para ilustrar algunas de las dificultades en la estrategia de transmisión de mensajes particulares consideremos el problema del diseño de configuración de FMS en el nivel de producción. Para más detalles véase Smirnov *et al.* 1995 a-c. Supongamos que la tarea de modificación de fragmentos sea para organizar el equipo en el área de trabajo de la célula FMS como una función del mismo atributo de sus componentes: máquina torno (tpk-125b) y robot (mru):

working_area(la_155f3)=h(working_area(tpk-125b),(working_area(mru)))

Lo anterior se puede escribir como

$f(x,y) = h(g_1(x), g_2(y))$,

donde $g_1, g_2 : R \to R$, $h: R^2 \to R$, $f:R \to R$, son funciones de cálculo. El proceso de comunicación para este sistema de actores se ilustra en la figura 8. Se inicializa el cálculo mediante el envío de un mensaje M a su destino -actor F que es capaz de hacer arreglos y calcular la función f. El mensaje M contiene los argumentos x, y y un indicador hacia un actor C, al cual se refiere como un destino de resultados. Como resultado del procesamiento de mensajes F crea tres nuevos actores: G1, G2, H, haciendo arreglos parciales y calculando funciones g_1, g_2, h en forma simultánea. Además se envían los mensajes M1 y M2 a G1 y G2. Ambos mensajes contienen argumentos x o y y *result_to* relación para *H.* Este último enviará el resultado de la operación al

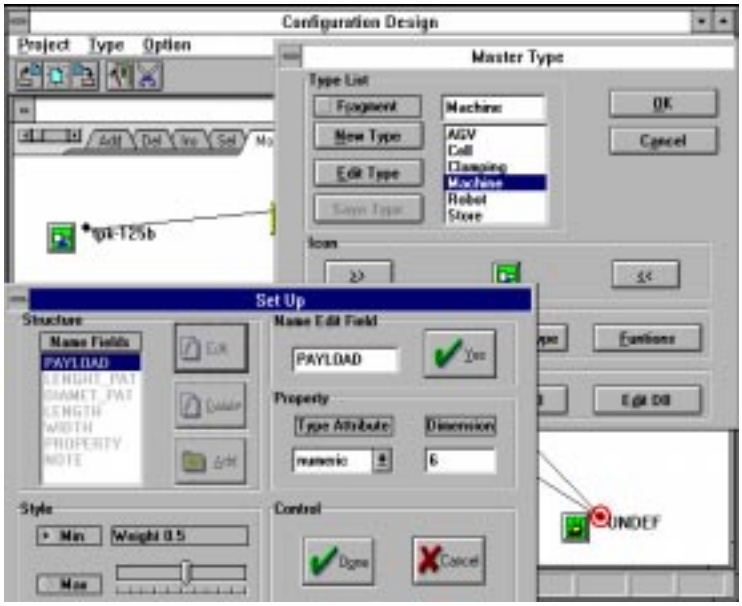


Figura 7. Interfaz del usuario del proceso-CE

actor-destino C. G1 y G2 procesan a M1 y M2 enviando mensajes M3 y M4 que contienen valores para $g_1(x)$, $g_2(x)$ simultáneamente a H. H procesa M3 y M4 y envía el resultado al actor cliente C en el mensaje M5.

De esta manera la coordinación de cálculos se apoya por medio del paso paralelo de mensajes entre los actores. Un actor puede aceptar y enviar mensajes, lo cual puede definir las restricciones de los datos o la regla de la integridad de las restricciones, transferir un mensaje transparente a otros fragmentos y crear un nuevo fragmento. Pero para solucionar los problemas de la consistencia e integridad de datos por medio de los métodos formales de la aplicación de la reescritura de grafos, es necesario realizar un modo centralizado de

cálculo. Se llevan a cabo las funciones de la coordinación del proceso para cada proyecto por medio del módulo especial de comunicación llamado proceso-CM (o moderador -para más detalles ver en las secciones posteriores-) (figura 9). Este proceso se inicializa por el primer proceso-CE que captura un fragmento de este proyecto para actualizarlo. Se cierra el proceso-CM después de que se haya terminado el último proyecto capturado del proceso-CE y se haya hecho la actualización resultante.

La teoría de la reescritura de grafos sensibles a contexto es una herramienta básica para la solución del problema de coordinación en la actualización de DB & KB. En la coordinación de los procesos-CE se utiliza otra idea de un sistema de actores -la de actividad de datos según la cual un estado fragmentario define las posibles operaciones de fragmentos. En el caso de que DB & KB compartidas sean utilizadas por los agentes, los moderadores responden a dudas y otros comandos que operan sobre las estructuras del conocimiento y las traduce a un sistema meta apropiado. El moderador usa el esquema de DB, construyendo una representación del esquema para el gerente de DB basado en esta información. Subsecuentemente, en respuesta a una solicitud de actualización por parte de una aplicación que requiera acceso al DB, el moderador analiza gramaticalmente la solicitud y genera afirmaciones de DBMS en el lenguaje apropiado para la manipulación de datos; por ejemplo, las solicitudes en SQL y los comandos de manipulación de datos. En el caso de una solicitud de inmediato procesa las tuplas que se le hayan regresado por el DB en la forma en que se pidió en la solicitud.

El proceso-CM es el coordinador del proyecto, el cual está basado en el modelo servidor-cliente. Éste coordina todo el intercambio de datos en el proyecto conectado, controla la integridad de DB & KB y la consistencia del proyecto (checa la consistencia más débil), concurrentemente modifica las partes del proyecto. El proceso-CM captura y monitorea un flujo de datos concurrentes para actualizar el estado del proyecto y

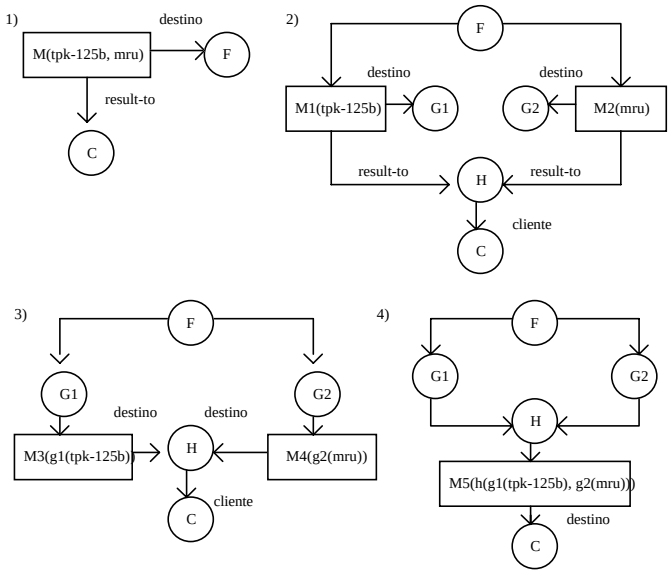


Figura 8. La resolución de problemas y las estrategias de comunicación utilizando actores-CE

proveer diseñadores con importantes mecanismos de coordinación durante las diferentes fases de CE. Si un fragmento es una parte de diferentes proyectos, entonces su actualización se controla y se checa por todos los procesos-CM de estos proyectos.

4.3 Agentes homogéneos: solución basada en CLP(R)

En esta sección vamos a describir la estructura homogénea de agentes múltiples propuesta para resolver el análisis de costos-ganancias y los problemas

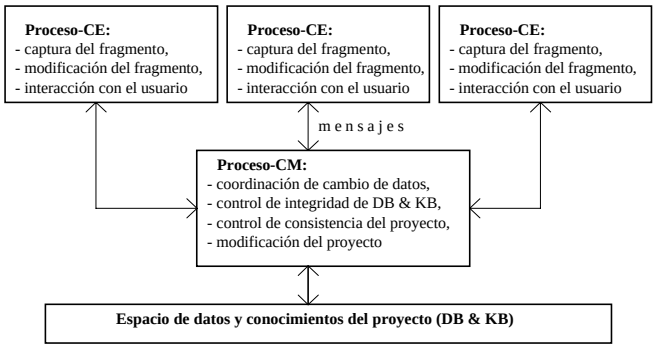


Figura 9. Estructura del sistema de procesos-CE

del diseño de la distribución, utilizando el modelo de la representación del conocimiento basado en

restricciones y la fuerza expresiva en programación lógica con restricciones *(Constraint Logic Programming - CLP)* para la programación de agentes. El diseño de distribución es una de las tareas de re-ingeniería, corresponde al problema de ubicación y tiene que cubrir las restricciones geométricas y topológicas. Para implementar el diseño de la distribución, recurrimos al poder del lenguaje CLP(R) (programación lógica con restricciones sobre estructuras R) porque es el indicado para el problema a resolver, debido a su poderoso mecanismo de satisfacción de restricciones [Jaffar 1992, Smirnov *et al.* 1995a]. Con la aplicación de la definición de AOPS propuesta en (1) para este caso particular, enfocado al modelo de satisfacción de restricciones de la representación del conocimiento, podemos definir a un AOPS basado en CLP(R) de la siguiente manera:

K'=A'=0 -fórmula atómica arbitraria, regla de CLP(R) que se le puede agregar a la LKB del agente si es necesario,

M' -funciones de CLP(R),

I' -métodos de resolución y satisfacción de restricciones,

L' -CLP(R) y KQML,

S' -es un subconjunto de L' (KQML),

F' -búsqueda primero en profundidad y métodos para la propagación de restricciones,

C' -mecanismos de bajo nivel para la transferencia de mensajes,

G' - base de conocimiento con apoyo directivo, incluyendo operadores *"assert"* y *"retract"*.

Los vocabularios estándar de agentes se encuentran en el fondo de la jerarquía funcional de AOPS heredada por muestras de agentes. El núcleo del lenguaje de programación del agente se lleva a cabo como un conjunto de vocabularios de tres tipos diferentes. Se puede realizar a (K') como un conjunto de átomos y de reglas en la semántica de CLP(R) utilizado para representar el conocimiento en LKB. El vocabulario (M') de métodos es un conjunto de funciones (programas en CLP) utilizados para realizar el repertorio de comportamiento de agentes (o clases). El vocabulario de atributo (A') es un conjunto de átomos y reglas en semántica de CLP(R) utilizado para accesar los atributos y las características de agentes.

Consideremos una estructura interna del agente representada en la figura 10. Cada agente es un proceso contenido en sí mismo y consta de un programa de un solo agente. La capa de contenido de la interacción entre los agentes se define en gran medida por la funcionalidad de los agentes. Cada agente hereda LKB vacío de un agente estándar. Su vocabulario de LKB está orientado a problemas y contiene hechos, reglas y afirmaciones conocidos que son necesarios para describir el estado de conocimiento del agente en cada situación especial de toma de decisiones. También contiene conocimiento acerca del comportamiento, atributos, métodos de heredar de otros agentes que se utilizan para realizar la actividad de planeación y explicación de un agente.

El vocabulario de los métodos representa un conjunto de operaciones básicas (atómicas) definidas para un agente patrón y está compuesto de 5 grupos básicos de métodos: solicitudes a LKB, administración de LKB, métodos de herencia, creación/destrucción de agentes, transformación de solicitudes (transformaciones del conocimiento).
Para aprovechar la ventaja de KB dinámica en un ambiente multiagente, cada agente tiene una posibilidad para incluir y excluir el conocimiento de un LKB si es necesario; se utiliza el siguiente conjunto de métodos en notación de tipo Prolog para apoyar esta acción:

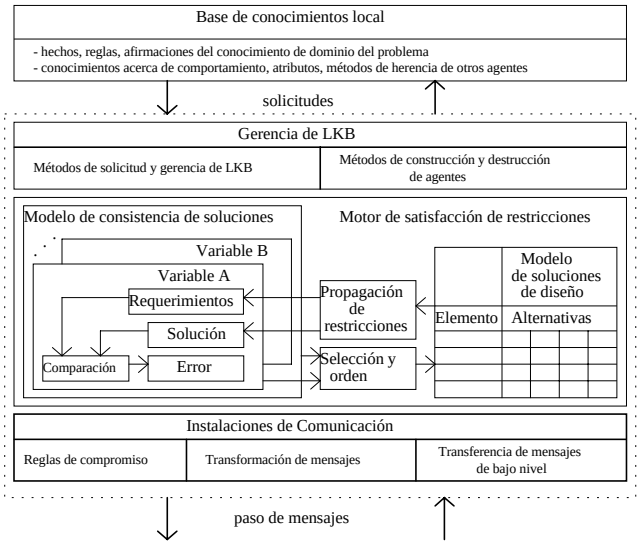
add_fact(TermOrList)
add_rule(RuleOrRules)
retract_kb(Term)
retract_all_kb(Term)

clear_kb.

Los mecanismos para heredar son apoyados de la siguiente manera de acuerdo a la información acerca del estado interno del agente y los estados de otros agentes:

get_parent(Parent)
get_parents(ParentList)
get_ancestors(AllAncestors)
get_children(SuccessorsList)

Como ya hemos mencionado, los agentes son capaces de modificar su comportamiento por su cuenta. Esto significa que es necesario atender el uso de la información de la mayoría de estos mensajes debido a que a través de un conjunto de mutaciones adaptativas un agente y su origen tal vez no tengan mucho en común. La producción de copias de agentes se realiza por medio de:



make_clone(Name)
make_mixed_copy(Name, Agents, FactList, RuleList, MethodList, AttributeList)
delete_self.

Mientras utilizamos el primer método se crea una copia de un origen con LKB vacío.

En concepto de AOPS los agentes pueden obtener conocimiento mediante la aplicación de diferentes formas: por medio de pruebas al ambiente para checar sus variables descriptivas (clave) para los valores críticos, la aplicación de la habilidad de aprendizaje por la explotación de KB interna y externa mediante el razonamiento deductivo, solicitar conocimientos de otros agentes o participar en un proceso de negociación. Para apoyar estas últimas posibilidades aplicamos los siguientes métodos:

in_kb(Term)
get_facts(FactList)
get_rules(RuleList)
get_kb(FactList, RuleList, Format)
transfer_knowledge_to (AgentOrAgents[, Filter, Format])

Se aplica el método *transfer_knowledge_to* en el caso de ambiente heterogéneo para transformar el conocimiento de una forma de representación a otra, o cuando se aplican diferentes protocolos de comunicación. Para realizar una adaptación como una de las demandas generales al sistema de agentes, cada agente particular debe tener las posibilidades de cambiar el comportamiento de aquellos agentes que se encuentren bajo su responsabilidad. Para realizarlo se propone el siguiente conjunto de métodos que dan como resultado el cambio de sus vocabularios de métodos:

can_handle_message(Message)
defined_messages(MessageList)
get_all_messages(FullMessageList)
assert_method(MethodHeader,(Method:-MethodBody))
retract_method(MethodBody)
optimize_self(pointers)
optimize_self(execute)

Mientras aplicamos el primer método de optimización se utiliza la división de la estructura, aplicando la técnica de búsqueda en el grafo de herencia, la selección de

todos los apuntadores a los antecesores, capaz de ejecutar el modelo a la mano, el cual provee una flexibilidad de herencia funcional. El segundo método está basado en copiar la estructura de los métodos originales directamente al cuerpo de sucesores. Al aplicar esto un agente cambia su lugar en la jerarquía heredada, proveyendo independencia del agente de los cambios de la funcionalidad de los padres o las relaciones de herencia.

El siguiente paso en este trabajo es conectar este ambiente basado en CLP(R) a otras aplicaciones, realizados en otras semánticas que no sean CLP(R) dentro de la red. La manera de hacerlo utilizando KQML se describe en la siguiente sección.

4.4 Interacción heterogénea de agentes

La implementación de la arquitectura basada en IAA en el nivel superior del sistema provee una estructura sencilla para la comunicación entre los subsistemas DESO. Varios sistemas comercialmente disponibles apoyan la transmisión distribuida de mensajes a través de aplicaciones heterogéneas. DESO provee la estructura para especificar el contenido compartido y posibilita a las herramientas para interoperar sin comprometer determinados formatos de datos. También hace posible la rutina y notificación basada en contenido de los mensajes y no solamente en los patrones sencillos de sintaxis.

La heterogeneidad de los agentes puede ser clasificada de la siguiente manera:

- Heterogeneidad sintáctica -es real debido a los diferentes esquemas de representación del conocimiento y la implantación de formalismos.
- Heterogeneidad semántica, que surge por los diferentes significados del mismo conocimiento para los agentes.
- Heterogeneidad de control a causa de las diferentes estrategias de inferencia, aplicadas por los agentes.

Los agentes se comunican entre sí usando el ACL. El operador que envía el mensaje facilita la comunicación entre los agentes intercambiando mensajes en una forma expresiva de KQML y en el formato de KIF. Como está definido bajo una metodología declarativa en versión de prefijo de mensajes con predicado de primer orden, se apoya la comunicación entre agentes por vocabulario exterior a KQML. La mayoría de los mensajes KQML consisten en preguntas y afirmaciones en KIF (al igual que las funciones básicas de control como *reiniciar*). Los agentes usan también expresiones en KIF para informar a cada uno de sus intereses y para solicitar la notificación de cambios de información informales que pudieran afectarlos. Debido a que las solicitudes y los intereses se expresan en un lenguaje declarativo formal, el significado de los términos no depende de programas particulares y puede compartirse entre los programas con diferentes implementaciones y reservas de conocimiento. Por lo tanto se describe la comunicación en el nivel alto, mientras que se esconden los detalles de la transmisión iterativa de mensajes asociada con las plataformas de operación y los procedimientos de resolución de problemas.

En seguida se presenta una lista de ejecuciones estándar con una breve descripción. Los mensajes informan a un agente sobre algún hecho o hechos relacionados con el mundo. Aquí se da la definición rápida de cada mensaje estándar:

Tell: p es verdadero.
Untell: p tal vez no sea verdadero.
Assign: el valor de x es y.
Unassign: No se sabe el valor de x.
Eliminate: Se elimina todo lo que se cree sobre x.
Observe: Se tomó en cuenta la condición x.
Event: Se observó evento x.

A continuación se ofrece una definición rápida de cada solicitud estándar:

Ask-if: Pregunta si p es verdadero.
Ask-one: Encuentra un x de tal manera que p sea verdadero.

Ask-all: Encuentra todos los x para que p verdadero.
Ask-value: Da el valor de x.
Ask-about: Da todas las oraciones sobre x.
Ask-words: Proporciona todas las palabras sobre las cuales tienes creencias.
Achieve: Actúa para producir un estado en condición x.
Help: Proporciona una cadena de ayuda para x.
Perform: Ejecuta el programa x.

Para apoyar el contenido de los mensajes descritos entre diferentes subsistemas presentamos el uso de moderadores *(facilitators)* que pueden utilizar el contenido de los mensajes para canalizarlos en forma selectiva a los agentes que tengan intereses o conocimientos relevantes. La naturaleza heterogénea de los componentes del sistema realizados en diferentes ambientes y sobre diferentes plataformas hace que el empleo de los moderadores traduzca los mensajes entre los agentes (véase figura 5), lo cual mejora la comunicación entre ellos. Un tipo de programa que tendría mucho éxito en este tipo de ambiente es un mediador (o agente de información) [Wiederhold 1992, Cutkosky *et al.* 1993]. Los mode-radores son procesos (agentes) que se sitúan entre los procesos que proveen y los procesos que consumen y realizan servicios sobre la información no-procesada, como proporcionar interfaces estandarizados, la integración de información a partir de varias fuentes y la traducción de solicitudes de información o respuestas. Los moderadores se están volviendo cada vez más importantes debido a que se les propone como un método efectivo para la integración de nuevos sistemas de información con sistemas inflexibles de herencia. Los agentes interactúan a través de moderadores que traducen el conocimiento específico para una herramienta dentro y fuera de un lenguaje estándar de intercambio de conocimientos. Por lo tanto cada agente puede razonar en sus propios términos, solicitando información a otros agentes y proveyendo a otros agentes de información conforme se vaya necesitando a través de los moderadores.

Cada moderador tiene la responsabilidad de proveer una interfaz entre una colección local de agentes y los agentes remotos, cumpliendo con cuatro propósitos:

1. Proveyendo de una capa confiable de transmisión de mensajes,
2. Canalizando los mensajes de salida para su(s) destino(s) apropiado(s),
3. Traduciendo los mensajes que ingresan para el consumo de su agente y
4. Iniciando y monitoreando la ejecución de sus agentes.

Por lo tanto, la comunicación y coordinación se da entre agentes, agentes y moderadores y entre moderadores, dependiendo en realidad a qué subsistemas pertenezcan los agentes comunicativos. A esta organización de agentes y moderadores se le llama arquitectura federativa. Los mensajes de agentes a moderadores son indirectos; es decir, tienen contenido pero no direcciones. Es responsabilidad de los moderadores canalizar estos mensajes hacia agentes capaces de manejarlos. Al ejecutar esta tarea, los moderadores pueden ir más allá de patrones sencillos de combinación -pueden traducir mensajes, descomponer problemas en subproblemas y programar el trabajo en estos subproblemas. En algunos casos lo anterior puede realizarse interpretativamente (con mensajes que pasan por el moderador); en otros casos, puede hacerse de un solo golpe (con el moderador estableciendo enlaces especializados entre agentes individuales y luego salirse).

La sintaxis del mensaje es la de un ejecutante *(per-formative)* seguida por una lista desordenada de pares: palabra-clave - valor. Por ejemplo, un mensaje que representa una pregunta acerca del tamaño del área de trabajo de una herramienta especial para una máquina podría codificarse como:

```
(ask-one :receiver MachineDBManager
   :sender la_155f3
   :content working_area(Machine, X)
   :language prolog
   :reply-with tpk-125b)
```

causaría la respuesta:

(reply :receiver la_155f3
:sender MachineDBManager
:content working_area(Machine, 5.25)
:language prolog
:in-reply-to tpk-125b)

En este mensaje, el ejecutante de KQML es *ask-one* (pedir-uno), el contenido es *working_area(Machine, X)* y la ontología asumida es identificada por la señal *MachineDBManager.* La misma solicitud general podría comunicarse utilizando Prolog estándar como lenguaje de contenido en una forma que solicite el conjunto de todas las respuestas como:

(ask-all :content «working_area(Machine, X)»
:language standard_prolog
:ontology DESO).

A las dos operaciones se les requirió permitir a los mensajes de KQML ser transferidos hacia el nivel intérprete. La primera fue analizar la cadena de ASCII representando en señales el mensaje KQML. La segunda fue convertir el formato de un mensaje KQML de una definición función LISP al formato apropiado del agente. Para simplificar en esta etapa, se decidió restringir las expresiones KQML a ser átomos CLP(R) válidos. El predicado resultante

transfer_knowledge_to(AgentOr-Agents, Filter, KQML),

convierte las hileras KQML en Prolog y viceversa.

Por lo tanto el mensaje CLP(R) es un mensaje de la forma ejecutante, palabra clave (valor). Por ejemplo:

[tell: sender(la_155f3),
receiver(MachineDBManager),
content(working_area(Machine, X)),
language(prolog)].

Cada mensaje recibido es examinado contra toda regla de compromiso, para permitir que ocurran acciones múltiples.

Para apoyar la comunicación interagente de bajo nivel se usa API de comunicación. Puede ser vista como una biblioteca de rutinas de C que permite al usuario enviar hileras de mensajes KQML vía protocolos TCP/IP, HTTP, IPX/SPX/NETBIOS. Éste analiza el nombre del campo *receptor* y entrega el mensaje KQML al puerto especificado por el Localizador Universal de Recurso *(Universal Resource Locator - URL)* asociado con el nombre del agente.

La comunicación entre cualquiera de los dos agentes procede al utilizar los métodos *ApiInterface::message* y *ApiInterface::Request.* De esta manera, ya sea que un agente esté en el mismo espacio de dirección o en otra máquina, es completamente transparente para el agente que esté enviando un mensaje o solicitud. Esta abstracción se vuelve útil en API de Windows y UNIX, donde una API puede residir en cada máquina. De esta forma, la comunicación procede de la misma manera estén o no los agentes en la misma máquina o en diferentes máquinas y estén escritos en diferentes lenguajes.

5. Conclusión

Con base en la experiencia obtenida a través del desarrollo de DESO, se puede concluir que el concepto de los agentes que se comunican en un nivel de conocimiento es la forma correcta para componer sistemas grandes y complejos a base de módulos existentes de software. En vez de integrar literalmente un código, se puede encapsular a los módulos por agentes y luego invocarse a distancia como servicios de red cuando se necesiten. Este tipo de metodología tienen una clara ventaja en situaciones en las cuales la instalación localmente del software requiera una reconfiguración costosa del sistema o donde se requieran expertos para correr aplicaciones y darle mantenimiento al código. Muchos paquetes de software en ingeniería tienen estas características, desalentando su explotación por los usuarios ocasionales y

organizaciones pequeñas.

A través de DESO estamos explorando una metodología para la solución cooperativa de problemas de ingeniería basada en conocimiento compartido. Las herramientas y estructuras de ingeniería son encapsuladas por agentes que intercambian información y servicios a través de un modelo compartido explícito del diseño. Conceptualmente, este modelo compartido es centralizado, pero en la práctica, es distribuido entre los modelos internos especializados mantenido por cada herramienta o estructura.

Para crear la ilusión de un modelo de diseño compartido, la mayoría de las interacciones entre los agentes son mediadas por moderadores. Cada moderador del agente es responsable de:

1. Localizar otros agentes en la red capaces de proveer la información y servicios solicitados,
2. Establecer una conexión a través de ambientes de computación (potencialmente) heterogéneos y
3. Manejar la conversación resultante.

La información es pasada entre los agentes y moderadores en KIF, basado en la lógica de primer orden, con agentes, traduciendo entre el interlingüa y las representaciones internas de sus clientes. El compartir los conocimientos a través de las disciplinas, es posible por los acuerdos ontológicos *a priori* entre los agentes acerca de los significados de los términos. Los agentes y los moderadores coordinan sus actividades usando un lenguaje de comunicación y control (actualmente KQML).

La arquitectura basada en KQML es más práctica que los agentes homogéneos en CLP(R), al poder ahora comunicarse a través de una red con otros agentes basados en KQML. Mientras se utilice KQML significa que nuestros agentes pueden comunicarse con otros agentes, esto no significa que puedan entender el campo del contenido sin el uso de moderadores. El problema puede solucionarse utilizando KIF como un lenguaje intermedio o como un lenguaje interno del agente.

KQML parece estar bien especificado. Sin embargo las diferencias entre ejecutores "informativos" (como *tell*) y los ejecutores de "base de datos" (como *insert)* causan distintas interpretaciones, como lo hacen las semánticas exactas de *achieve* y *unachieve*. Una segunda cuestión es que mientras en el lenguaje exista una palabra-clave para el campo de contenido, debe darse una palabra-clave para los campos restantes, como un *sender*, etcétera. Esto simplificaría los sistemas basados en non-LISP y non-Prolog, empleando KQML para utilizar estos campos. La propuesta final es que los mensajes KQML lleven un campo de tiempo, el cual es muy apreciado especialmente por el software de simulación.

6. Reconocimientos

El trabajo descrito lo apoyó inicialmente el Ministerio de Ciencia y Política Técnica de Rusia, bajo la concesión N 83-33. Los experimentos no habrían tenido éxito sin la contribución del equipo de investigadores y programadores del Laboratorio de Sistemas Automáticos Integrados del Instituto de Informática y Automatización de San Petersburgo, adscrito a la Academia Rusa de Ciencias. Asimismo, agradecemos la colaboración de los profesores Patrick Rafferty y María Esther León García, de la Universidad Tecnológica de la Mixteca en la traducción del artículo al español.

[Agha 1986] Agha, Gul A. *ACTORS: A Model of Concurrent Computation in Distributed Systems*. Series in Artificial Intelligence. The MIT Press, Cambridge, Massachusetts, 1986.

[Agha 1988] Agha, G. A. and Hewitt, C. E. "Concurrent Programming Using Actors." In Y. Yonezawa and M. Tokoro, editors, *Object-Oriented Concurrent Programming.* The MIT Press, MA, 1988.

[Agha *et al.* 1993] Agha, G., Wegner, P. and Yonezawa, A., editors (1993). *Research Directions in Concurrent Object-Oriented Programming.* The MIT Press, Cambridge, MA.

[Bates 1994] Bates, J. (1994). "The Role of Emotion in Believable Agents." *Communications of the ACM,*

Referencias

Reilly, W. (1992a). *An Architecture for Action, Emotion, and Social Behavior.* Technical Report CMU-CS-92-144, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA.

[Bond and Gasser 1988] Bond, A. H. and Gasser, L., editors (1988). *Readings in Distributed Artificial Intelligence.* Morgan Kaufmann Publishers, San Mateo, CA.

[Carver 1992] Carver, Norman and Lesser, Victor. "Blackboard Systems for Knowledge-Based Signal Understanding." In *Symbolic and Knowledge-Based Signal Processing,* Alan Oppenheim and Hamid Nawab, editors, Prentice Hall, 205-250, 1992.

[Chess 1994] Chess, D. M., Harrison, C. G. and Kershenbaum, A. *Mobile Agents: Are they a Good Idea?* IBM Research Report, RC 19887, 1994.

[Cutkosky *et al.* 1993] Cutkosky, M. R., Engelmore, R. S., Fikes, R. E., Genesereth, M. R., Gruber, T., Mark, W. S., Tenenbaum, J. M., and Weber, J. C. (1993). "PACT: An Experiment in Integrating Concurrent Engineering Systems." *IEEE Computer,* 26(1):28-37.

[Davies 1995] Davies, W. and Edwards, P. "Agent-Based Knowledge-Discovery." *Proc. of the AAAI-95 Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments.* 1995.

[Davis 1983] Davis, R. and Smith, R. G. "Negotiation as a Metaphor for Distributed Problem Solving." *Artificial Intelligence,* 20:63-109, 1983.

[Engelmore 1988] Engelmore, Robert and Morgan, Tony (editors). *Blackboard Systems,* Addison-Wesley, 1988.

[Erman 1980] Erman, Lee; Hayes-Roth, Frederick; Lesser, Victor and Reddy, D. Raj. "The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty." *Computing Surveys,* vol. 12, no. 2, 213-253, 1980 (also in *Blackboard Systems,* Robert Engelmore and Tony Morgan, editors, Addison-Wesley, 31-86, 1988).

[Etzioni *et al.* 1994] Etzioni, O., Lesh, N., and Segal, R. (1994). "Building Softbots for UNIX." In Etzioni, O., editor, *Software Agents - Papers from the 1994 Spring Symposium* (Technical Report SS-94-03), pages 9-16. AAAI Press.

[Fehling 1980] Fehling, M. (Ed.). "Report on Third Annual Workshop on Distributed Artificial Intelligence." *Sigart Newsletter,* 84:3-12, 1980.

[Finin 1994] Finin, Tim; Fritzson, Richard; McKay, Don, and McEntire, Robin (1994). "KQML as an Agent Communication Language." In *Proceedings of the Third International Conference on Information and Knowledge Management,* ACM Press.

[Finin 1995] Finin, Tim; Labrou, Yannis and Mayfield, James (1995). "KQML as an Agent Communication Language." In Jeff Bradshaw (Ed.) *Software Agents,* MIT Press, Forthcoming.

[Freeman 1990] Freeman, E. H. "An Agent-Oriented Programming Architecture for Multi-Agent Constraint Satisfaction Problems." *Proc. of the 2nd International IEEE Conference on Tools for Artificial Intelligence,* 1990, pp. 830-840.

[Galliers 1988a] Galliers, J. R. (1988a). "A Strategic Framework for Multi-agent Cooperative Dialogue." In *Proceedings of the Eighth European Conference on Artificial Intelligence (ECAI-88),* Munich, Federal Republic of Germany, pp. 415-420.

[Galliers 1988b] Galliers, J. R. (1988b). *A Theoretical Framework for Computer Models of Cooperative Dialogue, Acknowledging Multi-Agent Conflict.* PhD thesis, Open University, UK.

[Genesereth 1992] Genesereth, M. R., Fikes, R. E. *et al. Knowledge Interchange Format, Version 3.0 Reference Manual,* Technical Report Logic-92-1, Computer Science

Department, Stanford University, 1992.

[Genesereth and Ketchpel 1994] Genesereth, M. R. and Ketchpel, S. P. (1994). "Software Agents." *Communications of the ACM,* 37(7):48-53.

[Gruber 1992] Gruber, T. R., Tenenbaum, J. M. and Weber, J. C. "Toward a Knowledge Medium for Collaborative Product Development, AI." In *Design '92,* J. Gero, Ed., Kluwer Academic Publishers, pp. 413-432, 1992.

[Hewitt 1977] Hewitt, C. (1977). "Viewing control Structures as Patterns of Passing Messages." *Artificial Intelligence,* 8(3):323-364.

[Jaffar, 1992] Jaffar, J., Michaylov, S., Stuckey, P. J., Yap, R. H. C. "The CLP(R) Language and System." *ACM Transactions on Programming Languages and Systems,* 1992. V. 14, N 3, pp. 339-395.

[Kaiser 1989] Kaiser, G. E., Popovich, S. S., Hseush, W., and Wu, S. F. "MELDing Multiple Granularities of Parallelism." In *3rd European Conference on Object-Oriented Programming.* Nottingham, U. K., July 1989, Cambridge Un. Press, pp.147-166.

[Klein 1991] Klein, M. "Supporting Conflict Resolution in Cooperative Design Systems." *IEEE Transactions on System, Man, Cybernetics,* 21(5):1379-1390, 1991.

[Koulinitch 1995] Koulinitch, A. S. "Investigación y Desarrollo en Entornos de Información para la Ingeniería Concurrente." *Inter. Conf. CIECE Sonora 95,* March -95, pp. 192-196.

[Koulinitch *et al.* 1996] Koulinitch, A. S., Sheremetov, L. B., Smirnov, A. V., Turbin, P. A. "DB & KB Integration in Concurrent Engineering: Mathematical Model and Tools." Third ISPE International Conference on Concurrent Engineering (ISPE/CE96), Toronto, Canada, 1996.

[Lesser 1975] Lesser, Victor; Fennell, Richard; Erman, Lee, and Reddy, D. Raj. "Organization of the Hearsay-II Speech Understanding System." *IEEE Transactions on Acoustics, Speech, and Signal Processing,* ASSP-23, 11-24, 1975.

[Lesser 1987] Lesser, V. R., Durfee, E. H. and Corkill, D. D. "Coherent Cooperation Among Communicating Problem Solvers." *IEEE Transactions on Computers,* 36(11):1275-1291, 1987.

[Maes 1990a] Maes, P., editor (1990a). *Designing Autonomous Agents.* The MIT Press, Cambridge, MA.

[Maes 1994a] Maes, P. (1994a). "Agents that Reduce Work and Information Overload." *Communications of the ACM,* 37(7):31-40.

[Mayfield *et al.* 1996] Mayfield, J., Labrou, Y., Finin, T. "Evaluation of KQML as an Agent Communication Language." *Intelligent Agents,* Volume II - Proc. of the 1995 Workshop on Agent Theories, Architectures, and Languages. M. Wooldridge, J. P. Muller and M. Tambe (eds.). *Lecture Notes in Artificial Intelligence,* Springer-Verlag, 1996.

[Neches 1991] Neches, R., Fikes, R. E., Finin, T., Gruber, T., Patil, R., Senator, T. and Swartout, W. R. "Enabling Technology for Knowledge Sharing." *AI Magazine* 12(3), 16-36, 1991.

[Nii 1986a] Nii, H. Penny. "The Blackboard Model of Problem Solving." In *AI Magazine,* vol. 7, no. 2, 38-53, 1986.

[Nii 1986b] Nii, H. Penny. "Blackboard Systems Part Two: Blackboard Application Systems." In *AI Magazine,* vol. 7, no. 3, 82-106, 1986.

[OMG 1991] *The Common Object Request Broker: Architecture and Specification,* OMG Document # 91.12.1 Object Management Group, Framingham, MA., December 1991.

[Parunak 1987] Parunak, H. V. D. "Manufacturing Experience with the Contract-net." In M. N. Huhns, editor, *Distributed Artificial Intelligence,* pp. 285-310. Morgan Kaufmann Publishers, Los Altos, CA, 1987.

[Patil 1992] Patil, R. S., Fikes, R. E., Patel-Schneider, P. F., MacKay, D., Finin, T., Gruber, T. & Neches, R. "The DARPA Knowledge Sharing Effort: Progress Report." In *Proceedings of KR'92 - The Annual International Conference on Knowledge Representation,* Cambridge, MA, 1992.

[Pavlin 1988] Pavlin, J., Lesser, V. R. and Durfee, E. "Approximate processing in Real-time Problem Solving." *AI Magazine,* 9(1):49-61, 1988.

[Rosenschein and Genesereth 1985] Rosenschein, J. S.

and Genesereth, M. R. (1985). "Deals Among Rational Agents." In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (IJCAI-85), pp. 91-99, Los Angeles, CA.

[Roth 1991] Roth, S., Sadeh, N., Sycara, S. and Fox, M. "Distributed Constraint Heuristic Search." *IEEE Transactions on System, Man, Cybernetics,* 21(5):1446-1461, 1991.

[Serrano 1991] Serrano, D. "Constraint-Based Concurrent Design." *Systems Automation: Research & Applications,* 1991. N. 3. V. 1, pp. 217-230.

[Shapiro 1989] Shapiro, M., Gauton, P. and Mosseri, L. "Persistence and Migration for C++ Objects." In *3rd European Conference on Object-Oriented Programming.* Nottingham, U.K., July 1989, Cambridge Un. Press, pp.191-204.

[Sheremetov 1993] Sheremetov, L. B. "Programming Model Notion and Software Tools for Distributed Object-Oriented Simulation." *CAD/CAM, Robotics and Factories of the Feature.* St. Petersburg, 1993.

[Shoham 1990] Shoham, Y. *Agent-Oriented Programming,* Technical Report No. STAN-CS-90-1335, Computer Science Department, Stanford University, 1990.

[Shoham 1992] Shoham, Yoav. "Agent Oriented Programming: An Overview and Summary of Recent Research." In *Proceedings of the Workshop on Distributed Artificial Intelligence,* 1992.

[Shoham 1993] Shoham, Yoav. "Agent-Oriented Programming." *Artificial Intelligence,* 60(1):51-93.

[Singh 1991] Singh, M. P. (1991). "Towards a Formal Theory of Communication for Multi-Agent Systems." In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence* (IJCAI-91), pp. 69-74, Sydney, Australia.

[Smirnov 1994] Smirnov, A. V. "Conceptual Design for Manufacture in Concurrent Engineering." *Concurrent Engineering: Research and Applications: Conference Proceedings,* Aug. 29-31, 1994, Pittsburgh, Pennsylvania, pp. 461-466.

[Smirnov *et al.* 1995a] Smirnov, A. V., Sheremetov, L. B., Romanov, G. V., Turbin, P. A. "Multi-Paradigm Approach to Cooperative Decision Making." *Proc. of the II International Conference on Concurrent Engineering: Research and Applications.* Washington, DC. August 23-25, 1995. Concurrent Technology Corporation, 1995, pp. 215-222.

[Smirnov *et al.* 1995b] Smirnov, A. V., Sheremetov, L. B., Turbin, P. A. "Constraint-Based Expert System for the Design of Structured Objects." *Proc. of the International AMSE Conference on System Analysis, Control & Design, Methodologies & Examples SYS'95.* Brno, Czech Republic, July 3-5, 1995, V. 2, pp. 64-71.

[Smirnov *et al.* 1995c] Smirnov, A. V., Koulinitch, A. S., Sheremetov, L. B., Romanov, G. V. and Turbin, P. A. "DESO: A Constraint-Based Environment Prototype for Cooperative Design of FMS." *Proc. of the III IASTED International Conference.* Cancun, Mexico, June 14-16. IASTED/ACTA Press. Anaheim-Calgary-Zurich, 1995, pp. 384-387.

[Smirnov *et al.* 1995d] Smirnov, A. V., Rakhmanova, I. O., Sheremetov, L. B., Shpakov, V. M., Turbin, P. A. "GDSS for Dynamic Application Domains." *Proc. of the 4th St. Petersburg International Conf. Regional Informatics*, St. Petersburg, UNESCO, 1995.

[Smith 1980] Smith, R. G. (1980). *A Framework for Distributed Problem Solving.* UMI Research Press.

[Smith 1985] Smith, R. G. (ed.). "Report on the 1984 workshop on Distributed AI." *AI Magazine,* pp. 234-243, 1985.

[Thomas 1993] Thomas, S. R. *PLACA, an Agent Oriented Programming Language,* Ph. D. Dissertation, Computer Science Department, Stanford University, 1993.

[White 1994] White, J. E. (1994). *Telescript Technology: The Foundation for the Electronic Marketplace.* White Paper, General Magic, Inc., 2465 Latham Street, Mountain View, CA 94040.

[Wiederhold 1992] Wiederhold, Gio (1992). "Mediators in the Architecture of Future Information Systems." *IEEE*

Computer, 5(3):38-49.

[Wooldridge and Jennings 1994] Wooldridge, M. and Jennings, N. R. (1994). "Formalizing the Cooperative Problem Solving Process." In *Proceedings of the Thirteenth International Workshop on Distributed Artificial Intelligence* (IWDAI-94), pp. 403-417, Lake Quinalt, WA.