Arquitectura basada en objetos de computación distribuida en la configuración de sistemas distribuidos

Anatoli Semenovich Koulinitch*
Francisco Espinosa Maceda**
Carlos Benavides Martínez***

Resumen

Estamos en la etapa inicial de la computación distribuida. La idea básica de nuestro enfoque es el desarrollo de la infraestructura de aplicaciones de tal manera que el usuario no necesite saber nada sobre computadoras, redes, sistemas operativos, lenguajes de programación, componentes, módulos, etcétera. Para resolver este problema ofrecemos una organización basada en interfaces de los componentes para la computación distribuida que hemos usado para el desarrollo de sistemas de configuración de objetos complejos. Para lograr esta meta proponemos una arquitectura sofisticada de ambiente de aplicaciones distribuidas donde agentes de toma de decisiones interoperan y cooperan sobre objetos distribuidos y configurados a través de middleware *especial. La arquitectura de este* middleware *se ha desarrollado como ambiente distribuido apoyado por agentes inteligentes de proyectos. La integridad de propiedades y consistencia parcial de proyectos distribuidos complejos están soportados por métodos especiales basados en la integración de bases de datos y bases de conocimiento, así como en reglas de gramática de grafos.*

Abstract

We are at the beginning of the age of distributed computing. The basic idea of this approach is the development of application infrastructure so that the user does not need to know about computers, networks, operating systems, programming language, components, modules, etc. To meet this challenge, we offer an interface-based organization of components for distributed computing which we have used for the development of configuration systems of complex projects. To reach this goal, we propose a sophisticated architecture of a distributed application environment where decision making agents interoperate and cooperate over a distributed configured object by means of special middleware. *The architecture of this* middleware *has been developed as a distributed environment supported by intelligent project agents. Property integrity and partial consistency of distributed complex projects are supported by special methods based on data base and knowledge base integration and distributed computing using graph grammar rules.*

1. Introducción

Las tecnologías de información distribuida (IT's) proveen un acceso universal transparente a recursos distribuidos, y como existe una gran variedad de tipos de computadoras, redes y sistemas operativos, la computación empresarial es hoy en día una colección de diversas partes que trabajan en conjunto con un fin común. La tecnología cliente/servidor es actualmente un desarrollo estratégico en computación, que no suministra un modelo uniforme de conocimiento al compartir la información entre los usuarios y los sistemas; además tiene numerosos problemas, entre los que destacan los concernientes al crecimiento, la modularidad y el mantenimiento de una aplicación.

Tradicionalmente la configuración es considerada como una secuencia de estados entre los cuales destacan la formulación del problema, el diseño conceptual, la síntesis de parámetros, etcétera. Se ha desarrollado un análisis para evaluar las soluciones factibles y las características del objeto, las cuales son determinadas secuencialmente, destacando entre estos estados la solución efectiva.

* Director del Posgrado en Electrónica y Computación de la UTM.
** Profesor-investigador de tiempo completo del Instituto de Electrónica y Computación, UTM.
*** Tesista de la licenciatura en Ingeniería en Computación, UTM.

El progreso de la información tecnológica distribuida permite combinar diferentes configuraciones de estado, usando agentes basados en conocimiento funcional distribuido. Una configuración distribuida es un proceso de relación cooperativa en la cual trabajan agentes (diseñadores o expertos) en los diferentes aspectos o partes de un DC-objeto con el fin de promover acciones de integración en el consenso de los archivos. Los datos generados por los agentes deben ser compatibles con los DC-objetos y con los datos formados concurrentemente por otros agentes de cada DC-objeto.

Hoy la ingeniería concurrente *(Concurrent Engineering-CE)* necesita de una buena caracterización del modelo operacional; hay una significativa necesidad de formalizar la decisión cooperativa distribuida y concurrente, para lograr una interrelación de procesos que puedan ser manejados efectivamente por agentes basados en tecnologías de ambientes sin restricciones. El DCS, como una especialización intelectual en ambientes, puede facilitar el mantenimiento distribuido por medio del mecanismo de agentes para coordinar el desarrollo de los DC-objetos, pero hay muchos problemas relacionados con este desarrollo, por ejemplo: la colaboración en la actualización de procesos independientes, los cuales tienen que calcular la resultante de los datos concurrentes con una coordinación "semántica"; la integridad y consistencia de los datos y el modelado de las actividades, las cuales pueden ser complementadas por más de un objeto o por variantes.

El problema de configuración se formula como un intervalo del problema restricción-satisfacción, el cual asume que los valores del dominio son intervalos e intervalos aritméticos estándar usados para evaluar nodos y arcos de consistencia.

2. Tecnologías de información distribuida basada en objetos

La idea principal de la tecnología de información es dividir las aplicaciones complejas de programación en componentes o módulos fácilmente desarrollables, comprensibles, diseñándolos como módulos activos denominados agentes; estos componentes podrán trabajar en conjunto sólo si han sido diseñados y construidos bajo las interfaces estándar. La arquitectura basada en componentes para la construcción de aplicaciones de sistemas provee una lista de interfaces comunes para integrar sus diferentes componentes hacia una solución completa. En la actualidad la tecnología de la información está comprometida en solucionar la carencia de interoperabilidad de aplicaciones entre las diferentes plataformas de computadoras, y el ensamble de estos componentes comienza a ser el principal problema para el desarrollo de las aplicaciones distribuidas.

Para resolver este problema se necesita de una infraestructura capaz de proveer la invocación de programas remotos y la comunicación con ellos; la comunicación de las aplicaciones a través de la red, se logra con algún *middleware*. El *middleware,* como los procesadores de transacciones, conectores de bases de datos, protocolos de comunicaciones, entre otros, proveen de diferentes clases de infraestructura, donde la interacción de los objetos es perfecta a través de un sofisticado sistema de mensajes permitiendo a los agentes hacer requisiciones de servicio de otros agentes.

La tecnología de objetos está progresando y para poder proveer los principios de organización para la siguiente generación de la arquitectura cliente/servidor se fundamenta en la Computación Distribuida Basada en Objetos (ODC), la cual está encaminada a la computación cliente/servidor orientada a objetos. Las ventajas que resultan de este ambiente de computación distribuida basada en objetos son las siguientes:

1. Las aplicaciones conformadas por objetos distribuidos son una imagen única de la empresa.
2. Los servidores son transparentes hacia los clientes, los cuales pueden ser implementados como agentes migratorios.
3. Tanto los objetos remotos, como locales, se comunican de la misma manera a través de mensajes.

4. El particionamiento permite a los usuarios un cambio de recursos de cómputo en sus accesos.
5. La envoltura de los objetos (interfaces orientadas a objetos) está elaborada para la comunicación entre ellos.

Los objetos distribuidos (DC-objetos) se presentan al usuario como algo muy familiar, no como máquinas, redes o lenguajes de programación; son actores, jugadores con su propia actividad o sustitutos de cosas del mundo real o representación de algunos conceptos. Los módulos pueden ser cubiertos y aparecer al desarrollador como agentes y cuando se cuenta con esta cubierta, el código puede participar en un ambiente ODC. El envolver o cubrir es una técnica para crear una interface basada en objetos, con la cual se pueden accesar de una manera específica y funcional los contenidos de una o más de las aplicaciones existentes en una computadora.

2.1 Arquitectura de la computación distribuida basada en objetos

La ODC es uno de los paradigmas de la computación que admite distribuir objetos a través de una heterogénea red permitiendo a cada uno de sus componentes interactuar como una sola unidad. Este paradigma provee de una infraestructura para abastecer los componentes de servicios disponibles y así reunir los procesos cooperativos distribuidos de una forma universal y transparente. La ODC es un avance dramático en los formatos de computación, es un resultado de la convergencia de la tecnología orientada a objetos y la tecnología cliente/servidor, mezclando las ventajas de distribución de la tecnología cliente/servidor con la riqueza de información del mundo real contenida en los modelos orientados a objetos.

Los DC-objetos pueden distribuirse en diferentes computadoras a través de la red, y la ODC puede verlos como una red global de clientes y servidores heterogéneos y cooperativos. La ODC satisface al paradigma al reunir de manera efectiva los cambios punto a punto en los accesos cliente/servidor, donde el cliente servidor global de la red es una computadora distribuida basada en objetos; también cuenta con provisiones para el paradigma universal de construcción, transparente y adaptativo en las infraestructuras de información y sistemas.

La ODC propone un sofisticado sistema de servicio proveído por los DC-objetos partidos en agentes (P-agentes), para garantizar la integridad de propiedad y la debilidad en la consistencia de los datos objeto. Una arquitectura ODC tiene que soportar la interoperación de los actos concurrentes de los DM-agentes sobre las partes de los DC-objetos, manteniendo una coordinada actualización en los conflictos de fuentes comunes entre los procesos de los DC-objetos, por tanto hay una reducida carga en los DM-agentes. Los servicios de la ODC están bien solicitados en el ambiente distribuido; esto es, en forma dinámica y proveyendo diferentes soportes en la colaboración de alto nivel.

Esta arquitectura es una consecuencia de la selección de la representación del conocimiento y la estructura de datos orientados a objetos, ambas usadas por los modelos de objetos. Los P-agentes son interfaces flexibles y programables entre DM-agentes y objetos distribuidos. Existen implementaciones aproximadas de tres filas *(three-tie-red)* para controlar todos los objetos de operación escritura/lectura. La arquitectura ODC es usada por los P-agentes para entender el contexto dentro del cual están sucediendo las actualizaciones de los eventos. Los P-agentes utilizan técnicas de inferencia para hacer factible las operaciones de los DM-agentes sobre los DC-objetos y poder conceptualizar una vista del ambiente de DCS como un *middleware*.

Los DM-agentes se comunican regularmente a través de redes estructuradas, las cuales están soportadas por P-agentes; cada P-agente está conectado con una lista de DM-agentes desarrollando las partes de los DC-objetos más un número de P-agentes para poder soportar la coordinación de datos comunes compartidos entre las partes de los DC-objetos; también un DM-agente puede contener una lista de canales para intercambiar con P-agentes en las diferentes plataformas.

Las comunicaciones están establecidas mediante una estructura de intercambio de mensajes la cual interopera de acuerdo con la lista preestablecida en el formato de transferencia (protocolos). Estas interoperaciones están basadas en un grafo de transformación de especificaciones de los agentes de acción. Los DM-agentes pueden ser distribuidos con la información incompleta, pero utilizando una forma semánticamente rica en la descripción de las posibilidades del diseño basado en fragmentos.

Los DM-agentes pueden activar algunos actores como agentes computacionales (C-agentes) para que decidan entre las diferentes tareas computacionales. Un C-agente siempre espera una llamada de comunicación y puede interpretar dicho mensaje de acuerdo a su estado real. Un C-agente puede generar un nuevo C-agente para que decida nuevas sub-tareas.

3. Funciones de los agentes

Los sistemas cooperativos están compuestos de una lista de agentes (llamados sistemas multi-agente-MAS) y son considerados sistemas modulares de ODC. La teoría de sistemas multi-agentes surge debido a la tendencia del desarrollo de sistemas inteligentes y a la computación distribuida. Aunque desafortunadamente todavía no existe una metodología que permita analizar, especificar, diseñar e implementar estos sistemas multi-agentes, nosotros especificamos el conocimiento de cada agente interno y su comportamiento en el mundo externo, además de sus interacciones con otros agentes. Esto significa que la comunicación entre ellos se da con especificaciones semánticas, por lo que la corrección de los protocolos de interacción en sistemas específicos está garantizada.

Para especificar la mayor parte de los sistemas multi-agentes es necesario definir el conocimiento interno que le pertenece a cada agente y sus funciones que proceden del mundo exterior, muy cerca de la interacción con otros agentes. En nuestra investigación el modelo del sistema multi-agentes es usado por el modelo de ambiente cooperativo-DCS, en donde el MAS toma ventaja sobre la participación dinámica de cada uno de los miembros del grupo, lo cual constituye un avance.

Un DM-agente puede usar toda una base de datos, pero las operaciones de actualización sólo pueden ser efectuadas fuera, en la parte de captura. Un DM-agente tiene que capturar la parte seleccionada de un fragmento de DC-objeto, como protección de posibles actualizaciones no coordinadas con otros DM-agentes. Un DM-agente puede crear nuevos tipos de datos como parte de modelos de un DC-objeto o sus instancias, buscar prototipos para diseñar parte de nodos-consistentes de los fragmentos capturados, transformando los datos de ciertas operaciones mediante cálculos de intervalos para definir valores restringidos para los atributos y, finalmente, enviar sus propósitos al P-agente para su aceptación o rechazo.

La propuesta hecha localmente por un DM-agente está disponible de inmediato para los DM-agentes que usen elementos de interface para capturar el fragmento. En general, la factibilidad de las operaciones de actualización de un P-agente sobre un DC-objeto está restringida por la semántica de datos de los objetos; es decir, las propiedades de un DC-objeto y sus requerimientos aplicables a él. La coordinación de las acciones concurrentes de los DM-agentes están basadas en los conceptos de integridad de propiedad y la consistencia de su cobertura para una base de datos en la cual la idea de "actividad de un objeto" es usada de acuerdo con las posibles operaciones definidas sobre el estado del fragmento.

Los mensajes entre los P-agentes y los DM-agentes son soportados por la arquitectura DCS y consiste en referenciar los fragmentos capturados, sus transformaciones, activos restringidos, etcétera. Los mensajes P-agente permiten DM-agentes, por ejemplo, para organizar una efectiva búsqueda de partes constitutivas de un fragmento diseñado con propiedades compatibles.

4. Integridad y consistencia de datos en los accesos

Las reglas de configuración para las acciones específicas de actualización, como crear, borrar, insertar y modificar, deben satisfacer las reglas de integridad y consistencia de los datos. En este caso solamente un modelo DC-objeto puede contener todos los mecanismos necesarios para coordinar una actualización concurrente. Esta ventaja es motivada por el módulo de la filosofía de la ingeniería concurrente donde ambos, el objeto desarrollado y los aspectos de administración de la ingeniería concurrente, son integrados. La idea básica de proponer esta ventaja es incrementar las posibilidades de expresión de una base de datos para la representación semántica de sus relaciones entre las partes de una entidad; por ejemplo la representación de las compatibilidades semánticas de un modelo de objeto para el soporte de un objeto agregado. Estas posibilidades se conservan, lo cual significa la inclusión del conocimiento al modelo de objeto que describe la compatibilidad restringida entre sus componentes.

Una base de datos y una base de conocimientos modelo pareja (DB & KB) se considera como una integración de un objeto modelo de datos y la representación de su conocimiento con el mecanismo de coordinación que usa una semántica específica para sus relaciones con cada DC-objeto y sus variantes. Esto se desarrolló para organizar los datos y el almacenamiento del conocimiento, así como la actualización concurrente en la búsqueda de decisiones eficientes y factibles basadas en mecanismos coordinados.

Para soportar el modelo de consistencia de DC-objetos introdujimos los conceptos de propiedad de integridad y la cobertura de consistencia de datos de los DC-objetos mediante la semántica de datos. La propiedad de integridad incluye la demanda de compatibilidad de propiedad para partes pertenecientes a una simple estructura identidad. Esta nueva cualidad es usada para analizar la factibilidad de los DM-agentes de decisión y su agregación al modelo de DC-objetos.

Una de las ideas para la deducción clásica de consistencia en las bases de datos es la noción de un modelo de mundo cerrado. Una base de datos es consistente si es un modelo verdadero de un mundo dado. Por "verdadero" se entiende la ausencia de hechos contradictorios existentes en el mundo (DB), tanto como los obtenidos de las aplicaciones de las leyes del dominio del problema. Estas leyes son el significado para la captura de un rico mundo de semánticas.

En el mundo real cada objeto tiene una única cualidad, un único proceso para tomar decisiones, reglas de ingeniería, constreñimiento y requerimientos. Esto significa que cada DC-objeto es el resultado de un proceso colaborativo basado en hechos únicos, conocimiento, toma de decisiones y reglas de diseño. Para satisfacer la demanda de nociones de la cobertura de consistencia para DB & KB, se introduce la consistencia dentro de un mini-mundo de cada DC-objeto.

Todos los objetos son incluidos en un objeto por referencia de la persistencia de objetos. Los modelos DB & KB propuestos no son una unión disjunta de mini-mundos de todos los proyectos. Los mini-mundos de dos DC-objetos pueden ser interceptados si uno de los DC-objetos usa alguna parte de otro DC-objeto como un fragmento. Entonces el segundo DC-objeto acepta todos los datos y conocimientos del primer mini-mundo en la forma de un modelo fragmentado. Ambos, el primero y el segundo DC-objeto, deben ser consistentes con respecto a los hechos relativos y al conocimiento de los fragmentos compartidos.

4.1 Fragmentación

Un fragmento es una parte de un objeto que se interpreta como un objeto u objetos interrelacionados con restricciones entre ellos. Los fragmentos son unidades de toma de decisión y distribución. Cada DM-agente puede ser referenciado como una toma de decisión de un fragmento cualquiera en un nivel conceptual como definición de este modelo, o en un nivel paramétrico como definición de propiedad. El diseño conceptual, como un paso de la tarea de configuración, se exporta por la definición de este mo-

delo en forma de una clase de objeto usando un mecanismo de agregación. Las operaciones en este nivel pueden generar nuevas clases, agregando subclases y definiendo una restricción entre ellas. La propiedad del fragmento diseñado es definida en el nivel paramétrico de ingeniería de acuerdo con la restricción existente.

El estado actual del fragmento y sus transiciones a otro estado se dan por una lista de reglas con condiciones. La principal dificultad se presenta por la necesidad de incluir una nueva aplicación con acciones de larga duración. Para aplicaciones ordinarias complejas las acciones DB se unen a un acceso atómico de transacciones para actualizar los fragmentos capturados. La concurrencia de transacciones de larga duración no pueden ser ejecutadas secuencialmente. En el orden de realizar una ejecución bien definida de estas acciones, se desarrolla el criterio de menor número de correcciones basada en el grafo de transformación paralelo (GT). Este criterio garantiza que una ejecución distribuida de una lista de acciones de DM-agentes sea equivalente a alguna ejecución de un componente (amalgamado) GT, el cual es resultado especial de un P-agente para todas las actualizaciones aplicadas a la parte del DC-objeto concurrente.

5. Modelos de grafos de transformación

5.1 Ventaja sobresaliente

La teoría de transformación de grafos provee un paradigma uniforme y un formalismo flexible para la especificación, la implementación y el análisis secuencial de sistemas distribuidos y concurrentes. Las técnicas basadas en grafos se aplican exitosamente en el desarrollo de DCS, representando un DC-objeto como una estructura jerárquica de grafos atribuidos para el modelo comunitario de objetos de multi-nivel.

La ventaja de GT se usa para el modelo concurrente y distribuido de acciones del DM-agente y a la vez coordina la acción entre ellos. Un sistema GT es una colección de reglas GT basadas en la llamada ventaja única de *pushout,* donde *pushout* corresponde a un término de unificación. Un paso en GT se define en términos de producción y derivación de grafos. Una producción GT se presenta mediante morfismos y aplicaciones de producción en subgrafos, los cuales son grafos que se describen en el diagrama conmutativo llamado *pushout* (PO). Se usa la ventaja algebraica de la representación GT en la categoría de atributos de grafos y por lo mismo se crea su independencia, lo cual permite aplicar el paralelismo, la concurrencia y la unificación. De esta manera la actualización concurrente de los estados DC-objeto se modelan por la noción del grafo de unificación para la computación concurrente.

El GT corresponde a una operación sobre el estado de un DC-objeto representado por un atributo global del grafo. Un DC-objeto se modela por una segmentación del estado de un grafo en un conjunto de localidades con una interface de grafos global (IGs). Los segmentos de un grafo DC-objeto y los grafos locales con sus IGs globales, se describen en la figura 1. La concurrencia directa GT de dos actualizaciones simultáneas DC-objeto son independientes en forma asíncrona, o unificados de manera síncrona.

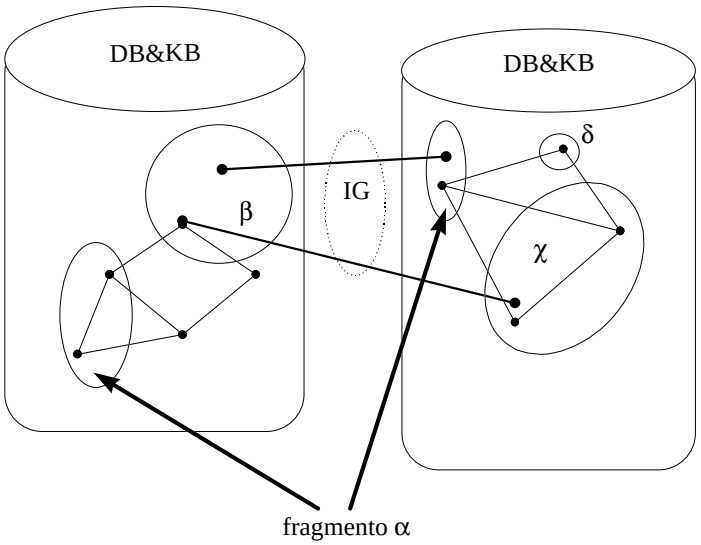

Figura 1. Segmentos de un grafo DC-objeto

Cada P-agente mantiene un grafo local, GT es local si puede ser ejecutado por un P-agente y es global si en su implementación se usan dos o más P-agentes. Un P-agente puede coordinar actualizaciones de DM-agentes locales paralelos y proveer una colaboración global concurrente de GTs, así las partes distribuidas integran un modelo de objetos consistente, los cuales se unen a grafos locales distribuidos a lo largo de sus IGs. De esta forma todos los pasos distribuidos para un objeto son sincronizados y distribuidos. Los GT globales se consideran como el inicio de una secuencia compuesta de segmentos, uniones y pasos de distribución.

Un GT puede representarse formalmente de la siguiente manera: r corresponde a las reglas de un GT y su representación es r=L® R, donde L y R son grafos y el patrón es incluido en la entrada dada por el grafo G: m=L® G. Aquí L es sustituida por la segmentación R en G, creando un nuevo grafo H. Generalmente a través de las reglas r se forma un GT de un grafo G por la eliminación de algunas partes de Del de G y define algunas partes de K_g en el contexto de Del. De inmediato se agrega una nueva parte de Add con un nuevo contexto de K_h, resultando un grafo derivado H, donde K_h es construido desde K_g y los nodos y arcos del contexto están estrechamente unidos.

El principio de la construcción de las fórmulas de unión para derivaciones concurrentes en grafos se da en EH, por lo tanto nuestras investigaciones se enfocan al desarrollo de estos métodos formales para GTs condicionales, basados en el análisis de la consistencia.

5.2 Transformación de atributos de grafos

Las especificaciones GT consisten en objetos y referencias entre ellas, representando condiciones de compatibilidad, por lo cual nosotros usamos una lista ordenada para la evaluación de atributos de objetos. Las reglas GT descritas por los morfismos y el álgebra se usan como un dominio semántico de las especificaciones de los datos de un grafo, el cual va equipado con el componente tipo de datos, como atributo de un grafo codificado gráficamente en estructuras con propiedades de objetos. Los atributos de los grafos se usan en el modelo de evaluación de datos de un objeto como combinaciones algebraicas de una estructura gráfica y otra del tipo componente de datos. Las transformaciones directas se forman en un paso para las transformaciones de estructura de grafo y componente de datos, y relaciona sus pasos donde los atributos se componen con condiciones de consistencia adicional. Las composiciones de transformación se integran con la composición doble de grafos y la transformación de datos.

Los tipos de datos se definen algebraicamente. La sigla para el atributo de un grafo debe ser la misma en la estructura gráfica (llamada estructura G). La sigla para el tipo de datos es DT-set, y los atributos a los cuales se les asignan valores de datos de objeto a objetos se representan como arcos de una forma especial.

Una estructura G es un modelo parcial algebraico en donde se permiten todos los tipos de operación. Las reglas GT son morfismos parciales y satisfacen diferentes condiciones. Las reglas de la estructura G son inyectivas para una lista de transformaciones D, y son isomórficas para toda la organización de datos A. Sintácticamente, con el álgebra se pueden representar variables con base en siglas que contengan sólo datos organizados y operaciones de datos para los tipos de datos seleccionados.

Los fragmentos de objetos son entidades persistentes que sobreviven a métodos de ejecución. Las configuraciones de los objetos se modelan por varias selecciones algebraicas y cada configuración representa una comunidad estructurada de objetos, la cual refleja la arquitectura de interconexiones de los componentes.

5.3 Condiciones de aplicación del GT

La consistencia de un sistema GT se basa en el uso del teorema del paralelismo para la derivación de grafos con condiciones de aplicación contextual (CACs) con restricciones de consistencia. La aplicabilidad de las reglas GT

se controla directamente por CACs en un nivel semántico y se representa como subgrafos generalizados vinculados. Para la interconexión de los grafos existen reglas específicas hacia la derecha o izquierda y se verifican en el tiempo de ejecución; antes de que CACs aplique la regla, puede expresar un contexto de condiciones positivas o negativas, así como las combinaciones entre ellas, que consisten en conclusiones de premisas disyuntivas. CACs puede, por ejemplo, describir de manera única y afirmativa los elementos en un grafo, sus tipos, propiedades o ciertas restricciones concernientes a las siglas de la lista del modelo DT. Para satisfacer las condiciones de aplicación se requiere la consistencia de las especificaciones GT y la consistencia de un modelo de objetos de datos que se codifican dentro de los atributos de un grafo usando identificadores adicionales expresados en fórmulas lógicas. CACs puede caracterizarse por medio de ecuaciones, o no, para la representación de intervalos restringidos. Nosotros usamos estas condiciones para las especificaciones de consistencia y apoyar las propiedades de integridad en el modelo de objetos distribuidos.

Las condiciones conceptuales de la aplicación comparten toda una estructura donde las restricciones son totalmente morfismos L® L´ y se satisface por la pareja L® G. Esta ventaja se define como y bajo las reglas GT, pueden ser aplicadas con CACs, que provee niveles intermedios para expresar la potencia entre la gramática libre de contexto y los grafos de gramática sensitiva de contexto.

Una solución en el intervalo CSP es la asignación de valores para cada fragmento. Cada dominio puede ser multi-evaluado desde cada atributo y caracterizado por una lista de intervalos de atributos. El sistema GT con CACs es consistente si el grafo inicial satisface las condiciones de consistencia y las reglas de preservación de esta propiedad.

6. Coordinación

La coordinación entre dos DM-agentes se expresa por la sincronización de sus P-agente(s) que permitan la transferencia de fragmentos capturados a procesar sobre el P-agente(s). La coordinación en DCS permite la resolución de intereses conflictivos (recursos) y la interoperación de DM-agentes. El DCS distribuido se construye en el nivel superior de un mecanismo de comunicación, para efectuar el trabajo cooperativo por interoperación a través de la red, vinculado a la ingeniería para lograr tanto las metas individuales como las comunes.

6.1 Estructura dinámica virtual

Para resolver un problema de coordinación de GT en forma local, cada P-agente desarrolla una estructura dinámica virtual (VDF) de la parte del objeto que apoya, para generar el espacio del arco de consistencia. Un VDF es un grafo específico con nodos como DM-agentes y arcos como restricciones externas para los fragmentos capturados por ellos. Los arcos entre dos nodos son IG locales de DM-agentes y se usan para coordinar sus acciones. Un P-agente identifica cómo un nuevo DM-agente puede incluirse en un VDF y cómo incluir las actualizaciones del DM-agente al objeto. Un VDF es lo más adecuado para representar diseños individualmente y permitir todas las restricciones activas para que se consideren simultáneamente por parte del DC-objeto.

Un P-agente posee información completa sobre las limitaciones, variables y dominios de valores para encontrar estados consistentes de VDF; además utiliza los conceptos de consistencia en los nodos y arcos para determinar y apoyar la integridad y consistencia de datos para las decisiones del DM-agente, capturando todas las posibles interacciones entre el DM-agente y sus acciones coordinadas, traduciendo los requerimientos del objeto en las especificaciones de restricciones, y los envía a todos los DM-agentes pertinentes. El P-agente usa un mecanismo cronológico de búsqueda para seleccionar otro conjunto de DM-agente propuesto, si se alcanza un ciclo final.

6.2 Amalgamación

El sistema GT especifica las acciones con diferentes estrategias y permite implementar ejecuciones distribuidas

de operaciones sobre el DC-objeto, así provee la semántica operacional a través de GTs. El mecanismo de sincronización se representa como una amalgamación de grafos con reglas de GT y se emplea con el fin de definir la semántica operacional para un sistema GT. La técnica de amalgamación se usa para describir el efecto de interconexión que provee un mecanismo del control de proceso GT.

Los GTs se ejecutan concurrente e independientemente uno del otro y por su tipo de interacción se distinguen si son síncrono o asíncrono. Dos GTs pueden sincronizarse mediante operaciones GT, utilizando IG comunes, cambiando éstos durante el proceso de las operaciones. En este trabajo un GT síncrono se considera como un caso especial dentro de los asíncronos.

6.3 Envío de mensajes e IG compartidos

Una de las características principales de los sistemas distribuidos es la naturaleza local del cómputo. Un conjunto de DM-agentes, conectados en alguna forma específica, trata de alcanzar una meta común. Las propiedades globales del objeto pueden ser computadas con el significado de GR local, o global, sobre DC-objetos. La arquitectura compartida de DC-objetos es una estructura de red dinámica que integra sus componentes en DCS con recursos comunes, incrementando su rehuso, modularidad y portabilidad de sus componentes.

Si uno de los DM-agentes cambia al IG compartido, el otro tiene conocimiento del cambio. Esto ocurrirá independientemente de que las partes sean distribuidas, o no; este mecanismo se usa para la actualización de la coordinación. La sincronización se conforma cuando todos los procesos compartidos IG tienen que esperar ser coordinados por sus acciones.

El DCS contiene dos mecanismos para la coordinación basados en partes del IG-compartido.

La *coordinación de procesos paralelos* se produce cuando dos DM-agentes capturan fragmentos apoyados por un P-agente (caso no distribuido). Esta situación se resuelve por un P-agente que usa un IG local compartido para dos nodos de VDF apropiados para estos agentes. La sincronización se desempeñaría entonces en todos los procesos de los IGs compartidos teniendo que esperar la coordinación de sus acciones.

La *coordinación de procesos concurrentes* se da cuando el DM-agente captura el fragmento apoyado por dos o más P-agentes. Una coordinación de acciones distribuidas usa el modelo de paso de mensajes entre P-agentes para la comunicación de las actualizaciones sincronizadas del DC-objeto.

El mensaje del DM-agente se transfiere usando el enfoque *3-tiered* de acuerdo a una colección global de actualizaciones para todos los P-agentes pertinentes. Los IGs globales como objetos compartidos son accesibles por todas las partes de un GT global que esté involucrado.

6.4 Negociación y comunicación de agentes

Para la correcta cooperación y coordinación de trabajo en un MAS, donde la actuación de los agentes es en forma interdependiente, debe existir un mecanismo adicional que les permita una colaboración y coordinación factible entre ellos. Para definir un método de interacción entre agentes, los procesos de negociación se especifican como la coordinación y cooperación.

La cooperación significa que la solución de un problema determinado sea el resultado de la interacción cooperativa entre todos los agentes; la coordinación entre un grupo de agentes permite considerar todas las tareas a realizar y coordinarlas, ya que es una decisión de compatibilidad entre ellos. A continuación se mencionan algunas características de MAS:

1. Conocimiento de todos los agentes del sistema, es posible ubicarlos cuando envían o reciben los requerimientos a otro;
2. Conocimiento del papel o rol que cada agente cumple en la interacción, y en cada caso es posible saber a quién puede dirigirse;

3. Contar con una opinión de cada uno de los agentes (conocimiento), siendo posible tener "criterios" en la toma de decisiones, entre otras.

6.4 Modelo de negociación

La negociación es un mecanismo para resolver los conflictos haciendo lo posible por alcanzar un consenso entre DM-agentes. La tarea computacional de una configuración es la formalización de un intervalo de satisfacción de restricciones (ICSP). El ICSP asume que los valores del dominio son intervalos y usan un intervalo aritmético estándar para evaluar la consistencia de nodos y arcos. En la implementación actual se asume que todas las limitaciones son monolíticas, sin embargo nosotros investigamos el problema de cómo los DM-agentes que actúan de manera concurrente y resuelven conflictos por medio de negociaciones con un P-agente, de tal forma que los problemas de diseño pueden ser resueltos efectivamente en su totalidad. Un P-agente necesita ser capaz de evaluar las propuestas del DM-agente para definir la consistencia de datos y al fin se comprometa, tomando en consideración su contexto.

Las estrategias de negociación incluyen la selección de decisiones efectivas cuando existen múltiples soluciones de fragmentos en forma factible; si alguna solución no es factible es necesario omitir algunas restricciones para lograr la solución al problema.

Cada DM-agente trata de encontrar soluciones de fragmento con base en criterios (conocimiento) propios y envían propuestas al P-agente, el que evalúa estas propuestas concurrentemente y si satisfacen restricciones de intervalos entonces el P-agente envía el mensaje de confirmación y escribe las propuestas en DB & KB.

Si el acuerdo no puede alcanzarse, el P-agente envía mensajes para disminuir u omitir algunas restricciones al DM-agente que esté participando en cada conflicto. Estos pasos se repiten hasta que se logre una decisión satisfactoria o se reconozca que el conflicto no puede decidirse debido a las restricciones abrumadoras y se aborta la operación o tarea.

6.5 Comunicación

Los DM-agentes comunican lo establecido por medio de canales a través de redes regularmente estructuradas. La comunicación de agentes se representa como un conjunto de esquemas de mensajes con la siguiente estructura: El vocabulario es una parte de dominios específicos donde cada palabra tiene una anotación formal escrita en KIF (Formato de Intercambio de Conocimiento).

Un KIF es una versión de prefijo de cálculo de primer orden con diversas extensiones para mejorar su expresividad, y define un conjunto de objetos, funciones y relaciones cuya semántica es fija. Pero es abierto y los usuarios son libres para definir el significado de cualquier otro símbolo que no esté predefinido. Un mensaje es una expresión KQML (Preguntas de Conocimiento y Lenguaje de Manipulación) en donde los argumentos son los términos o frases del vocabulario en KIF.

7. Computación distribuida basada en objetos: especificaciones e implementación

7.1 Arquitectura ODC

ODC es un ambiente complejo de computación y requiere una infraestructura con una tecnología muy sofisticada y robusta. Las infraestructuras IT distribuidas del futuro deben ser elaboradas con base en arquitecturas muy confiables.

Una arquitectura es un nivel descriptivo alto de la organización de las responsabilidades funcionales dentro de un sistema, y además es la estructura general del sistema que define la relación de componentes del mismo; por ejemplo, cliente/servidor es la arquitectura que define una relación entre el consumidor y proveedor de recursos.

Los siete niveles básicos que el modelo de la tecnología de objetos apoya para proveer el procesamiento orientado a objetos son:

1. La capa de programación OO es tradicional al modelo OO y herramientas de programación.
2. La capa de implementación provee la infraestructura para el ambiente basado en el objeto.
3. En el nivel conceptual los desarrolladores crean las "entidades" básicas del dominio del problema, sus asociaciones con unos y otros, así como sus respectivos papeles en la configuración.
4. Las reglas de mantenimiento y restricciones limitan la definición y mecanismos de inferencia.
5. El nivel de eventos permite a los usuarios redefinir sucesos de comunicación anteriormente definidos.
6. El nivel de escenario permite ensamblar entidades para obrar recíprocamente en algún escenario del dominio del problema.
7. El nivel de agentes organiza sucesos múltiples en un flujo de tareas o trabajos que son capaces de reconocer a ambos el contexto y contenido de la información.

7.2 Tecnologías de objetos distribuidos: componentes opuestos a los objetos

Un modelo de computación basado en el paradigma de objetos puede proveer grandes ventajas y un alto rendimiento. La interacción del objeto se realiza mediante un sofisticado ambiente de mensajes que permite a los objetos solicitar servicios a otros objetos.

Los objetos VE necesitan de las ventajas de la arquitectura cliente/servidor distribuida que apoye la distribución, independencia, continuidad y procesamiento en tiempo real.

La integridad semántica de objetos con el potencial de la arquitectura cliente/servidor se concibe como un gran reto del cómputo VE.

Los componentes (agentes) interactúan en forma recíproca a través del paso de mensajes representando solicitudes para la obtención de información o servicios. Durante la interacción, los objetos asumen dinámicamente los papeles de clientes/servidores y la amalgamación de estos objetos distribuidos se unen en un Objeto de Solicitudes (ORB). El ORB provee los medios para localizar y activar otros objetos en la red, efectuándose de forma transparente a los usuarios. El ORB es el *middleware* de ODC que permite la funcionalidad de interoperabilidad en redes heterogéneas de objetos.

Una de las contribuciones claves de la tecnología orientada a los objetos es ocultar los detalles de la implementación dentro de componentes para manejar con más eficiencia la complejidad en ODC. El usuario no tiene que estar tan enterado de plataformas y ambientes de programación, simplemente debe ver la interacción de los DC-objetos que colaboran recíprocamente como un todo unificado (multiagentes).

7.3 Tecnologías orientadas a objetos

Un objeto es una pieza de código fuente de software que puede usarse para construir parte de una aplicación. El término orientado a objetos (OO) se aplica a numerosas áreas tecnológicas como: lenguajes de programación, análisis y diseño de métodos, bases de datos y sistemas operativos. Los lenguajes OO son los más eficientes medios para expresar los conceptos e ideas en implementación que se detallan en un código fuente. Para que un lenguaje de programación sea totalmente orientado a objetos, debe apoyar los tres pilares del paradigma de objetos: encapsulación, herencia y polimorfismo.

La *encapsulación* permite al programador definir modelos que contienen tanto métodos (comportamiento) como atributos (datos) con diferentes niveles de acceso.

La *herencia* permite que los datos y procedimientos de un modelo (la clase) se definan una sola vez y se reutilicen por subclases de objetos, lo cual conduce a estructuras llamadas jerarquía de clase (una clase hereda comportamiento [métodos] y estructuras de datos [atributos] desde la clase base).

La *herencia múltiple* es un poderoso concepto que provee la capacidad para heredar directamente desde dos o

mas clases; pero no es indispensable para considerar a un lenguaje como orientado a objetos.

El *polimorfismo*, que literalmente significa "muchas formas", permite a un segmento de código de programa enviar un mensaje a un objeto sabiendo que el objeto receptor responderá correctamente aun cuando la clase precisa del objeto no es conocida. Así familias enteras de objetos pueden compartir los mismos nombres de métodos que permitan a otras aplicaciones reutilizar grandes partes (componentes) de código desarrollado.

7.4 Cómputo basado en componentes

Un componente es un módulo de software de trabajo. La diferencia entre los componentes OOP y CC es semejante a la que existe entre un producto y la especificación de ingeniería de ese producto. Esta especificación contiene la información que es útil para crearlo, pero no establece su utilidad. Las ventajas de los componentes de software son las siguientes:

- Expansión de sistemas operativos basados en arquitecturas subyacentes de objetos.
- Interfaces estándar programables con base en un sistema binario robusto como estándar de objetos
- Transparencia corporativa con los servicios de sistemas distribuidos basados en objetos.
- Modelos de sistemas multiplataformas basados en objetos con componentes autónomos de software
- Generación de arquitecturas cliente/servidor: primero como *two-layers* y después como *three -layers*.

La *implementación de la herencia* consiste en que los componentes pueden reutilizarse fácilmente a través de diferentes aplicaciones mediante la herencia de clases, que es una manera en que un objeto llama con facilidad a otro objeto para proveer un servicio. Una aplicación basada en objetos puede crearse como una tarea independiente en un espacio de proceso separado -archivos con extensión EXE-, proveyendo servicios a servidores remotamente, o como una biblioteca de enlace dinámico en el servidor que se procese en un mismo espacio -archivos con extensión DLL-, que proveen servicios mediante la función de una llamada local. La aplicación de objetos ejecutables que corre en un espacio de proceso separado se refiere tanto a servidores locales como a remotos.

Las aplicaciones de objetos que se ejecutan en el espacio del proceso del servidor se conocen como *in-process* de servidores, y cuando se realiza una aplicación de objetos, se reemplaza la mayoría de la funcionalidad proveída por el gestor de objetos.

7.5 Componentes distribuidos basados en objetos

En esta sección se hace una introducción a las especificaciones e implementación de diferentes sistemas de ODC y sus características claves del ambiente en que operan. Esta comparación nos permite apreciar diversas características de ODC, como la extensibilidad, robustez, transportabilidad y la operabilidad, así como de qué manera estas características ambientales motivan y forman muchos componentes reusables, además otras cualidades que se encuentran en las estructuras ODC.

Los clientes y servidores pueden ejecutarse en diferentes familias de sistemas operativos como UNIX, Windows, O2 o MVS, que proveen diferentes conjuntos de características (multi-hilos de ejecución, memoria compartida, entre otros) y diferentes sistemas de interfaces (POSIX o Win32). Los componentes deben desarrollarse para trabajar con eficiencia y robustez a través de redes. El enfoque de ODC ofrece muchos beneficios potenciales con base en técnicas robustas de arquitectura que existen en las estructuras distribuidas, como CORBA, OODCE y OLE/COM/DCOM. CORBA es una norma emergente para la computación de objetos distribuidos; OODCE es una estructura en C++ para el DCE, y OLE/COM/DCOM es una tecnología de microsoft para componentes distribuidos. Los clientes y los servidores se ejecutan en diferentes computadoras, que se unen por una red heterogénea.

El protocolo de red que conecta los componentes distribuidos puede ser con base en cualquier tipo de familias de protocolos como TCP/IP, X.25, ISO OSI y Novell IPX/SPX, las cuales apoyan punto-a-punto la comunicación y tienen diferentes características. Por ejemplo: TCP/IP es un protocolo de transporte por flujo de bytes que ignora límites de mensaje.

Para proteger y eficientar las aplicaciones se necesita seleccionar unos ODC que operen transparentemente a través de diferentes protocolos a un nivel bajo de comunicaciones, y estas estructuras tienen que proveer componentes reusables que simplifiquen el desarrollo de herramientas y sistemas distribuidos, por ejemplo lenguajes de definición de interfaces (IDLs) y de compiladores. Estas herramientas pueden generar códigos que de manera automática ordenan y reorganizan parámetros de métodos a procesar tanto local como remotamente.

Este proceso convierte datos binarios de un proceso a otro en un formato que es reconocible a lo largo de un sistema heterogéneo de computadoras. CORBA no intenta definir un conjunto estándar de interfaces en un SO, sin embargo el usuario puede definir estas interfaces para acceder a características de estructuras del ODC, como la Interface de Invocación Dinámica (DII) de CORBA. La DII permite a un cliente acceder a los mecanismos de solicitudes provistos en un Agente Objeto de Solicitud *(Object Request Broker*-ORB). Las aplicaciones usan la DII y manejan en forma dinámica las solicitudes a objetos sin requerir interfaces específicas que tengan que ser vinculadas en el componente.

La localización simplifica la administración de un sistema distribuido y promueve la distribución más flexible y dinámica de servicios a través de la automatización en la selección de objetos distribuidos. Tradicionalmente los sistemas cliente/servidor identifican sus servicios por medio de direcciones de memoria que indican la localización de objetos y subrutinas.

El esquema general para dirigir servicios remotos en Internet es a través de puertos numerados (IP). Sin embargo, este mecanismo es inadecuado para un alto grado de escalamiento distribuido, por lo que necesitamos mecanismos más sofisticados y robustos para la identificación y ubicación de los servicios remotos que permitan a los clientes acceder a los servicios de objetos remotos (más que por el bajo direccionamiento de memoria o número de puertos IP).

7.6 Especificaciones e implementaciones

CORBA

El *Object Management Group* (OMG) se formó con el fin de definir las normas requeridas para los sistemas distribuidos basados en objetos en ambientes heterogéneos. Así definen a un objeto como "una representación de la naturaleza y comportamiento de conceptos o cosas mundiales verdaderas en términos que son significativos a un problema común".

El principal propósito de OMG radica en crear estándares que soporten la interoperabilidad entre aplicaciones independientemente desarrolladas a través de redes heterogéneas, y su objetivo es la definición de la Arquitectura de Gestión de Objeto (OMA), que comprende cuatro conjuntos de normas:

1. *Common Object Request Broker Architecture* (CORBA) define los servicios que serán proveídos por un ORB y mantiene el modelo distribuido de objeto que incluye los mecanismos para hacer y recibir peticiones desde otros objetos en forma transparente.
2. *Common Object Services Specification* son colecciones de servicios con interfaces de objeto que provee funciones básicas para usar e implementar la metodología de objetos.
3. *Common Facilities* son colecciones de servicios orientadas a usuarios finales, por ejemplo documentos compuestos.
4. *Application Objects* son aplicaciones específicas basadas en objetos para usuarios finales.

La interoperabilidad ORB es una de las direcciones de las especificaciones de CORBA 2.0. Varias firmas están trabajando sobre productos de cómputo distribuido con diseño de objetos, tomando como base los ORB. Los componentes ORB se implementan en varios productos comercialmente disponibles, como el *Object Broker* de *Digital* o el DOE de SUN. En la actualidad hay muchos productos de cómputo distribuido orientados a objetos disponibles en el mercado.

Dentro de las implementaciones más importantes y actuales de objetos distribuidos están: OMG's CORBA, Microsoft's OLE/COM (DCOM), COSS, CF y BOMSIG; e IBM's SOM/DSOM, Sun's NEO y Java, Taligent's Ambiente de Desarrollo, IONAS Orbix, OCX, NeXT's OpenStep y HP's CORBA 2.0-compliant ORB.

DCE

DCE es el Ambiente de Cómputo Distribuido de la *Open Software Foundation* y se diseña como un modelo cliente/servidor que consta de componentes múltiples que se han integrado para trabajar estrechamente juntos.

DCE es conocido como el *middleware* y debería estar integrado en un sistema operativo, o bien como parte de librerías soportadas por varias firmas.

DCE es una infraestructura para aplicaciones distribuidas suplementarias en un ambiente *enterprise* cliente/servidor; además es una fundación sólida para el cómputo de objetos distribuidos y es un conjunto comprensible integrado a un conjunto modular de productos para apoyar transparentemente la *interworking* y recursos compartidos en sistemas heterogéneos de ambientes en red.

DCE apoya a dos tecnologías orientadas a objetos:

1. Productos de CORBA, basados en DCE. CORBA es una especificación para tecnologías distribuidas de objetos y provee una amplia gama de servicios estándares de interfaces. De los productos disponibles de buena calidad basados en la especificación CORBA, es probable que seleccionemos alguna implementación de una firma específica basada en CORBA a través de la organización de un ambiente DCE.

2. El *Network OLE de Microsoft*. Mientras Network OLE parece ser un buen ajuste técnico con DCE, es muy pronto para saber si se adaptará bien a nuestros requerimientos.

DCE es un enfoque derivado del desarrollo basado en objetos y constituye la base sobre la que se construye la tecnología distribuida de objetos. Además apoya la encapsulación y el polimorfismo de dos de las tres características requeridas para la orientación a objetos. La piedra angular de DCE RPC es el lenguaje de definición de interface (IDL) que permite especificar y definir los atributos externos de un conjunto de operaciones de un servidor.

Microsoft OLE/COM y ActiveX/DCOM

El COM (DCOM) es el modelo subyacente de arquitectura para la industria, que difunde ampliamente el OLE. Éste es un sistema extensible abierto que apoya tecnológicamente el desarrollo para la creación de aplicaciones que puedan operar a través de otras aplicaciones, multiplataformas, y sobre todo para desarrollarlas como una colección de componentes que se instalan de manera independiente y que puedan comunicarse mediante interfaces basadas en el objeto.

Así, COM es la especificación mediante la cual se define el estándar binario para la implementación de objetos que es independiente del lenguaje de programación, y OLE provee funciones de interoperabilidad usando ORPC (objetos RPCs) que se extienden en la implementación de OLE Distribuido utilizando DCOM compatibles al DCE RPCs, aunque Microsoft no ha licenciado DCE desde el OSF. OLE provee la comunicación entre módulos de código binario bajo un sistema en ejecución. Este estándar habilita la comunicación entre dos aplicaciones mediante la interface basada en objetos.

COM distribuido (DCOM) es un producto de COM y el término "distribuido" se refiere a que es capaz de ejecutar servidores y clientes en diferentes procesos a través de Intranet o Internet. Por esto DCOM trabaja como COM, por ejemplo cuando un cliente solicita el registro donde el servidor se localiza y, en vez de indicar alguna localidad o registro de memoria sobre la máquina local, indica una dirección IP: 123.234.199.27. La diferencia básica entre COM y DCOM es que los procesos COM se ejecutan en la misma máquina en diferentes espacios de dirección, y los procesos DCOM se extienden a través de una red.

Las tres especificaciones primarias para OLE son:

a) Documentos OLE son un tipo de documentos compuestos que pueden incorporar los datos creados en cualquier aplicación OLE permitida o disponible. Mediante la vinculación de objetos las aplicaciones pueden unirse a objetos de datos dentro de otras aplicaciones. La incrustación de objetos es la capacidad para vincular un objeto dentro de otro documento sin mantener un nexo con el objeto fuente.

b) Controles OLE se emplean para la creación y gestión de controles personalizados. Esta especificación describe cómo crear o manipular cualquier tipo de control personalizado en un modelo OLE. La arquitectura de controles OLE permite trabajar con aplicaciones en ambientes múltiples de desarrollo y a través de diferentes plataformas tanto de equipos como de sistemas operativos.

c) La automatización OLE se usa para programación y lenguajes script. Esta especificación describe cómo crear un objeto o un "controlador" de automatización que pueda manejar objetos según un script o mensajes de algún tipo. Por otra parte permite a las aplicaciones exponer conjuntos de procedimientos y datos que operen dentro de estas y a través de otras aplicaciones para utilizar los servicios suministrados por otras aplicaciones habilitadas con la tecnología OLE; todo esto a través de objetos de automatización.

Componentes ActiveX en aplicaciones cliente/servidor

Debido a que ActiveX está basado sobre objetos compartidos, sus componentes tienen intrínsecamente definida la relación cliente/servidor, en donde el cliente solicita objetos y el servidor los provee. Sin embargo, la distinción entre clientes y servidores es difusa, porque el mismo componente puede ser ambos, y con frecuencia es a la vez un cliente y un servidor. Por esto podemos referirlos típicamente como "componentes ActiveX" más que como "clientes" y "servidores ActiveX". Dentro de una ejecución de contexto determinada, es importante la relación cliente/servidor.

El estándar de ActiveX está diseñado para permitir que los clientes se comuniquen con otros componentes sin importar dónde se ejecutan; puede ser dentro del mismo proceso en la misma máquina, o en una máquina diferente. Esto significa que hay un modelo simple de programación para todos los tipos de componentes ActiveX. Estos componentes en la relación cliente/servidor pueden estar contenidos en una computadora, o ejecutarse en diferentes computadoras conectadas por una red.

Un componente ActiveX se llama local cuando se ejecuta en la misma computadora, y remoto cuando se hace en una computadora diferente. Un componente es *in-process* o *out-of-process* respecto a su cliente; es *in-process* si se implementa dinámicamente como una librería de enlace dinámico (archivo *.dll), y por lo tanto se ejecuta en el mismo espacio de proceso como un consumidor de sus objetos, y *out-of-process* si el componente es un archivo exe, que se ejecuta en un espacio con dirección propia.

7.7 Integración de DCE y OLE/COM

La familia de tecnologías OLE incluye una gran variedad de cosas, que van desde documentos compuestos hasta interfaces estándar para acceder a una base de datos. Todas las tecnologías OLE se basan en el modelo COM. El protocolo de objetos RPC está altamente vinculado con el protocolo de red DCE RPC, y ocurre en ambos niveles, en el de especificación y en el de implementación.

Un procedimiento de llamada remota en red basada en COM (referido a ORPC) es equivalente a la llamada DCE (DCE RPC). COM especifica convenciones y provee servicios para definir y usar objetos, y al igual que CORBA, tiene una Interface de Lenguaje de Definición (IDL) para especificar interfaces de objetos, incluyendo servicios que permiten las instancias y la comunicación con otros. Microsoft usa DCE como base para la comunicación dentro de ActiveX. La mayoría de las firmas han intentado usar DCE RPC como un ambiente específico de protocolos dentro de la interoperabilidad de las especificaciones que provee CORBA. OLE (DCOM) ha seleccionado DCE RPC como su protocolo de comunicaciones, e idealmente esto permitirá la interoperabilidad entre OLE y DCE.

7.8 Middleware

Middleware se está usando como una manera de intercambiar la información por medio de componentes en ambientes distribuidos, por ejemplo se diseñaron para permitir el acceso remoto a bases de datos. Así constituye la infraestructura fundamental de la computación distribuida que tradicionalmente se ha usado en los proyectos *Enterprise* punto-a-punto cliente/servidor. Sin embargo este desarrollo desde el punto de vista de la tecnología de información (IT), el enfoque del *middleware* es complicado, difícil de usar y de mantener.

Los problemas asociados con su uso están intrínsecamente definidos con la manera como se emplee en la interacción de las aplicaciones. A fin de reducir el impacto de complejidad es necesario definir una arquitectura que las integre en componentes de aplicaciones distribuidas, y que además pueda apoyar todos los requerimientos de la computación distribuida, desde aplicaciones complejas de apoyo de decisión hasta robustos sistemas de aplicación de transacciones, y permita una fácil integración a cualquier sistema de *Enterprise* que opere en un ambiente de computacion distribuda (DECE-*Distributed Enterprise Computing Environment)*.

El DECE permite la integración de ambientes heterogéneos para organizar y automatizar el uso de la tecnología *middleware* ocultando la complejidad, ya que provee una capa protectora entre los programas de aplicación y la tecnología subyacente de la infraestructura. DECE contiene un conjunto de herramientas para automatizar el uso del *middleware* y un conjunto de servicios en tiempo de ejecución que aumenta la robustez de su ambiente.

DECE se basa en una arquitectura de componentes, que trata a cada elemento del sistema como un componente de software reutilizable o DC-objeto.

Estos DC-objetos están encapsulados usando una sola y bien definida interface abstracta que involucra al usuario con los lenguajes y herramientas a usarse en la implementación.

Los DC-objetos pueden fácilmente transportarse o duplicarse dentro de diferentes plataformas, y ser reutilizados para operar en red, junto con una variedad de configuraciones flexibles de objetos. El DECE permite construir sistemas a gran escala de información utilizando y ensamblando un conjunto de objetos pre-existentes mediante una arquitectura de aplicación lógica *3-tiered*.

OLEnterprise y su arquitectura

OLEnterprise es un producto DECE que provee un acceso dinámico a objetos OLE distribuidos para un acceso universal suministrando las herramientas y utililerías para la conección de las aplicaciones de escritorio con datos y servicios que residen remotamente en un ambiente *Enterprise* utilizando mecanismos OLE. También es un lenguaje y plataforma independiente de tecnologías de OLE distribuido, lo que facilita la automatización de objetos sin ninguna modificación en el código. Al utilizar el Microsoft RPC *OLEnterprise* de inmediato provee la potencia y los beneficios de OLE, permitiendo a las organizaciones construir y visualizar sofisticadas aplicaciones basadas en objetos con arquitectura *multi-tiered.*

La arquitectura *OLEnterprise* provee una alta posibilidad de crecimiento y los requerimientos de servicios son fáciles de mantener ya que se apoyan en la base DECE en una multi-infraestructura. Estos servicios permiten la localizacion de procesos y datos, la moderación en forma balanceada de la carga de trabajo en los servidores, la detección automática de las fallas en los servidores, la autorización de usuarios y la protección de datos.

OLEnterprise tiene tres componentes: el Explorador de Objetos (local y remoto), el Objeto Agente (servidor OLE de automatización) y el Controlador Remoto de Automatización OLE *(Object Factory)*.

El *Objeto Agente* es una aplicación *in-process* de servidor OLE de automatización que se ejecuta como una Biblioteca de Enlace Dinámico (DLL) y se localiza en la máquina cliente, proveyendo un acceso transparente y dinámico a cualquier objeto OLE o RPC. Así la aplicación cliente se ve simplemente como cualquier otro servidor OLE de automatización, con la capacidad de devolver un objeto solicitado a un servidor remoto. Si el objeto remoto es OLE la solicitud es devuelta al *Object Factory* que vuelve a emitir la solicitud al sistema remoto; en cambio si el objeto remoto es *enterprise*, la petición se convierte dinámicamente en una llamada RPC solicitada por el Objeto Agente y devuelta al servidor apropiado.

Un Objeto Agente se encuentra en la máquina cliente y tiene la capacidad de interpretar y procesar transparente y dinámicamente cualquier solicitud de automatización OLE y convertirla en una llamada RPC para devolverla a cualquier servidor. En ambos casos la comunicación se maneja por el mecanismo RPC, que se llama RPC nativo (NRPC). Cuando un componente se ejecuta en una plataforma PC, NRPC usa el RPC de Microsoft. Si el objeto remoto es una automatización OLE, la solicitud es emitida al *Object Factory,* la cual a su vez emite una solicitud de automatización OLE al sistema remoto. El Objeto Agente es el servidor de automatización en la máquina cliente, y el Microsoft RPC es el mecanismo de comunicación para OLE Remoto que sirve como interface entre el Objeto Agente y el *Object Factory.*

El *Object Factory* es un controlador OLE remoto (archivo *.EXE) que distribuye los servicios de automatización OLE (administrador de solicitudes remotas OLE) y se encuentra en el servidor que es responsable de procesar las solicitudes, a través de instancias de objetos que se localizan en clientes remotos.

El *Explorador de Objetos* tiene tres componentes principales: un visualizador para inspeccionar registros locales o remotos; un mecanismo de exportación de objetos OLE, para seleccionar servidores de automatización para el acceso remoto, y un mecanismo de importación para registrar servidores OLE remotos de automatización local. En la plataforma del servidor se generará y registrará localmente el servidor de automatización OLE usando sus utilerías estándar. La referencia para invocar al servidor de automatización OLE es un programa identificado (ProgId) que se asocia a una clase identificada (CLSID), y ambos se crean cuando el servidor de automatización OLE se agrega a la base de datos de registros locales.

8. Implementación de DCS

El diseño de configuraciones se considera, en síntesis, como el principal contenido y análisis estructural de los sistemas complejos de diferente naturaleza [Yos87, SK95A]. Los modelos y métodos descritos se aplicaron al desarrollo de la búsqueda de un prototipo DCS en Sistemas Flexibles de Manufactura (FMS) de diseño de configuración.

Un FMS se concibe como un diseño basado en objetos que tiene una estructura jerárquica con un conjunto de limitaciones relativas a diversos niveles de proyectos que los modelan [GK93]. La meta de la implementación del prototipo de búsqueda consiste en mostrar cómo los formalismos y métodos mencionados pueden usarse en ODC con base en un enfoque de multi-agentes. El DCS también comprende el apoyo para la especificación de requerimientos, control de consistencia de los proyectos y el diseño concurrente con componentes predefinidos de un esquema global.

En este tipo de diseño, el DM-agente selecciona y captura un fragmento del proyecto. Los fragmentos en ABCD pueden ser virtualmente de cualquier complejidad, además pueden ser parametrizados (paramétricamente diseñados), o no parametrizados (diseñados de manera conceptual).

El uso de fragmentos aprovecha la distribución del DM-proceso en forma concurrente sobre el proyecto que comparte el espacio de conocimiento, donde el DM-agente trabaja con un fragmento incrustado en el modelo distribuido de proyectos. El P-agente es el coordinador de proyectos que se basa en la cooperación y compartición de componentes sobre el mecanismo *3-tiered* para apoyar la interacción entre DM-agente y P-agente con múltiples aspectos de configuración paralela de proyectos y sus variantes.

Un P-agente coordina todos los intercambios de datos e información en la parte capturada del proyecto; modela y ejecuta concurrentemente el control de las propiedades de integridad y de consistencia más débil con base en el análisis de VDF, y modifica de manera concurrente las partes del proyecto.

El DCS se diseña para complementar y extender las capacidades de muchas de las herramientas de desarrollo actuales, sobre todo agregándoles la funcionalidad requerida para las aplicaciones *enterprise* con un amplio panorama en la computación distribuida. Las interfaces del DCS no se escriben en las aplicaciones, pero pueden fácilmente mejorarse para aprovechar las tecnología ODC si tienen esa disponibilidad.

La búsqueda del prototipo de DCS se ha estado desarrollando para el SO Windows95, utilizando Borland C++ 5.0 y la tecnología OLE de automatización [Bro93]. Esta última provee la estructura conceptual para crear, administrar y acceder mediante mecanismos basados en componentes de objetos. Además permite al DM-agente (OLE de cliente) exponer un conjunto de operaciones que el P-agente (OLE servidor) puede invocar o solicitar, y en cada operación de actualización los P-agentes usan el VDF para el análisis de la consistencia.

El DM-agente organiza una búsqueda que parte del fragmento capturado que es análogo al mecanismo que se basa sobre métricas de características deseables en los componentes.

Los módulos de agente desarrollados en C++ ofrecen su funcionalidad por medio de alguna interface. La ventaja de usar la tecnología OLE consiste en el modelo de múltiples hilos de control sobre el nivel de eventos, ya que los hilos son capaces de compartir datos comunes dentro de una plataforma determinada. Nosotros usamos la sincronización de hilos a fin de permitir la implementación del control concurrente.

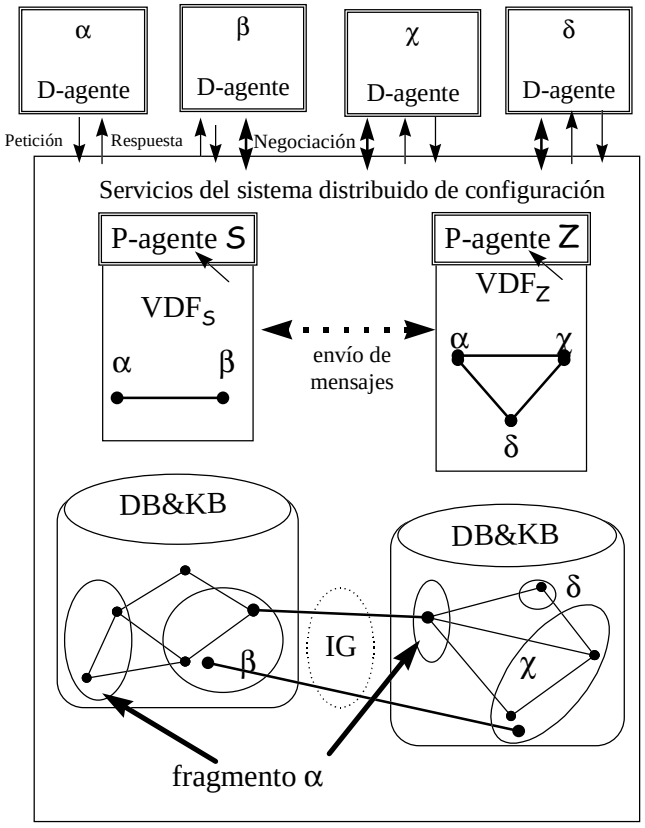

Figura 2. Modelo de la arquitectura DCS

La representación de mensaje en C++ tiene un API que oculta todos los detalles asociados con la comunicación de red para el usuario, y se usa para apoyar la interoperación entre aplicaciones que se ejecutan en Windows (UNIX en versión próxima).

Las acciones de los DM-agentes se llevan a cabo a través de mensajes KQML que incluyen restricciones de propiedades de objetos y otras definiciones primarias de usuario. Un mensaje es una expresión KQML en donde los argumentos son términos o sentencias KIF [Gen92]. Estos mensajes se encuentran en una lista de componentes agrupados entre paréntesis donde la primera palabra indica el tipo de mensaje. KQML soporta lo que nosotros podemos obtener de una metodología que permita la especificación, diseño e implementación del software cooperativo. En otras palabras, es una metodología que permite construir modelos para el software cooperativo que trabaje en un ambiente distribuido y de manera asíncrona.

El mensaje toma las expresiones KQML que se procesan como eventos semánticos (llamada a un servidor OLE). Un P-agente se utiliza para combinar datos y operaciones que se transforman en eventos; posteriormente el P-agente convierte la estructura de eventos a estructuras de datos que son aceptados por servidores OLE.

La especificación del objeto interface es similar al IDL (DCOM basado en OLE [DCOM96]), por lo que un DCS provee un número de servicios ODC, que permite a los clientes encontrar una referencia al P-agente en el tiempo de ejecución. Esto reduce la complejidad de implementación del cliente y le da flexibilidad. El administrador de objetos distribuidos (DCOM basado en OLE) provee transparencia de localización para las conexiones de DM y P-agentes. Los componentes de agentes OLE proveen una interface basada en mensajes que son independientes de estructuras de datos internos y algoritmos.

El P-agente (servidor OLE) toma las expresiones KQML y las transforma en eventos semánticos. El mecanismo de sincronización se usa para combinar operaciones y estructuras de datos con estructuras GT aceptadas o válidas para las operaciones GT. El proceso de operación del *OLEnterprise* es como se indica a continuación:

En la máquina cliente:

- El explorador de objetos se conecta a la máquina remota e importa la información del registro del servidor de automatización OLE.
- El controlador de automatización OLE crea un objeto OLE en el momento de la ejecución.
- En el momento de la ejecución OLE localiza el servidor de automatización usando el ProgId del objeto (que se encuentra en CLSID).
- El objeto agente usa el registro local para encontrar el correspondiente *Object Factory.*

En la máquina servidor:

- El explorador de objetos exporta el servidor de automatización OLE en el registro.
- El *Object Factory* recibe la solicitud y usa su CLSID al ejecutar el servidor de automatización OLE remoto en el momento de ejecución del Microsoft OLE.

Ahora el P-agente y el DM-agente pueden procesar invocaciones de métodos mediante el objeto agente del *Object Factory.*

Hay un API para las representaciones de notificaciones y mensajes en C++, que oculta todos los detalles asociados con la comunicación de red repecto al usuario. El API se usa para apoyar la interoperación entre aplicaciones que son ejecutables en Windows (y en UNIX en versión próxima).

Los primeros resultados son prometedores. Un proyecto colaborativo de tecnologías de configuración, como parte de la ingeniería concurrente fundamentada, llega a tener una mayor perspectiva en la toma de decisiones descentralizadas y distribuidas sobre diferentes dominios de problemas. Nosotros usaremos esta metodología y las experiencias que obtengamos con el fin de emplear el mejor enfoque en este desarrollo.

Referencias

[Arb96] "The IWIM Model for Coordination of Concurrent Activities", *First Int. Con. On Coordination Models, Languages and Applications*, Italy, 15-17 April, 1996, *LNCS,* 1062, Springer, pp. 34-56.

[AI88] ASIRELLI, P.; INVERARDI, P.; MUSTARO, A. "Improving Integrity Constraint Checking in Deductive Data Bases", *ISDT'88*, Ed. M. Gyssens, *LNCS,* N. 326, 1988.

[BF85] BOEHM, P.; FONIO, H. R., HABEL, A. "Amalgamation of Graph Transformations with Applications to Synchronization", *Mathematical Foundations of Software Development, LNCS,* V. 1, N. 185, 1985.

[Bro93] BROCKSCHMIDT, K. "Inside OLE", *Microsoft Press*, 1993.

[Csu93] CSUHAJ-VARJÚ, E. "On Grammars with Local and Global Context Conditions", *Int. J. Computer Math.*, 47, 17-27, 1993.

[Cut93] CUTKOSKY, M. R.; ENGELMORE, R. S.; FIKES, *et al.* "PACT: An Experiment in Integrating Concurrent Engineering Systems", *IEEE Computer*, 1993, 26(1):28-37.

[DCOM96] "Distributed Component Object Model Protocol DCOM/1.0", *Network Working Group, Internet-Draft*, 1996.

[EBH88] EHRIG, H.; BOEHM, P.; HUMBMMERT, U; LOWE, M. "Distributed Parallelism of Graph Transformation", *LNCS,* 314, pp. 1 -19, Berlin, 1988.

[Ehr90] EHRIG, H.; MAHR, B. "Fundamental of Algebraic Specification: Module Specification and Constraints, V.21 of EATCS", *Monographs of Theoretical Computer Science*, Springer, 1990.

[Ehr93] EHRIG, H.; Lowe, M. "Parallel and Distributed Derivation in Single Pushout Approach", *TCS,* 109: 123-143, 1993.

[Fin95] FININ, T.; FRITZSON, R.; MCKAY, D.; MCENTIRE. "KQML as an Agent Communication Language", *Proc. of 3th Int. Conf. on Information and Knowledge Management*, ACM Press, 1994.

[Gen92] GENESERETH, M. R.; FIKES, R. E. *et al.* "Knowledge Interchange Format, Version 3.0 Reference Manual, Technical Report Logic-92-1", Stanford University, 1992.

[Gen94] GENESERETH, M. R.; and KETCHPEL, S. P. "Software Agents", *Communications of the ACM*, 37(7):48-53, 1994.

[GG87] "Graph-Grammars and Their Application to Computer Science", Ed. H. Ehrig, Springer-Verlag, Berlin, *LNCS,* N. 291, 1987.

[Gym91] GYMTRASIEWICZ, P. J.; DURFEE, E. H.; WEHE, D. K. "A Decision-Theoretic Approach to Coordinationing Multiagent Interactions", *Proc. of the Twelfth Int. Con. on AI*, Sidney, Australia, 1991, pp. 62-68.

[GK93] GUSIKHIN, O. Y.; KOULINITCH, A. S. "Animated AI-based Simulation in Production Scheduling: Case Study. IFIP Transactions", B-11, KNOWHSEM'93, *Inter. Knowledge Based Hybrid System in Engineering Manufacturing Working Con. on Hungary,* IFIP, North-Holland, 1993.

[GT94] GAVRILA, I. S.; TREUR, J. A. "A Formal Model for Dynamics of Compositional Reasoning Systems", *Proc. 11th Eur. Con for the on AI, EC AI-94*, Wiley & Sons, pp. 307-311.

[Jen95] JENNINGS, N. R. "Controlling Cooperative Problem Solving in Industrial Multi-Agent Systems Using Joint Intentions", *AI,* 75(2), pp. 195- 240, 1995.

[Kle91] KLEIN, M. "Supporting Conflict Resolution in Cooperative Design Systems", *IEEE Transactions on System,* Man, Cybernetics, 21(5):1379-1390, 1991.

[Kou93] KOULINITCH, A. S. "Integrity Support for Concurrency Decision Making in Design", *Int. Con. on CAD/CAM, Robotics and Factories of the Future.,* St. Petersburg, 1993, pp. 243-251.

[Low93] LOWE, M.; KORFF, M.; WAGNER, A. "An Algebraic Framework for the Transformation of Attributed Graphs", *Term Graph Rewriting: Theory and Practice,* Chapter 14, pp. 185-199, 1993.

[Mac77] MACKWORTH, A. K. "Consistency in Networks of Relations", *Artificial Intelligence,* 1977, N. 8.

[Poi85] POIGNE, A. "Elements of Categorical Reasoning: Predicates and Coproducts (Colimits)", *Category Theory and Computer Programming,* Ed. D.Pitt, *LNCS,* N. 240, 1985.

[Rot91] ROTH, S.; SADEH, N.; SYCARA, S.; FOX, M. "Distributed Constraint Heuristic Search", *IEEE Transactions on System*, Man, Cybernetics, 21(5): 1446-1461, 1991.

[Ser91] SERRANO, D. "Constraint-Based Concurrent Design. Systems Automation: Research & Applications", 1991, N. 3, V. 1, pp. 217-230.

[SK95a] SMIRNOV, A. V.; KOULINITCH, A. S.; SHEREMETOV, L. B. "Knowledge-Based Configuration Design of Electronic Devices: A Case Study", *Workshop on Design Methodologies for Microelectronics,* Austria, Sept., 11-15, 1995.

[SK95c] SMIRNOV, A. V.; KOULINITCH, A. S.; SHEREMETOV, L. B.; ROMANOV, G. V.; TURBIN, P. A. "DESO: A Constraint-Based Environment Prototype for Cooperative Design of FMS", *Proc. of the III IASTED Inter. Conf.,* Cancún, México, June, 14-16, Anaheim-Zurich, 1995, pp. 384-387.

[Tae95] TAENTZER, G. "Towards Synchronous and Asynchronous Graph Transformation", *Special Issue of Foundamenta Informaticae,* 1995.

[Tan92] TANENBAUM, A.; KAASHOUEK, F.; BAL, H. "Parallel Programming Using Shared Objects and Broadcasting", *IEEE Computer,* 25 (8), 1992, pp.10-19.

[Wie92] WIEDERHOLD, G. "Mediators in the Architecture of Future Information Systems", *IEEE Computer*, 5(3):38-49, 1992.

[Wo94] WOOLDRIDGE, M.; JENNINGS, N. R. "Formalizing the Cooperative Problem Solving Process", *Proceedings of the 13th Inter. Workshop on Distributed Artificial Intelligence (IWDAI-94),* Lake Quinalt, WA., 1994, pp. 403-417.

[Yos87] YOSHIKAWA, H. "General Design Theory and Artificial Intelligence", *Artificial Intelligence in Manufacturing,* Elsevier Science Publishers (Nort-Holland), Amsterdam, 1987, pp. 35-61