UNIVERSIDAD DE GRANADA

Departamento de Lenguajes y Sistemas Informáticos

Programa de Doctorado en
Tecnologías de la Información y la Comunicación

TESIS DOCTORAL

Enfoque metodológico semántico basado en un modelo arquitectónico para el desarrollo de *groupware*

Mario Anzures García

Directores:
Dr. D. Miguel J. Hornos Barranco
Dra. Dª. Patricia Paderewski Rodríguez

Editor: Universidad de Granada. Tesis Doctorales
Autor: Mario Anzures García
ISBN: 978-84-9163-704-2
URI: http://hdl.handle.net/10481/48880

El doctorando / The *doctoral candidate* [**Mario Anzures García**] y los directores de la tesis / **and the thesis supervisor/s: [Dª. Patricia Paderewski Rodríguez y D. Miguel J. Hornos Barranco]**

Garantizamos, al firmar esta tesis doctoral, que el trabajo ha sido realizado por el doctorando bajo la dirección de los directores de la tesis y hasta donde nuestro conocimiento alcanza, en la realización del trabajo, se han respetado los derechos de otros autores a ser citados, cuando se han utilizado sus resultados o publicaciones.

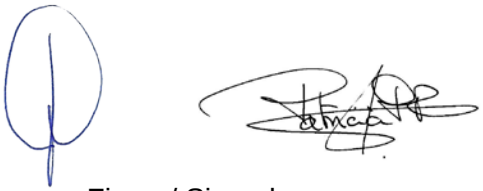
/

Guarantee, by signing this doctoral thesis, that the work has been done by the doctoral candidate under the direction of the thesis supervisor/s and, as far as our knowledge reaches, in the performance of the work, the rights of other authors to be cited (when their results or publications have been used) have been respected.

Lugar y fecha / **Place and date:**

Granada, España a 6 de septiembre de 2017

Director/es de la Tesis / ***Thesis supervisor/s;*** Doctorando / ***Doctoral candidate:***



Firma / Signed

Firma / Signed

Resumen

Actualmente, el desarrollo y mantenimiento de *groupware* resulta ser muy complejo, ya que implica aspectos multidisciplinarios en su construcción. Por lo general, el desarrollador debe manejar varios aspectos, como protocolos, conexiones de red, intercambio de recursos, distribución de procesos, visualización de la información, gestión de sesiones, etc.

En la literatura de CSCW (*Computer Supported Cooperative Work*, Trabajo Colaborativo Soportado por Computadora), se pueden identificar cuatro enfoques para desarrollar *groupware*: Ad hoc, basado en *Toolkits*, Componentes y Modelo Conceptual. Sin embargo, éstos carecen de modelos teóricos y/o computacionales que permitan especificar y desarrollar actividades grupales e interactivas de manera conceptual y/o formal para sustentar las necesidades propias del trabajo en grupo.

Por tanto, en la presente tesis doctoral se propone un enfoque metodológico semántico basado en un modelo arquitectónico, que sirva como guía para el análisis, diseño y desarrollo de *groupware*, de una manera formal y explícita.

El enfoque metodológico semántico se basa en una ontología *workflow*, que suministra un modelo formal sobre el conocimiento de cómo las entidades deben ser usadas y combinadas para controlar la ejecución de un conjunto de pasos ordenados para desarrollar *groupware.* A su vez, la ontología *workflow* es creada a partir del estilo arquitectónico Modelo-Vista-Controlador (MVC), que contiene aquellos elementos (entidades) que permiten este desarrollo. El componente modelo que sustenta el MVC es representado por la ontología de la política de manejo de sesión, que permite ajustar la estructura organizacional del grupo a los cambios del mismo y a los diferentes estilos de trabajo de varios grupos. Además, se emplean tablas de especificación que sirven de guía en el análisis, diseño e implementación. El enfoque metodológico semántico se ha validado a través de una serie de proyectos académicos, demostrando ser de gran ayuda para los desarrolladores de *groupware.*

Finalmente, se presenta una metodología de evaluación que se ha aplicado al enfoque metodológico semántico propuesto, mostrando su usabilidad y utilidad para el desarrollo de este tipo de aplicaciones.

Contenido

Índice de Figuras

Índice de Tablas

1. Introducción

En este capítulo introductorio se proporcionan los antecedentes fundamentales en el área de investigación de CSCW (*Computer Suported Colaborative Work*, en español trabajo colaborativo soportado por computadora) y se presenta la motivación, objetivos y el enfoque metodológico semántico para desarrollar *groupware* que se presenta en este trabajo de tesis. Finalmente, se describe la organización de esta memoria de tesis.

1.1. Trabajo colaborativo soportado por computadora

CSCW comenzó como un esfuerzo de los expertos en computación por aprender de economistas, psicólogos, sociólogos y de todos aquellos que pudiesen aportar algo para entender mejor las actividades de grupo; llegando a ser un área de investigación en la que se comparten experiencias, se discuten diferentes posibilidades y restricciones tecnológicas con respecto al trabajo en grupo [Grudin 94].

El concepto de CSCW se acuño en el año 1984, gracias a una iniciativa de la DEC (*Digital Equipment Corporation*) y el MIT (*Massachussets Institute of Technology*), que convocó a un conjunto de desarrolladores e investigadores de diferentes áreas con la finalidad de examinar el rol de la tecnología en el entorno de trabajo.

CSCW es el estudio de sistemas organizacionales que integran actividades de comunicación y procesamiento de información, consistiendo en analizar cómo trabajan los grupos y descubrir cómo las tecnologías pueden ayudarlos [Beaudouin 99].

El término *groupware,* que es una contracción de las palabras *group* y *software*, fue acuñado por Peter y Truddy Johnson-Lenz en 1978. *Groupware* (también conocido como *aplicación colaborativa*) es un sistema basado en computadora que asiste a un grupo de personas implicadas en un objetivo (o meta) común, proporcionando una interfaz para un entorno compartido [Ellis 91], y que permite la comunicación, colaboración y coordinación del grupo, sin importar la situación geográfica, cultural y social de los miembros del mismo.

Para algunos autores los términos CSCW y *groupware* son sinónimos o similares. Sin embargo, CSCW es la disciplina que analiza el trabajo en grupo y *groupware* es la tecnología que lo permite [Martínez 05].

A lo largo de su historia, CSCW ha intentado soportar a un grupo de personas con un interés común para organizar y hacer su trabajo, tanto para escenarios profesionales (representando reuniones planeadas o de negocios, soportar tareas de organización, etc.) como de ocio (reuniones espontáneas con fines de ocio de comunidades de personas, con el objetivo de compartir o intercambiar experiencias) [Araújo 97].

Tanto las organizaciones como las comunidades aportan un lugar de reunión interactivo, en el cual los miembros (o usuarios) y recursos pueden estar geográficamente distribuidos, funcionando como una unidad coherente y permitiendo el acceso (síncrono y/o asíncrono) a los recursos compartidos. Además, ambas están reguladas por un conjunto de reglas que determinan su estructura, donde dicha estructura es el medio y el resultado de la conducta organizacional comunitaria o de grupo. Es por ello que CSCW se ha orientado hacia la creación de notaciones computacionales apropiadas para modelar y ejecutar la interacción (centrada llevar a cabo trabajo profesional o actividades de ocio) del grupo [Borghoff 98]. Esta interacción depende del contexto en el que se organiza y desarrolla el trabajo, así como de la intención de los usuarios del grupo. De esta manera, el desarrollo de aplicaciones colaborativas o *groupware* debería estar fundamentada en la especificación y modelado de estructuras organizacionales acordes a las necesidades y estilos dinámicos de cada grupo.

Por consiguiente, en este trabajo de tesis se propone un enfoque metodológico semántico basado en un modelo arquitectónico [Anzures 09a, Anzures 11a, Anzures 11b, Anzures 14a, Anzures 15b] para el desarrollo de *groupware*; que guíe este proceso de manera formal, precisa y flexible, integrando modelos ontológicos y arquitectónicos sustentados en especificar y representar la política de manejo de sesión, que gestiona la estructura organizacional del grupo, de tal manera que la política pueda ser modificada con

respecto a los estilos organizacionales y necesidades del grupo, ya sea para sustentar la interacción de reuniones centradas en trabajo profesional o simplemente de ocio. Además, se plantea una metodología de evaluación de la utilidad y usabilidad basada en un modelo lingüístico difuso [Heradioa 13, Sánchez-Gálvez 15], que permite evaluar y determinar la calidad del enfoque metodológico semántico. La metodología de evaluación parte de conjuntos de números difusos que permiten determinar la calidad de un entorno; en este trabajo, se ha adecuado para evaluar la calidad del enfoque metodológico propuesto.

1.2. Motivación y objetivos

En esta sección se presenta la motivación que ha conducido a este trabajo de tesis, y los objetivos, tanto el general como los específicos, que han permitido el desarrollo de la misma.

1.2.1. Motivación

Actualmente, el desarrollo y mantenimiento de *groupware* resulta ser muy complejo, ya que implica aspectos multidisciplinarios. Así, por regla general, el desarrollador debe manejar y controlar diferentes aspectos, tales como: protocolos, conexiones de red, intercambio de recursos, distribución de procesos, visualización de la información, gestión de sesiones, etc. Aunque existen trabajos centrados en el proceso de desarrollo de *groupware*, éstos se clasifican principalmente en cuatro enfoques, que se listan a continuación [Sosa 06, Molina 06, Molina 09, Rodríguez 12]:

- **Ad hoc.** Se construye conforme al problema específico a resolver.
- ***Toolkits.*** Se crean utilizando funciones y APIs (*Application Programming Interfaces*). Ejemplos de estos *toolkits* son: Groupkit [Roseman 92, Roseman 96a], COAST (*COoperative Application Systems Toolkit*) [Schuckmann 96], Habanero [Chabert 98], JSDT (*Java Shared Data Toolkit*) [Burridge 99] y AGILO [Guicking 05];
- **Componentes.** Se concibe manipulando elementos predefinidos, como por ejemplo: ANTS [García 03], EVOLVE [Stiemerling 99a, Stiemerling 00], DACIA (*Dynamic Adjustment of Component InterActions*) [Litiu 99, Litiu 00], CoCoWare (*Collaborative Component softWare*) [Slagter 01] y DreamTeam [Roth 00];
- **Modelos.** Se generan mediante la especificación de los elementos requeridos. Siguiendo este enfoque se han hecho bastantes propuestas, entre las que destacan las siguientes: Teoría de la Coordinación [Crowston 06], Cognición Distribuida [Hollan 00], Análisis de Tarea [Van Welie 98], Teoría de la Actividad [Kuutti 91], AMENITIES (*A MEthodology for aNalysis and desIgn of cooperaTIve systEmS*) [Garrido 03], CIAM (*Collaborative Interactive Applications Methodology*) [Molina 07, Molina 09], TOUCHE (*Task-Oriented and User Centred process model for developing interfaces for human-Computer-Human Environments*) [Penichet 07a, Penichet 08], MOLLICC (*Modeling Language for Interaction as Conversation, Collaborative*) [De Souza15], Meta-modelo Genérico [Jeners 12], Meta-Modelo de Entornos Virtuales [Jeners 13], Modelo de Clasificación Socio-Técnico [Cruz 12] y MoCA (*Model of Coordinated Action*) [Lee 15]. Estos, por un lado, carecen de un modelo formal y explícito, así como de la flexibilidad necesaria para adaptar la estructura organizacional del grupo a las necesidades cambiantes de la organización o comunidad. Por otro, no presentan un conjunto de líneas guía que simplifiquen el proceso de construcción de *groupware,* guiando al desarrollador en cada etapa.

En los últimos años, el desarrollo de groupware se ha centrado en el cuarto enfoque; sin embargo, algunos autores [Weiseth 06, Penichet 07b, Giraldo 08, Verginadis 10, Grudin 12, Rodríguez 12, Anzures 16b] han identificado limitaciones en el mismo; tal como:

A. Falta de modelos teóricos y computacionales que permitan especificar las actividades grupales.
B. Dificultades para abordar el modelado integral de los aspectos interactivos.
C. Carencia de artefactos de especificación conceptual para modelar tareas colaborativas.
D. Nuevas herramientas colaborativas sustentadas en investigaciones que, por una parte, estudien el contexto actual y los efectos significativos en la sociedad de implementaciones exitosas, y

por otra, creen o validen estructuras ontológicas de procesos de colaboración para especificar la interacción del grupo.

Falta de un modelado formal de estructuras organizacionales acordes a las necesidades y estilos dinámicos del grupo, que soporten la creación de aplicaciones colaborativas.

Consecuentemente, la principal motivación de este trabajo es que las propuestas (analizadas en la literatura de CSCW) para el proceso de desarrollo de *groupware,* carecen de un modelo formal y explícito, así como de la flexibilidad necesaria para adaptar la estructura organizacional del grupo a las necesidades cambiantes de la organización o comunidad. Además, de que éstas no presentan un conjunto de directrices que simplifiquen el proceso de construcción de *groupware*, guiando al desarrollador en cada etapa. En consecuencia, en este trabajo de investigación se propone un enfoque metodológico semántico basado en una ontología *workflow* del proceso de negocio de la estructura organizacional del grupo, que guía a los desarrolladores en dicho proceso [Anzures 14b, Anzures 16b, Anzures 17a, Anzures 17b].

1.2.2. Objetivos

El objetivo general es proponer una metodología con un enfoque semántico, que soporte y guíe la construcción de aplicaciones colaborativas.

Los siguientes objetivos específicos de este trabajo de investigación son:

- Analizar en la literatura relacionada con CSCW diversos enfoques centrados en el desarrollo de *groupware*, para establecer el enfoque más apropiado para este tipo de desarrollo, así como los elementos que permiten el mismo.
- Revisar la literatura de CSCW concerniente a modelos arquitectónicos que sirven como sustento para la construcción de *groupware,* de tal manera que sea posible determinar el estilo y modelo arquitectónico adecuado para esta construcción.
- Estudiar los sistemas basados en políticas, determinando los elementos, características, ventajas y desventajas que los caracterizan, para establecer el manejo de sesión de *groupware* basado en políticas.
- Formular el modelo ontológico de la política de manejo de sesión, que permita representar y adaptar formal y explícitamente la estructura organizacional del grupo; para adecuarla a las necesidades, dinámicas y estilos de trabajo del grupo.
- Proponer un patrón arquitectónico MVC, que establezca el conjunto de elementos o ítems que soporten el desarrollo de una aplicación colaborativa; así como también permita definir una serie de directrices para dicho desarrollo.
- Establecer un conjunto de plantillas, que simplifiquen la definición y seguimiento de la metodología semántica propuesta para desarrollar *groupware*.
- Plantear una ontología *workflow* con la finalidad de modelar y combinar de manera formal las entidades y el conjunto de pasos ordenados a seguir para construir una aplicación colaborativa.
- Proponer una metodología basada en la ontología *workflow*, para desarrollar aplicaciones colaborativas.
- Plantear una metodología para evaluar la usabilidad del enfoque metodológico semántico propuesto.

1.3. Propuesta para desarrollar *groupware*

En este trabajo de tesis, se propone un enfoque metodológico semántico basado en un modelo arquitectónico para el desarrollo de *groupware*. Por una parte, se pretende que sirva como guía precisa y flexible para este tipo de desarrollo; y por otra, que provea tanto una especificación formal como mecanismos para la adaptación de la aplicación y del grupo a diferentes escenarios colaborativos. Además, el enfoque metodológico se sustenta en los principios del proceso de desarrollo del software y en el carácter interdisciplinario de CSCW, ajustándose tanto a los requisitos generales de una organización o comunidad como a los requisitos particulares de sus individuos.

El desarrollo de este tipo de enfoque metodológico semántico tiene que hacerse siempre centrado en el desarrollador, para obtener un producto de calidad, por lo que es de vital importancia que el enfoque tenga un alto grado de usabilidad y utilidad. Por ello, se aplica y adecua una metodología [Heradioa 13, Sánchez-Gálvez 15] basada en un modelo lingüístico difuso para evaluar la usabilidad y utilidad del enfoque semántico.

El enfoque metodológico se representa a través de la ontología *workflow* de la estructura organizacional de grupo y de la ontología que sustenta el enfoque metodológico [Anzures 16b, Anzures 17a]. La ontología ha sido utilizada en diversas disciplinas, originalmente en el dominio de la filosofía, donde se trata de una teoría sobre la naturaleza de la existencia, los tipos y estructuras de objetos, propiedades, eventos, procesos y relaciones relativas a cada parte de la realidad [Gómez-Pérez 04]. Posteriormente, en la comunidad de Inteligencia Artificial, siendo un concepto que permite definir formalmente las relaciones entre los términos de un dominio específico. Esto es posible porque las ontologías se definen independientemente de los datos y reflejan una comprensión común de la semántica de un dominio. Existen muchas razones por las que las ontologías tienen una amplia gama de aplicaciones, entre las que destacan su relevancia para: aportar una terminología común sobre un dominio, necesaria para la comunicación entre las personas y las organizaciones; proporcionar la base para la interoperabilidad entre sistemas; y permitir una representación estructurada y formal de un dominio de conocimiento específico, que ayuda a eliminar la ambigüedad y la redundancia, al detectar errores y facilitar el razonamiento automático. En este trabajo:

- El modelo ontológico especifica la estructura organizacional del grupo basada en política, en particular, en la política de manejo de sesión, ya que las ontologías proporcionan un vocabulario común y compartido, a través de una descripción semántica en términos de conceptos, relaciones y axiomas, para describir este dominio de una manera formal. Esto permite la flexibilidad de adaptación a la dinámica inherente del trabajo en grupo, con tan sólo cambiar los conceptos, relaciones, axiomas o instancias de la ontología. Así, no sólo se puede modelar una aplicación, sino una variedad de ellas, para lo que basta modificar esos elementos de la ontología.
- La ontología *workflow* representa el conjunto y orden de pasos que componen la metodología para desarrollar una aplicación colaborativa. Está basado en la ontología de la estructura organizacional, suministrando una estructura semántica con suficiente expresividad, formalidad y flexibilidad, tanto para automatizar la secuencia de tareas como para controlar su flujo de ejecución.

Un *workflow* es una tecnología que define la automatización computarizada de un proceso de negocio, parcialmente o en su totalidad [Fischer 04]. Un proceso puede ser considerado como el conjunto de actividades realizadas por diferentes entidades y su orden de ejecución a través de diferentes constructores (por ejemplo, secuencia, elección, paralelismo, concurrencia y sincronización), permitiendo el control del flujo de ejecución, así como la definición de los diferentes elementos que participan en el flujo del proceso, tales como documentos, roles, intercambio de información y herramientas necesarias para la realización de cada paso. Generalmente, cada uno de estos procesos se describe por medio de algunas características básicas, como la información de entrada, la información de salida y las transformaciones realizadas por una persona o una máquina desempeñando un rol específico [Marinescu 02].

El modelo arquitectónico utilizado en la propuesta es un MVC (*Model-View-Controller*) [Goldberg 84] personalizado. MVC fue diseñado originalmente por Reenskaug Trygve en Smalltalk-80, donde fue inicialmente llamado Modelo-Vista-Controlador-Editor [Addy 15]. MVC es un estilo arquitectónico creado para reducir el costo y mejorar la calidad del software en el paradigma orientado a objetos, ya que facilita la modularidad, al encapsular los detalles de la implementación detrás de interfaces estables, reduciendo el esfuerzo necesario para entender y mantener el software existente. MVC se ha aplicado exitosamente en diversas áreas, un ejemplo de ello es el desarrollo Web; de esta manera, se ha convertido en un destacado patrón de diseño, ya que describe soluciones que han sido probadas, dando excelentes resultados. MVC separa los datos de negocio (Modelos) de las interfaces de usuario (Vistas), usando un tercer componente (Controlador) como mediador, que tradicionalmente maneja la lógica de la aplicación y la entrada del usuario [Goldberg 84]. Esta separación permite hacer una clara división entre los *objetos de dominio*, que modelan conceptos en el mundo real, y los *objetos de presentación*, que se visualizan en la pantalla del usuario. Por tanto, MVC establece un conjunto de recomendaciones que simplifican el proceso de desarrollo

de *groupware*, a partir de los requisitos indispensables para ello, los cuales se organizan en tres niveles de abstracción o capas, constituyendo de esta manera los componentes de las mismas. A su vez, cada componente se traslada a *templates* (plantillas), para suministrar una guía precisa y flexible en el análisis, diseño e implementación de este tipo de aplicaciones. Estos componentes se han concretado tras el estudio del estado del arte en el desarrollo de *groupware*, tanto de enfoques (cuatro en particular: *ad hoc*, *toolkits,* basados en componentes y modelos conceptuales) como de arquitecturas.

En este trabajo, se propone implementar cada tarea junto con su mecanismo de notificación y/o concurrencia (en caso de que sea necesario) del enfoque metodológico semántico propuesto mediante la tecnología de servicios Web [Anzures 06c]. Ya que esta tecnología facilita la reusabilidad y adaptabilidad, es más fácil reusar o adaptar una parte que una aplicación completa. De esta manera, cada tarea puede ser reutilizada en otro *groupware* o adaptada según las necesidades del grupo. Cuando se decide implementar soluciones de negocio basándose en la tecnología de Servicios Web, la palabra clave es Servicio [Bloomberg 06, Heffner 07]. Un servicio es una unidad de trabajo que se realiza por un proveedor para obtener un resultado final, el cual es requerido, utilizado o ejecutado por un consumidor [Erl 15]. Tanto el proveedor como el consumidor del servicio son representados por elementos de software que se ejecutan al lado de cada actor que participa en la comunicación [He 03].

1.4. Organización de la memoria

El presente trabajo de investigación se ha estructurado de la siguiente forma:

En el Capítulo 2 se presenta un estudio del estado del arte en el desarrollo de *groupware*. Este capítulo muestra los aspectos fundamentales del desarrollo de *groupware*, así como una revisión y análisis de los cuatro enfoques principales para tal desarrollo, como son: *ad hoc*, *toolkits,* componentes y modelos conceptuales.

En el capítulo 3 se presentan en primer lugar los conceptos fundamentales que caracterizan a las arquitecturas de software, para después realizar un estudio de los estilos arquitectónicos más relevantes para el desarrollo de *groupware,* centrándose en el Modelo-Vista-Controlador y analizando herramientas *groupware* basadas en los diferentes estilos revisados.

En el capítulo 4 se ofrece una revisión del concepto y uso de las ontologías, así como de los tipos de ontologías, los principales lenguajes y las herramientas existentes que permiten la definición y descripción de las mismas.

En el Capítulo 5 se presenta el concepto, clasificación y uso de *workflows*, así como su especificación con una ontología.

En el Capítulo 6 se presenta y describe la propuesta de un enfoque metodológico semántico basado en un modelo arquitectónico para el desarrollo de *groupware*. Partiendo de la ontología que representa la política de manejo de sesión, se da paso al estilo arquitectónico MVC para desarrollar *groupware*. Con base a este último, se establece la ontología *workflow* que sustenta el enfoque metodológico semántico, así como las etapas que lo constituyen y los modelos que de éste se derivan.

En el Capítulo 7 se describe y aplica la metodología empleada para evaluar la utilidad y usabilidad del enfoque metodológico semántico presentado en el capítulo anterior.

En el Capítulo 8 se resumen las aportaciones del presente trabajo de investigación y se describen los posibles trabajos futuros que pueden emprenderse a partir de la situación actual.

2. Aplicaciones Colaborativas

En el presente capítulo se puntualizan los aspectos esenciales que describen a *groupware*. Consecutivamente se presenta un análisis sobre las propuestas de *toolkits,* componentes y modelos conceptuales para la creación de aplicaciones colaborativas. Finalmente, se muestran las conclusiones del capítulo.

2.1. Aspectos fundamentales de groupware

Las aplicaciones colaborativas son sistemas complejos que requieren mecanismos para el manejo de la interacción del grupo tanto en organizaciones como en comunidades, de tal forma que puedan suministrar la comunicación, colaboración y coordinación para soportar la naturaleza dinámica y las necesidades cambiantes del trabajo de grupo. En la literatura de CSCW se consideran la comunicación, colaboración y coordinación, como los tres aspectos que caracterizan a *groupware* [Ellis 94, Gea 00*,* Fucks 04] los cuales serán descritos a continuación.

2.2.1. La comunicación

Es un aspecto fundamental en toda actividad humana, que permite la interacción entre las personas que están involucradas en ella. Los elementos que permiten caracterizar este proceso son los usuarios, la información que se comparte y el medio o artefacto utilizado para tal efecto (ver Figura 1) [Gea 00]. Cuando se introduce un artefacto (la computadora) para dar soporte a la comunicación, pueden ocasionarse posibles interferencias; sin embargo, el uso de la computadora como soporte de la comunicación ofrece grandes ventajas, debido a la variedad de medios que ofrece, como texto, imágenes, audio, video, etc. El proceso de comunicación está soportado por un modelo de distribución [Phillips 99] que define qué partes de *groupware* se ejecutan sobre un servidor central, qué partes se ejecutan en sitios descentralizados y cómo los sitios se enlazan lógicamente unos con otros. Las dos alternativas más comunes para las aplicaciones colaborativas son las arquitecturas Cliente-Servidor y Peer-to-Peer.

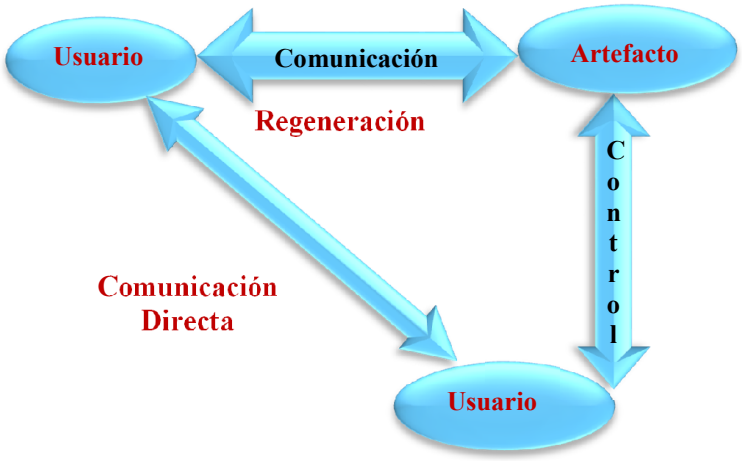


Figura 1. Comunicación entre usuarios a través de un artefacto [Gea 00].

Mientras la arquitectura Cliente-Servidor reduce la complejidad y simplifica la consistencia, Peer-to-Peer evita cuellos de botella y puntos de fallo simples en el servidor [Phillips 99].

También se puede ver el proceso de comunicación en su dimensión temporal, distinguiendo la comunicación síncrona (mismo tiempo) de la asíncrona (diferente tiempo). Por ello, comúnmente se habla de aplicaciones colaborativas síncronas y aplicaciones colaborativas asíncronas; así, las podemos clasificar atendiendo a la dimensión temporal y al medio de comunicación empleado (véase la Figura 2). En esta figura se pueden observar tres tipos básicos de soporte a la comunicación: texto, audio e imagen, además de que algunos medios pueden utilizar varios soportes.

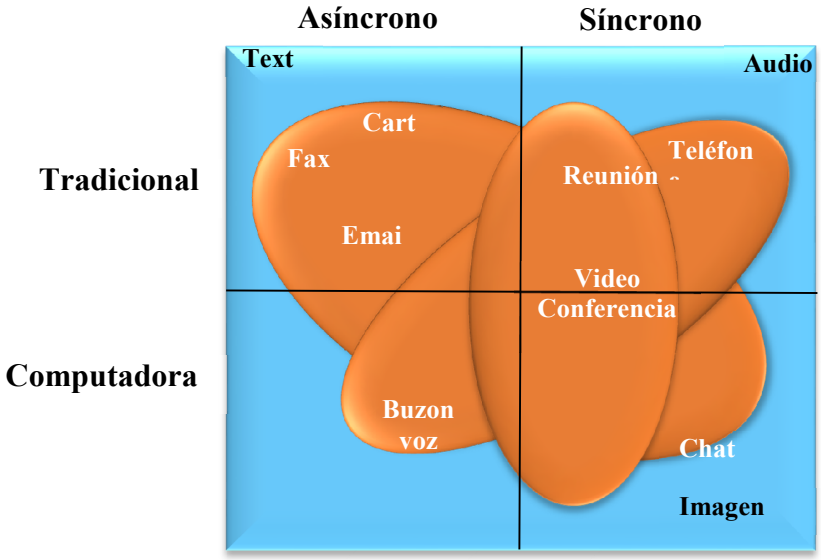


Figura 2. Tipos de comunicación según medio y modo [Gea 00].

La comunicación desde la perspectiva de CSCW permite que los usuarios intercambien información usando la computadora o algún dispositivo móvil como medio para comunicarse, a través de protocolos utilizados en un modelo de distribución (cliente-servidor o *peer-to-peer —p2p*), suministrando tanto comunicación síncrona como asíncrona.

2.2.2. La colaboración

Es la piedra angular de la actividad de grupo. La colaboración supone un paso adelante en el proceso de comunicación, ya que denota un grado de participación mayor de los usuarios para la consecución de un determinado objetivo, mediante su participación expresa en actividades de grupo. La colaboración efectiva demanda que las personas compartan información, de tal forma que en la aplicación se debe considerar la información que se va a compartir, así como los mecanismos de acceso a la misma. En este proceso se tienen los siguientes elementos [Gea 00]:

- *Actores*: Son los usuarios del grupo de trabajo. Los actores pueden tener diferentes responsabilidades, intenciones o modos de acceso, así que debemos identificar sus características y formas de trabajo.
- *Roles:* Determinan un patrón de comportamiento asignado a cada usuario, condicionando su actividad dentro del sistema. Los usuarios pueden desempeñar más de un rol en la organización, en función de las necesidades y distribución de responsabilidades dentro del grupo.
- *Tareas*: Son el conjunto de actividades encaminadas a la consecución de un objetivo determinado. En cada tarea se puede analizar el grado de participación de los actores, para obtener el mayor beneficio con el mínimo esfuerzo.
- *Recursos*: Son datos (documentos, repositorios, etc.) u objetos (artefactos, sistema, etc.) que se comparten para obtener una colaboración efectiva, de tal forma que se eviten inconsistencias o falta de robustez en la aplicación.

La colaboración se produce a través de la interacción de los usuarios involucrados en tareas conjuntas, utilizando los recursos de que se dispone para tal efecto; así que, además de requerir diferentes niveles de compartición, también se requiere notificación explicita a los usuarios, cuando sea necesario [Gea 00]. Por lo tanto, es importante realizar un diseño basado en tareas, ya que permite un mayor nivel de abstracción, centrándose en los aspectos que son relevantes a los usuarios.

La colaboración enfatiza un grado de participación mayor que la comunicación, facilitando la compartición de información entre usuarios con características únicas, pero con formas de trabajo especificadas por los roles o funciones del actor, que establecen derechos o permisos sobre qué tareas pueden realizar estos usuarios utilizando los recursos compartidos [Gea 00]. Se debe notificar o informar al resto del grupo o a determinados usuarios de las tareas que hace cada uno de ellos utilizando los recursos compartidos, con el fin de simplificar el proceso de la colaboración.

2.2.3. La coordinación

Gestiona las dependencias entre actividades realizadas en el grupo para alcanzar un objetivo. La coordinación requiere de una planificación y sincronización de las actividades efectuadas por los miembros del grupo, por lo que es necesario la identificación, la distribución y delegación de responsabilidades [Gea 00]. Es decir, se deben definir dependencias temporales entre actividades y asignar tareas a los usuarios conforme a los roles aceptados por la comunidad. La coordinación es un proceso indispensable cuando varios actores están participando en una misma tarea (véase la Figura 3). La coordinación es un aspecto primordial en las aplicaciones colaborativas, porque proporciona mecanismos para atenuar las condiciones de competencia y garantizar el uso mutuamente exclusivo de los recursos compartidos, reduciendo la probabilidad de conflictos y evitando la inconsistencia de los datos compartidos. Estos mecanismos, denominados *mecanismos de concurrencia*, deben ser lo suficientemente eficientes para controlar y gestionar el acceso concurrente a los servicios por diferentes clientes, así como el uso combinado de componentes de aplicación asíncronos y síncronos, sin degradar los requisitos no funcionales (por ejemplo, la adaptabilidad). Existen dos enfoques principales para garantizar la consistencia: 1) evitar conflictos mediante el bloqueo de los datos antes de la modificación, y 2) detectar y resolver los conflictos. Los mecanismos de bloqueo comunes incluyen mecanismos de exclusión mutua y semáforos. [Dommel 97]. Los mecanismos de detección y resolución de conflictos comunes incluyen transacciones y protocolos para la actualización de datos compartidos. [Dommel 97].

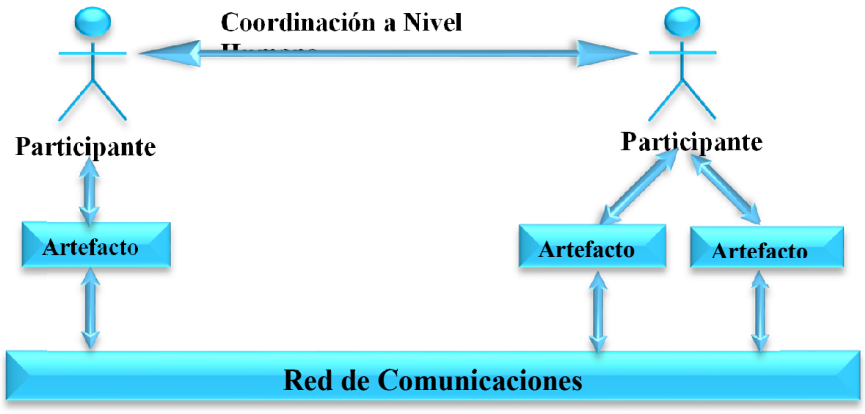


Figura 3. Coordinación [Gea 00].

El bloqueo es adecuado si, por ejemplo, los cambios son difíciles de detectar o complicados de resolver. Sin embargo, el bloqueo reduce el rendimiento, ya que la aplicación tiene que esperar a que pase el bloqueo antes de poder continuar. Por todo esto, se han implementado diversos mecanismos para el control de la concurrencia, como: *floor control* (asignación de turno), transacciones atómicas, bloqueos, versionado, paso de *tokens*, sistemas de votación, etc. [Dommel 97, Ellis 89].

Por ende, la coordinación establece principalmente mecanismos de concurrencia para sincronizar la colaboración entre actores cuando realizan tareas que requieran de recursos compartidos, suministrando el acceso en exclusión mutua a los mismos, para evitar inconsistencias en la modificación de la información.

Por otra parte, el espacio donde se desenvuelve el grupo, que permite explorar propiedades espaciales, reforzando los aspectos de comunicación, colaboración y coordinación entre los integrantes del mismo [Benford 98], es el espacio de trabajo compartido.

2.2.4. Espacio de trabajo compartido

En CSCW se han utilizado diferentes términos para el espacio de trabajo compartido, tales como: *sesiones* [Edwards 94], *conferencias* [Venkat 91], *conversaciones* [Kaplan 92], *espacios* [Beaudouin 94], *locales* [Fitzpatrick 95], *lugares* [Fitzpatrick 96], *colaboratorios* [Wulf 93], *comunidades* [Preece 00], *equipos* [O'Leary 07] y *e-ciencia* o *e-investigación* [Wang 13]. Estos términos denotan un conjunto de individuos, geográficamente distribuidos, que comparten un interés común para realizar tareas colaborativamente [Anzures 07c]. En este trabajo se utiliza el concepto de sesión para denotar el espacio de trabajo compartido, que permite especificar y estructurar las interacciones entre un grupo de usuarios que trabajan conjuntamente con recursos compartidos, especificando modos de colaboración.

Las aplicaciones colaborativas comúnmente suministran un mecanismo para controlar y gestionar las sesiones, denominado *manejador de sesión*, que permite definir sesiones por medio de una interfaz de usuario, a través de la cual los usuarios establecen la conexión, se unen a, salen de, detienen y terminan sesiones. Estos mecanismos generalmente sólo especifican una *política de manejo de sesión,* que determina la organización del grupo para realizar el trabajo conjuntamente. En la mayoría de las ocasiones, el proceso de establecer la sesión y el proceso que lleva a cabo la política de manejo de sesión dependen uno de otro, lo que hace que las sesiones sean inflexibles y difíciles de ajustar a diferentes formas de organización o necesidades del trabajo en grupo. Sin embargo, lo ideal es que la política se adapte tanto a las necesidades del trabajo en grupo como a los estilos organizacionales de diferentes grupos.

El trabajo en grupo se organiza definiendo políticas, que enumeran un conjunto de reglas (o protocolos) de comportamiento aceptadas dentro del grupo como parte del proceso de interacción [Gea 00]. Las reglas especifican: cómo los nuevos usuarios pueden unirse a sesiones (*protocolos de registro*); quiénes y en qué momento realizan las tareas del grupo (*protocolos de trabajo*); y cómo los usuarios deben terminar en el entorno colaborativo (*protocolos de terminación*). A la vez, los protocolos de trabajo establecen estrategias (o planes de actuación) para la obtención de la meta en común. La estrategia se fija de acuerdo a la meta común a alcanzar, ya que a partir de ésta se planea en qué momento participarán los usuarios y realizarán las tareas, así como los roles que debería desempeñar un usuario para llevar a cabo una tarea concreta y los recursos compartidos que se utilizarán para la misma. Esta planificación permite prever en qué momento se hará uso del mecanismo de concurrencia y gestionar la dinámica del trabajo en grupo, que modifica la composición, los objetivos a corto o medio plazo y la propia interacción del grupo.

En CSCW se han utilizado una variedad políticas de manejo de sesión [Gea 00], entre las cuales se pueden mencionar:

- **Puerta abierta:** Adecuada para la colaboración entre participantes del mismo estatus o jerarquía.
- **Lluvia de ideas:** Centrada en colaboraciones informales entre actores con la misma jerarquía, por tanto, los turnos de participación se especifican mediante el principio: "el primero que llega, es el primero que participa".
- **Moderada:** Diseñada para estructuras jerárquicas, que son controladas por una sola persona, el moderador.
- **Puntos de reunión:** Los actores asisten a un "lugar" o "cuarto", donde puede colaborar con los demás y usar las herramientas que se encuentran ahí.
- **Específicas:** Se *c*orresponden con la necesidad de trabajo en grupo, como son, el modelo de una institución, de sistemas de gestión de conferencias, de un salón de clases, de colaboración de tutores y tutorados, etc.

Finalmente, el espacio de trabajo compartido se muestra por medio de interfaces de usuario, permitiendo a diversos usuarios colaborar, comunicarse y coordinarse. No obstante, para ello, primero los usuarios deben ingresar en dicho entorno, usando un manejador de sesión que les permita conectarse a la sesión y establecer el estilo organizacional del grupo.

2.2. Un estudio del estado del arte en el desarrollo de *groupware*

En el dominio de CSCW existe una gran variedad de propuestas y aportaciones de y para aplicaciones colaborativas, cada una ha suministrado elementos que, de una u otra manera, han contribuido a estimular el interés y el crecimiento de este dominio. Esta sección analiza los trabajos realizados para crear trabajo en grupo agrupados en cuatro enfoques; teniendo en cuenta el contexto y el propósito que los guía. Se presentan los trabajos que han sido un punto de referencia para la generación de nuevos procesos y/o que facilitan la creación de trabajo en grupo. Se destacan cuatro enfoques en el desarrollo de *groupware* [Sosa 06, Molina 09, Rodríguez 12]:

1. **Ad-hoc.** La aplicación se construye de acuerdo al problema específico que se pretende resolver; hasta ahora ha sido la tendencia habitual en la creación de *groupware*.
2. **Uso de *toolkits*.** Estos proporcionan un mayor nivel de abstracción de programación usando funciones y una API (*Application Programming Interfaces*). Algunos de los *toolkits* más representativos en la literatura de CSCW serán analizados en la sección 2.2.1: Groupkit, COAST, Habanero, JSDT y AGILO.
3. **Uso de componentes.** La construcción de *groupware* se lleva a cabo mediante el uso de bloques de construcción predefinidos. Presentando en la sección 2.2.3, los que se han utilizado con mayor frecuencia en el dominio CSCW: ANTS, EVOLVE, DACIA, CoCoWare y DreamTeam.
4. **Uso de modelado conceptual.** El proceso de desarrollo del entorno colaborativo se basa en el modelado conceptual, generado mediante la especificación de los elementos requeridos. Con respecto a éste se han hecho algunas propuestas (véase la sección 2.2.4), de las cuales revisamos los más transcendentales en CSCW: Teoría de la Coordinación, Cognición Distribuida, Análisis de Tarea, Teoría de la Actividad, AMENITIES, CIAM, TOUCHE, MOLLICC, Meta-modelo Genérico, Meta-Modelo de Entornos Virtuales, Modelo de Clasificación Socio-Técnico y MoCA.

2.2.1. *Toolkits* para desarrollar *groupware*

En esta sección se presentan los *toolkits* más representativos, que han sido usados para desarrollar aplicaciones colaborativas.

2.2.1.1. Groupkit

GroupKit [Roseman 92, Roseman 94], que proporciona un conjunto de herramientas (*toolkit*) para *groupware* síncrono, está basado en el Lenguaje de Herramientas de Comando *(Tool Command Language* -Tcl*)* y en un conjunto de *widgets* de *Tcl/Tk (ToolKit)* [Ousterhout 94]. Este último es una herramienta para desarrollar interfaces de usuario gráficas para aplicaciones monousuario. GroupKit utiliza el concepto de *conferencia* para denotar la abstracción del contexto de espacio de trabajo compartido, consistiendo en un conjunto de usuarios que interactúan a través de este entorno para llevar a cabo el trabajo colaborativo.

Groupkit ha tratado de reducir la complejidad inherente de *groupware*, proporcionando cuatro aspectos importantes:

1. *Infraestructura en tiempo de ejecución:* Consiste en una variedad de procesos distribuidos (colocados en varias máquinas) que automáticamente manejan la creación, interconexión y comunicación de los procesos centralizados y distribuidos que forman parte de una conferencia.

2. *Conjunto de abstracciones de programación groupware*: Permite controlar la sincronización de eventos y la distribución de los datos, supervisando el comportamiento de los procesos distribuidos, tomando acciones apropiadas en los cambios de estado, y compartiendo los datos relevantes.
3. *Widgets groupware:* Proporcionan al programador un conjunto de herramientas colaborativas genéricas para las aplicaciones multiusuario síncronas, que suministran valiosas características para implementar la interfaz de usuario gráfica (*Graphic User Interface - GUI*), facilitando su uso y la interacción entre los usuarios.
4. *Manejador de sesión:* Permite a los usuarios crear y manejar sus conferencias. El *manejo de sesión* en *GroupKit* se construye sobre la estructura denominada *entorno*, que ofrece un conjunto de eventos que ayudan a proporcionar la *conciencia de grupo*, indicando, por ejemplo, qué usuarios se han unido o han dejado una conferencia, o proporcionando el estado actual de ésta al usuario que acaba de entrar. El *manejo de sesión* está apartado del *groupware* y es construido por los desarrolladores para ajustar el estilo de trabajo del grupo a las necesidades del mismo, gracias a que está basado en protocolos abiertos que facilitan la flexibilidad para proporcionar diferentes políticas de *manejo de sesión*.

En el estilo arquitectónico GroupKit (véase Figura 4), cada conferencia es un proceso independiente, ejecutándose en un intérprete de *Tcl,* con un estado local. GroupKit extiende *Tcl*, permitiendo la llamada a procedimiento remoto *(Remote Procedure Call - RPC)* para dirigirse simultáneamente a través de una conferencia a todos los usuarios (incluso al remitente), a todos (excepto al emisor) o específicamente a un usuario.

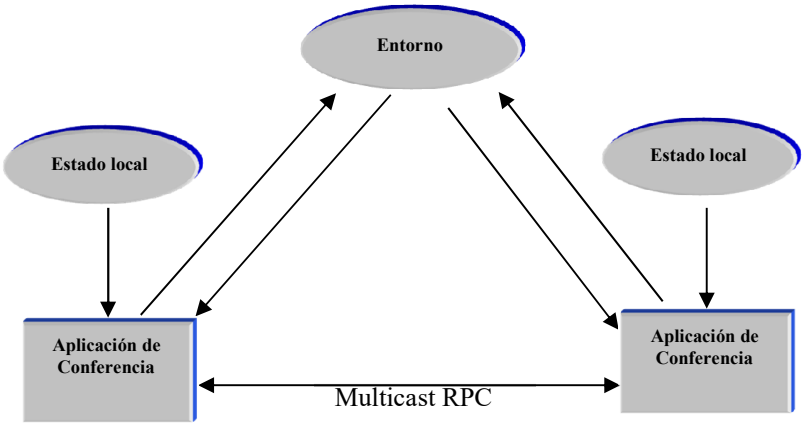


Figura 4. Esquema general de Groupkit [Roseman 94].

GroupKit permitió inicialmente que toda la comunicación entre aplicaciones fuera a través de RPC (*Remote Procedure Call*) [Alonso 04] o del modelo MVC (*Model-View-Controller*) [Goldberg 84], que permiten mantener los datos compartidos. Sin embargo, más adelante usó estructuras de datos denominadas *entornos*, que son diccionarios activos compartidos (colecciones de pares *clave-valor*) con claves colocadas en una estructura de árbol. Los usuarios manipulan los entornos directamente, vía la llamada a procedimiento. Las acciones sobre los entornos automáticamente generan eventos que son distribuidos a todos los miembros de la conferencia [Roseman 94]. Una conferencia puede incluir un número arbitrario de entornos; de esta forma, los usuarios pueden participar simultáneamente en múltiples conferencias.

Para concluir, indicar que Groupkit permite el desarrollo de aplicaciones que solo soportan trabajo síncrono distribuido, es dependiente de la plataforma, no es ejecutable en la Web y proporciona un único protocolo de comunicación.

2.2.1.2. COAST

COAST [Schuckmann 96] es un conjunto de herramientas basado en documentos compartidos entre sus procesos de aplicación. COAST ofrece componentes básicos y genéricos, un entorno para la construcción de aplicaciones *groupware* y una metodología para el desarrollo de *groupware* síncrono. COAST utiliza un enfoque de arquitectura replicada, es decir, una aplicación cooperativa que consta de varios procesos de aplicación ejecutándose en sitios diferentes. Cada aplicación opera exactamente en un documento compartido entre sus procesos de aplicación. Los datos del documento se replican completamente.

Un proceso de aplicación se diseña según la arquitectura general mostrada en la Figura 5. Cada aplicación cooperativa manipula un documento compartido. Un documento consta de partes que pueden utilizar una estructura arbitraria de objetos. Los documentos se replican completamente para cada proceso de aplicación. El acceso de los usuarios a un documento compartido (o a sus partes) es coordinado mediante un objeto de sesión. Los *objetos de sesión* proporcionan el conocimiento de grupo y el acoplamiento específico de aspectos del documento compartido entre los usuarios. Así, éstos se pueden usar para llevar a cabo modos cooperativos específicos. Los usuarios interactúan con el documento compartido por medio de las vistas (visualización del documento en una ventana) y los controladores (procesan la entrada del usuario desde la ventana).

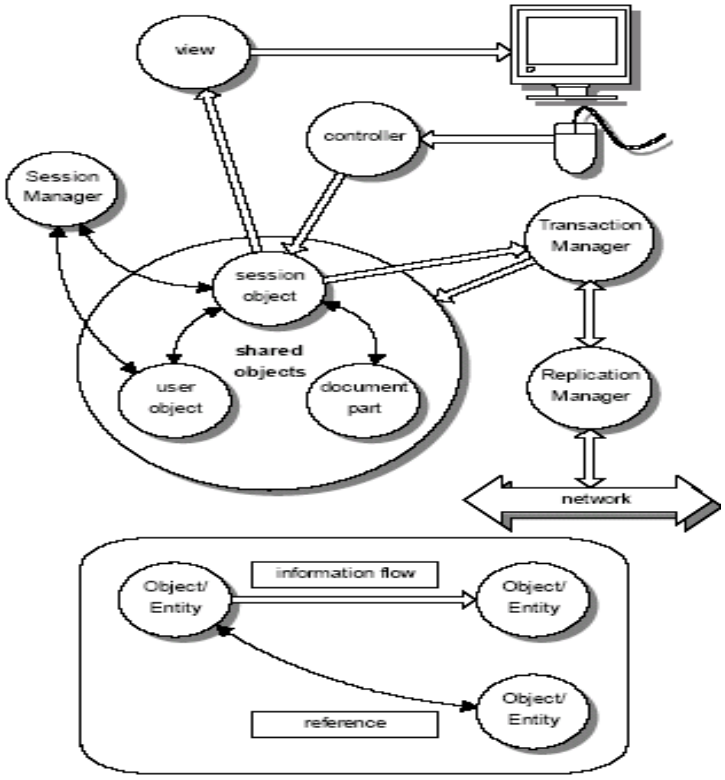


Figura 5. Arquitectura general de COAST [Schuckmann 96].

Las vistas y los controladores acceden al documento usando el objeto de sesión asociado. Se trata de una extensión del concepto MVC para habilitar acceso síncrono a los objetos del modelo compartido (es decir, documentos o sus componentes) de varios pares distribuidos vista/controlador vía réplicas de los objetos de sesión. Así, cada ventana de la aplicación (es decir, vista/controlador) despliega una parte del documento compartido asociado con una sesión que coordina el acceso a su parte del documento. Los *objetos de usuario*, que representan a los usuarios concurrentes del documento, son usados por los objetos de sesión para mantener las listas de usuarios que actualmente están participando. Todas las manipulaciones del documento compartido, así como las sesiones y los objetos de usuario, se encapsulan en transacciones. Un *manejador de transacción* asegura la integridad de los objetos compartidos. Un *manejador de*

replicación es el responsable de sincronizar los objetos replicados (es decir, documentos, objetos de usuario y de sesión). Guardando los objetos replicados, pueden hacerse documentos persistentes.

COAST suministra en tiempo de implementación:

- Modelado compartido del dominio, usando el lenguaje de descripción de COAST.
- Modelado predefinido y extensible de usuarios y de sus entornos de trabajo.
- Constructor de interfaz de usuario, que proporciona *widgets*.

En tiempo de ejecución, las aplicaciones COAST se benefician de:

- Replicación transparente de objetos de datos.
- Procesamiento de transacciones para asegurar la consistencia de los datos.
- Actualización automática de vistas, cuando cambian los objetos de datos compartidos.

2.2.1.3. Habanero

Habanero [Chabert 98] es un marco de trabajo desarrollado con el objetivo de hacer tan simple como sea posible para un desarrollador la creación de una aplicación compartida, ya sea alterando una aplicación monousuario existente o desarrollando una nueva aplicación desde cero. Este marco proporciona objetos que facilitan la construcción de software para comunicación síncrona sobre Internet; además, suministra un entorno colaborativo para crear y participar en sesiones, junto con una colección de herramientas para el trabajo colaborativo.

Habanero se centra en la construcción de *applets* Java para un entorno distribuido. Los *applets* deben estar disponibles como código fuente y en la mayoría de los casos pueden ser convertidos en un *applet* distribuido (denominado *Hablet*) sin demasiado esfuerzo. Habanero no soporta *multicast*, y el *canal* se ejecuta a través de un servidor, que se encarga de la gestión de las comunicaciones. *Groupware* se implementa como un conjunto de objetos clientes del API (*Application Programing Interface*) que suministra Habanero. Estos objetos interactúan a través de un objeto *wrapped*, que es responsable de manejar los eventos de la aplicación y de coordinar el resto de los objetos. Cada objeto *wrapped* proporciona una interfaz que permite a una aplicación desplegarse en una ventana, transferir el estado del programa y manejar eventos de otros clientes.

Habanero proporciona un conjunto de herramientas colaborativas para facilitar el desarrollo de *groupware*. Entre la colección de herramientas se cuenta con: herramientas básicas, que son esenciales para establecer una colaboración remota; herramientas de votación, para facilitar la coordinación; y herramientas para charlar en modo texto y/o audio, como instancias de comunicación, entre otras.

El entorno colaborativo soporta el proceso de *manejo de sesión*, permitiendo a cada participante crear, unirse a, salir de, y visualizar sesiones. Los entornos de Habanero soportan:

- **Múltiples Sesiones:** Un usuario puede colaborar concurrentemente en varias sesiones diferentes, pudiendo participar como anónimo si lo desea.
- **Información detallada sobre una sesión:** Suministrando, por ejemplo, su nombre, programa, agenda, lista de las herramientas colaborativas actuales y quiénes pueden participar en ella.
- **Facilidad de ejecución de sesión preconfigurada:** Permite a los usuarios describir una sesión una vez y guardar su información para su reutilización posterior.
- **Mecanismo de notificación:** Para invitar a los participantes a unirse a una sesión en línea. Este sistema soporta notificación síncrona (se abre una ventana en la pantalla de la persona invitada) y asíncrona (mediante un mensaje por correo electrónico).
- **Rol del usuario:** El iniciador de una sesión tiene un rol específico, que le permite establecer las características de una sesión, el grupo de colaboradores que pueden unirse a esa sesión, el proceso de notificación y el mecanismo de autenticación. Los colaboradores pueden desempeñar diferentes roles, como por ejemplo: participante activo, observador pasivo, u otros roles descritos por herramientas específicas.
- **Conciencia de grupo:** Habanero muestra quién está participando, dónde se localizan los participantes, qué actividad realizan y quién usó por última vez cada herramienta.

- **Extensión del conjunto de aplicaciones *groupware*:** Las aplicaciones pueden ser fácilmente agregadas al entorno de Habanero, que proporciona una API abierta para incorporar nuevas herramientas colaborativas, es decir, nuevas funcionalidades a la aplicación, permitiendo adaptarse a las nuevas necesidades del grupo. Sin embargo, estas nuevas herramientas colaborativas deben estar escritas en el lenguaje Java (que es el lenguaje en que se desarrolló dicho marco de trabajo), para que se puedan integrar de manera adecuada en el entorno Habanero.

La arquitectura de Habanero es centralizada, de modo que un cliente de Habanero se comunica con un único servidor. La comunicación se realiza por *acciones*, que son órdenes de ejecución distribuidas. El cliente duplica las *acciones* del servidor, para así darle más flexibilidad al entorno y permitir el trabajo individual localmente, sin estar conectado a un servidor.

2.2.1.4. JSDT

JSDT [Burridge 99] proporciona la abstracción básica de una sesión, es decir, un grupo de objetos asociados con algún patrón de comunicación común, que soporta comunicación completa de *multicast* a un número arbitrario de entidades conectadas a la aplicación sobre una variedad de diferentes tipos de red. La funcionalidad principal proporcionada por el mecanismo de comunicación de JSDT es la capacidad de colaboración consciente en Java, para enviar datos a todos (o a un subconjunto de) los usuarios dentro de una sesión de comunicación, lo cual se logra a través de tres tipos de implementación:

- **Socket.** Utiliza sockets TCP/IP (*Transmission Control Protocol/Internet Protocol*) para una comunicación fiable y restricciones de velocidad o un gran número de clientes, especialmente en Internet.
- ***HTTP***. Usa órdenes HTTP (*HyperText Transfer Protocol*). Este mecanismo de transporte es bueno si se trabaja en una LAN (*Local Area Network*). También es útil si es necesario cruzar cortafuegos (*firewalls*).
- **LRMP.** Usa LRMP (*Lightweight Reliable Multicast Protocol*) [Liao 00], que es un socket *multicast* ideal para distribuir información de vídeo y audio; sin embargo, hay que asegurarse de que los enrutadores (*routers*) soporten *multicast*, y no es muy conveniente para los *applets*.

JSDT principalmente proporciona:

- Soporte eficiente para la multidifusión de mensajes.
- Un mecanismo de sincronización distribuido basado en *token* para asegurar el acceso mutuamente excluyente a un recurso.
- Capacidad de asegurar la secuencia uniforme de mensajes entregados y de *tokens*.
- Capacidad de compartir vectores de bytes entre los miembros de una sesión.
- Ocultamiento de la variedad de protocolos de comunicación que utiliza para establecer la comunicación, así como de los procesos para seleccionar un protocolo específico.

Los elementos centrales de JSDT son *sesión*, *canal*, *vector de bytes* (*byte array*) y *token*, elementos que se explican a continuación:

- **Sesión:** Se construye alrededor de un modelo de conferencia similar a GroupKit, donde la conferencia JSDT se llama *sesión*. Los clientes en una sesión pueden interactuar enviando datos por los *canales*, modificando los *vectores de bytes* compartidos (véase Figura 6) o manejando primitivas de sincronización, llamadas *tokens*.
- **Token:** Es una primitiva de sincronización que puede ser requerida por un cliente. Se utiliza para advertir a los clientes que se han producido diferentes situaciones, por ejemplo: que otros clientes se han incorporado, se han ido, han sido invitados a unirse o han sido expulsados de la *sesión*. La semántica de los *tokens* está definida por la aplicación del cliente y se espera que las políticas de sincronización de alto nivel sean implementadas en las operaciones atómicas del *token*. En una sesión puede haber varios *tokens* para facilitar la sincronización entre los clientes y con los *vectores de bytes* que se estén utilizando. A los *tokens* se accede mediante un mecanismo de evento llamado *listener (*cada usuario a través de este evento conoce lo que sucede en el entorno

colaborativo, es decir, "está escuchando" lo que ahí sucede*)* o a través de un *callback* (referencia a una parte de código ejecutable que es pasado como argumento de otro código), lo que permite que una capa de bajo nivel llame a una función o subrutina definida en una capa de alto nivel).

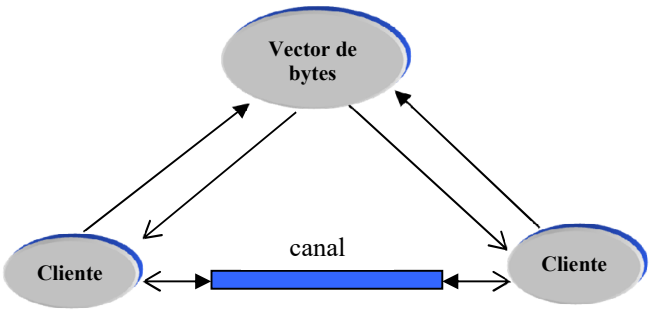


Figura 6. Arquitectura de JSDT [Burridge 99].

- **Canal:** Es el medio comunicación que puede dirigirse a un cliente específico o bien a todos los clientes (excepto al remitente) por medio del intercambio de mensajes. Un cliente puede unirse a varios canales en una sesión y puede supervisarlos mediante un bloqueo de lectura o registrando un *callback*. Para enviar datos a un cliente, primero los tiene que empaquetar en un *byte stream*, entonces invoca a uno de los métodos que proporciona JSDT para enviar mensajes por los canales tomando como parámetro el *byte stream*. De esta forma, muchos objetos pueden representarse como un *byte stream* mediante el mecanismo de serialización de Java.
- **Byte array:** Es el elemento de JSDT donde se almacenan los datos (los cuales son objetos serializados) que serán compartidos; por tanto, el *byte array* es compartido por los clientes que están colaborando en una *sesión* (véase la Figura 2.6). Para permitir que los clientes compartan varios datos, éstos pueden unirse a varios *bytes array* compartidos dentro de una sesión. El *byte array* usa una notificación al estilo MVC, es decir, sólo advierte a sus clientes de que algo ha cambiado en el vector; por lo tanto, los clientes deben obtener nuevamente el *byte array* y realizar la acción que se requiera.

Cada recurso JSDT (*sesión, canal, byte array o token*) puede tener un manejador de recurso asociado para implementar una política de gestión específica de la aplicación. Los manejadores de recursos proporcionan la autenticación y control de acceso al cliente. Entre las acciones que normalmente requieren autenticación están las de incorporar, crear o eliminar un recurso. Un manejador de recurso también puede expulsar a un cliente para que libere el recurso. El mecanismo del manejador de recurso le permite a JSDT soportar aplicaciones en las que los clientes son inestables o poco fiables.

2.2.1.5. Agilo

Agilo [Guicking 05] es un marco de trabajo basado en Java que permite desarrollar aplicaciones colaborativas síncronas y/o asíncronas, proporcionando una arquitectura modular, que permite suministrar suficiente flexibilidad y extensibilidad con respecto a cinco puntos de variación (que se describirán más adelante), facilitando la integración de diversas aplicaciones *groupware*. Su flexibilidad se incrementa mediante el uso de patrones de diseño del dominio de cómputo distribuido y concurrente. Esto conduce a una arquitectura *groupware* extensible y flexible, que permite la integración aplicaciones heterogéneas que se ajusten a la que se está construyendo.

Este marco de trabajo está diseñado sobre dos conceptos centrales: módulos y mensajes. Una aplicación *groupware* desarrollada por Agilo consiste en un conjunto de módulos ejecutándose en un nodo [Guicking 05] o más nodos del sistema. Los mensajes de entrada en un nodo son procesados secuencialmente y reenviados a uno o más módulos de aplicación, que suele enviar mensajes a uno o más módulos ejecutándose en otros nodos como resultado del procesamiento de un mensaje de entrada. Proporcionando este concepto de comunicación basado en mensajes, el marco de trabajo simplifica el desarrollo de aplicaciones *groupware* con proceso de comunicación simple o sofisticada. Las aplicaciones construidas

con Agilo pueden ser tan pequeñas como sea posible, ya que la mayor parte de la funcionalidad se implementa como módulos separados. Este enfoque ofrece dos ventajas:

1. Es necesario poco conocimiento sobre el marco de trabajo para construir una aplicación *groupware.*
2. Puede ser extendido sucesivamente en caso necesario.

Agilo proporciona:

- **Un modelo de distribución.** Ha sido diseñado para arquitecturas Cliente-Servidor como Peer-to-Peer. La arquitectura Cliente-Servidor, consta de tres partes: una del lado del cliente, una del lado del servidor, y una parte común que se requiere para ambos, cliente y servidor. La distribución Peer-to-Peer se consigue haciendo que en cada nodo se ejecuten componentes cliente y servidor. Además, ambas arquitecturas de distribución requieren configuraciones específicas para adaptar la funcionalidad del servidor y del cliente para trabajar en los respectivos tipos de distribución.
- **Una infraestructura de comunicación.** Permite el uso de diferentes protocolos de transporte y de datos. La comunicación entre cliente y servidor o entre pares (o iguales) puede ser personalizada en dos niveles: 1) En el nivel inferior, Agilo soporta diferentes protocolos de transporte, como TCP o HTTP. 2) El nivel superior soporta diferentes protocolos de intercambio de datos (por ejemplo, SOAP, XMPP), facilitando la integración de clientes heterogéneos, tales como PDAs y teléfonos inteligentes.
- **Un modelo compartido.** Además del concepto básico de mensajes de "bajo nivel", esta plataforma ofrece soporte para objetos compartidos a "alto nivel" que se implementan en los mensajes y los módulos. Agilo proporciona interfaces concretas de los objetos que deben ser compartidos, mientras que el esquema de distribución se lleva a cabo en un módulo separado denominado *Object Manager* (Manejador de Objetos). Los datos y su esquema de distribución están desacoplados, lo que permite el uso de diferentes sistemas de distribución, como centralizados, semi-replicados u objetos compartidos replicados.
- **Un modelo de concurrencia.** Prevé la ejecución en múltiples hilos concurrentes que pueden trabajar juntos al mismo tiempo en el contexto de la aplicación *groupware*. Otro aspecto de la concurrencia es el procesamiento de mensajes de entrada en los nodos del sistema. Además, si no hay necesidad para un orden globalmente consistente en los mensajes, en lugar de un simple hilo de la *Ruta del Mensaje*, puede ser usada una implementación multi-hilo con el patrón *Conducir/Seguir* [Schmidt 00] para notificación de los mensajes del módulo.
- **Un modelo de sincronización.** Utiliza diferentes mecanismos de bloqueo, tales como semáforos y exclusión mutua, así como mecanismos de sincronización construidos en Java para hacer cumplir el acceso controlado a datos compartidos; permitiendo la detección y resolución de conflictos. Agilo soporta el uso de transacciones para combinar múltiples acciones de un cliente en una acción atómica.

2.2.1.6. Análisis de *toolkits*

Los *toolkits* revisados anteriormente se analizan de manera general y de acuerdo a los elementos descritos en la sección 2.1 sobre los aspectos de Comunicación, Colaboración, Coordinación y Entorno.

En la Tabla 1 se presenta un panorama general de los *toolkits*, mostrando que todos están centrados en el grupo, basados en objetos y con mecanismos novedosos, en su momento, para desarrollar *groupware*.

En la Tabla 2 se analiza el aspecto de Comunicación, destacando que la mayoría de los *toolkits* proporcionan comunicación síncrona sólo Agilo presenta ambos tipos de comunicación; suministran la comunicación mediante el patrón de diseño MVC excepto Agilo y con herramientas como *socket*, RPC, TCP/IP, HTTP y/o LRMP; así como usan mensajes como la base de la comunicación, menos COAST que maneja documentos para ello.

Tabla 1. Panorama general del *toolkit*.

NOMBRE	ENFOQUE	ONTO_ LOGÍA	REPRESEN _TACIÓN	PRINCIPAL CONTRIBUCIÓN
Groupkit	Centrado en el grupo	NO	Orientado a objetos	Un conjunto de herramientas para *groupware* síncrono usando Tcl/TK.
COAST	Centrado en el grupo	NO	Orientado a objetos	Ofrece componentes básicos y genéricos para crear *groupware* síncrono.
Habanero	Centrado en el grupo	NO	*Hablet* (objeto distribuido)	Crear una aplicación compartida, desde cero o con una aplicación monousuario existente.
JSDT	Centrado en el grupo	NO	Orientado a objetos	Soporta comunicación *multicast* a un número arbitrario de entidades conectadas a la aplicación en una variedad de diferentes tipos de red.
Agilo	Centrado en el grupo	NO	Orientado a objetos	Desarrolla aplicaciones colaborativas síncronas y/o asíncronas.

Tabla 2. Aspecto de Comunicación en los *toolkits*.

ToolKit	Comunica_ ción	Modelo de Distribución	Middleware	Activación de la Comunicación
Groupkit	Síncrona	Peer-to-Peer, y un servidor central del registro de usuarios.	RPC a través del patrón MVC o *entornos* basados en RPC.	Utiliza eventos para iniciar el intercambio de mensajes entre procedimientos.
COAST	Síncrona	Cliente-Servidor, con un enfoque de arquitectura replicada.	Protocolo propietario basado en TCP, controlado por MVC.	Se realiza a través de documentos que son gestionados por objetos asociados al patrón MVC.
Habanero	Síncrona	Arquitectura cliente-servidor.	*Socket* TCP/IP basado en el MVC.	Se realiza por *mensajes*, realizados como objetos del cliente.
JSDT	Síncrona	Arquitectura centralizada.	Utiliza *Socket* TCP/IP, HTTP y LRMP con MVC.	Intercambio de mensajes de los objetos por medio de un canal.
Agilo	Síncrona y Asíncrona	Cliente-Servidor y Peer-to-Peer.	Comunicación en dos niveles: 1) TCP o HTTP; 2) protocolos de intercambio de datos.	Los módulos que constituyen Agilo se comunican a través de mensajes.

Con respecto al aspecto de Colaboración, se puede apreciar en la Tabla 3, que la mayoría de los *toolkits* utilizan el concepto de rol y de usuario, gestionan recursos excepto Groupkit y COAST, no usan el concepto de Tarea menos Groupkit. Además, que la interacción y la notificación son manejadas de diferentes formas en cada *toolkit*, debido a que sus elementos que permiten desarrollar *groupware* son distintos.

Tabla 3. Aspecto de Colaboración en los *toolkits*.

Nombre	Usua _rio	Rol	Ta_ rea	Recurso	Manejo de Interacción	Notificación
Groupkit	SI	SI	SI	Entornos	Las acciones sobre los *entornos* generan eventos distribuidos a los usuarios de la conferencia.	Uso de eventos, para sincronizar cambios en vistas o modelos.
COAST	SI	SI	NO	Docu-mentos	El acceso de los usuarios a un documento compartido es coordinado por el objeto sesión.	Supervisa el comportamiento de los procesos distribuidos, realizando acciones apropiadas a los cambios.
Habanero	SI	SI	NO	Objetos	Proporciona *Hablets* para transmitir la información a los usuarios por el componente canal.	Sólo soportado para invitar a usuarios a unirse a la sesión, de manera síncrona o asíncrona.
JSDT	SI	SI	NO	Objetos	El vector de bytes es enviado al canal por los usuarios que están en una sesión.	Advierte a los usuarios que algo ha cambiado en el vector de bytes, así que éstos llaman a dicho vector.
Agilo	SI	SI	NO	Objetos	Los objetos compartidos implementan una interfaz específica para ser gestionados por el *Object Manager*.	El esquema de distribución se realiza en el módulo *Object Manager*. Los datos y su esquema de distribución están desacoplados.

Con referencia al aspecto de Coordinación, se puede ver en la Tabla 4, que la mayoría de los *toolkits* utiliza mecanismos de sincronización basados en diferentes elementos (manejo de abstracciones, replicación, eventos, tokens, control de acceso, *floor control* y bloqueo) para asegurar el acceso mutuamente excluyente a los recursos; manejadores de sesión acordes a los elementos que suministra; y políticas de Manejo de Sesión basadas en la asignación de roles a los usuarios.

Tabla 4. Aspecto de Coordinación en los *toolkits*.

Nombre	Mecanismo de Concurrencia	Manejo de Sesión	Política de Manejo de Sesión
Groupkit	Con conjunto de abstracciones de programación *groupware*.	Manejador de sesión basado en la estructura entorno	Uso de protocolos abiertos para suministrar diferentes políticas de manejo de sesión.
COAST	Un manejador de transacción ofrece la integridad de objetos. Un manejador de replicación sincroniza los objetos.	El acceso del usuario a un documento compartido es coordinado por objeto sesión	Los usuarios interactúan con el documento por vistas y controladores, accediendo al mismo con el objeto de sesión asociado.
Habanero	Suministra un objeto para manejar los eventos y coordinar al resto de los usuarios.	Permite a cada participante crear, unirse a, salir de, y visualizar sesiones.	Asigna roles a los usuarios y cada recurso tiene un manejador que ejecuta una política de gestión específica de la aplicación.
JSDT	Mecanismo de sincronización distribuido basado en token, para asegurar el acceso mutuamente excluyente a un recurso.	El Token advierte a los clientes quiénes se han incorporado, ido, han sido invitados a unirse a o expulsados de la sesión.	A cada usuario se le asigna un rol.
Agilo	Utiliza diferentes mecanismos de bloqueo y sincronización, permitiendo la detección y resolución de conflictos.	Manejador de conexión a sesiones y de los usuarios a éstas.	A cada usuario se le asigna un rol.

En la Tabla 5 se presenta el aspecto de Entorno. Se puede apreciar que la mayoría de los *toolkits* denominan Sesión a su espacio de trabajo compartido menos Groupkit que le llama Conferencia; y hacen referencia a las interfaces de usuario compartidas, pero sólo manejan de manera implícita el concepto Vista de Participantes de los tres tipos de vista revisados (Información, Participantes y Contexto) en la sección 2.1, ya que proveen conciencia de grupo.

Tabla 5. Aspecto de Entorno en los *toolkits*.

Nombre	Espacio	Interfaz de Usuario
Groupkit	Conferencia	Uso de Widgets *groupware* para implementar interfaz de usuario gráfica. La estructura entorno maneja eventos, que proporcionan la conciencia de grupo.
COAST	Sesión	Los objetos de sesión proporcionan la conciencia de grupo y el acoplamiento específico de aspectos del documento compartido entre los usuarios
Habanero	Sesión	Presenta interfaces de usuario de la plataforma y de cada herramienta. Suministra la conciencia de grupo, mostrando quién está participando, dónde están localizados, qué actividad realizan y quién usó por última vez cada herramienta.
JSDT	Sesión	Presenta interfaces de usuario basadas en Java. Así como colaboración consciente para enviar datos a todos (o a un subconjunto de) usuarios.
Agilo	Sesión	Al *token* se accede vía un *listener* (cada usuario a través de este conoce lo que sucede en el entorno colaborativo). Además, proporciona interfaces de los objetos compartidos, así como de objetos específicos de la aplicación.

Finalmente, del análisis de *toolkits* para desarrollar *groupware,* se concluye que:

- Son Herramientas que ayudan a construir *groupware* síncrono y asíncrono.
- Usan el patrón de diseño MVC para sustentar este desarrollo.
- El modelo de distribución implementado es principalmente cliente-servidor.
- El middleware aplicado se centra generalmente en RPC y/o protocolos TCP/IP mediante intercambio de mensajes.
- Suministran la interacción de usuarios de manera simple y variada.

- Proporcionan mecanismos de notificación y concurrencia básicos, aunque las formas de hacerlo son distintas.
- Proveen tanto manejo de sesión como política de manejo de sesión, la primera de forma explícita (con mecanismos bien definidos) y la segunda de manera implícita (ya que ambas son dependientes).
- Ofrecen interfaz de usuario compartida y widgets para que los usuarios sean consciente de lo que ocurre en la sesión.

2.2.2. Desarrollo de *groupware* basado en componentes

El desarrollo basado en componentes busca crear aplicaciones que estén compuestas por un conjunto de bloques que contengan las funcionalidades requeridas por las mismas [Andersen 00]. Esta forma de creación contribuye a facilitar la extensión y adaptación de una aplicación. Es por ello, que el enfoque de componentes ha sido usado en el desarrollo de *groupware.*

2.2.2.1. ANTS

ANTS [García 03] está diseñado para proporcionar un marco de trabajo colaborativo, multiusuario y genérico, que se basa en tecnologías distribuidas y ofrece servicios CSCW genéricos a los usuarios y a las comunidades de desarrollo. ANTS usa Java y JavaBeans como tecnologías de desarrollo. El marco define un servicio de conciencia de grupo y uno de monitoreo, integrados de manera adecuada con el modelo de componentes que se define en JavaBeans. Este modelo oculta la complejidad a los usuarios, proporcionando persistencia remota de forma transparente, eventos distribuidos, descriptores y empaquetamiento de componentes. Los servicios de monitoreo y el modelo de componentes se establecen sobre el *middleware* de integración del marco de trabajo, que utiliza un enfoque estandarizado para servicios distribuidos.

ANTS brinda a los desarrolladores una variedad de herramientas y diseños arquitectónicos para la gestión de diferentes conjuntos de problemas, mediante la combinación de servicios arquitectónicos centralizados y replicados. La arquitectura general se compone de tres capas (véase Figura 7): capa de tecnología, capa CSCW y capa de aplicación.

La *capa de aplicación* abstrae el acceso a propiedades remotas basadas en componentes persistentes y eventos distribuidos. La información de eventos proporciona la infraestructura de conciencia, ya que permite conocer qué eventos se activan por cada recurso en el entorno compartido. Los componentes empaquetados contienen descriptores XML para que puedan ser registrados en el repositorio de componentes y enviados al servidor ANTS. El repositorio controla los permisos para obtener información sobre los componentes registrados, así como insertar y remover componentes en contextos de sesión compartida. Para ello, ANTS soporta coordinación genérica, proporcionando reglas de coordinación para el acceso, concurrencia y *floor control* [Dommel 97]. Estos manejadores de coordinación interponen llamadas a un componente para asegurar la validez de la acción.

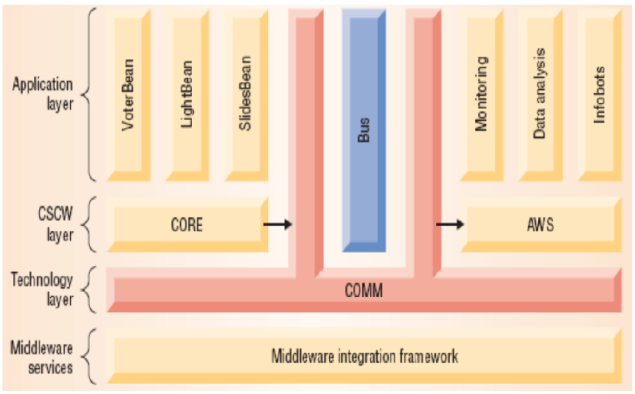


Figura 7. Arquitectura general de ANTS [García 03].

La *capa CSCW* describe el conjunto de servicios colaborativos que el contenedor de ANTS ofrece a la capa de aplicación. Contiene dos módulos principales interconectados transparentemente a través del bus de colaboración:

- **CORE:** es un contenedor en tiempo de ejecución, que incluye soporte para sesiones, artefactos compartidos, control de coordinación, seguridad e integración. ANTS suministra el objeto lugar para representar una sesión compartida, la cual contiene usuarios, componentes y enlaces a otros objetos lugar. Estos objetos son interconectados por objetos portal, que proporcionan métodos para enviar o suscribirse a los eventos en la sesión actual. Los manejadores de coordinación de ANTS se basan en el modelo de componentes de JavaBeans; estos manejadores aprueban o vetan los cambios de propiedades, para lo que se proporcionan coordinadores de control de acceso y *floor control*, además de un componente de bloqueo centralizado basado en *token*, que es la base para el *floor control*.
- **AWS:** Es un servicio de conciencia genérico, que proporciona acciones apropiadas para manejar los eventos recibidos del bus de colaboración. Este servicio permite activar un conjunto de actuadores que realizan tareas especiales en respuesta a sensores. Los sensores representan cualquier componente físico o de software que produce y transmite eventos al sistema de notificación. Los sensores se activan en respuesta a eventos de la aplicación que produce el bus de información. AWS crea un evento sensor mediante una suscripción que sigue las restricciones disponibles en el sistema de notificación. El planificador de tiempo se activa con sensores de tiempo, que pueden ser creados para una fecha específica o incluso para activar fechas repetitivas. El planificador de tiempo notifica al servicio de conciencia cuándo se activa un sensor de tiempo. Se integran descriptores XML en componentes empaquetados con el modelo de componentes de AWS. Cada descriptor contiene los eventos para que el componente active al bus de colaboración. AWS proporciona la base para aplicaciones de supervisión sofisticadas, agentes inteligentes y actuadores persistentes activados por eventos recibidos desde el sistema de notificación.

La capa de tecnología de ANTS usa J2EE como infraestructura tecnológica, permitiendo basarse en especificaciones y componentes de código libre; de esta forma, ANTS es independiente del proveedor. El Middleware Orientado a Mensajes (*Message-Oriented Middleware* —MOM) [Alonso 04] de este marco de trabajo especifica varios alias del bus *para* conectarse a diferentes servicios del middleware, como son: Servicio de Mensajes de Java (*Java Message Service* —JMS), Elvin (servicio de ruteo de eventos que utiliza el modelo de entrega publicador/suscriptor), SSL (*Secure Sockets Layer*), un canal de comunicación cifrado o el alias *RMCast*.

2.2.2.2. EVOLVE

EVOLVE [Stiemerling 99a, Stiemerling 00] es una plataforma basada en componentes que proporciona un entorno en tiempo de ejecución basado en Internet para implementar *groupware* y apoyar la personalización del usuario final. Los componentes se modifican por los usuarios en tiempo de ejecución, incluso aquellos recién adaptados están disponibles para otros usuarios o varios usuarios pueden ajustarlos a su entorno de aplicación de forma colaborativa. EVOLVE utiliza una arquitectura cliente-servidor.

La estructura de un *groupware* se describe por cinco tipos diferentes de archivos que se guardan en el servidor: archivo de la Arquitectura de Componentes para Personalización (*Component Architecture for Tailorability* — CAT) del servidor, archivo CAT del cliente, archivo de enlace remoto, Documento DCAT y una tabla de usuario. Un archivo CAT del servidor describe la estructura de composición del lado de servidor del *groupware*, mientras que un archivo CAT del cliente describe la estructura de composición del lado de cliente. Un archivo de enlace remoto describe la interconexión entre un archivo CAT del cliente y un archivo CAT del servidor. Un archivo DCAT describe un sistema completo especificando los componentes cliente y servidor del sistema. Este archivo se refiere a un archivo de enlace remoto específico

para determinar la conexión entre clientes y servidores. La tabla de usuarios relaciona los archivos DCAT con los usuarios, especificando qué aplicaciones son accesibles para los usuarios.

Los componentes de EVOLVE se implementan utilizando Flexibeans, una extensión del modelo de componentes JavaBeans. Flexibeans extiende JavaBeans con los conceptos de puertos nombrados, objetos compartidos e interacciones remotas. Los componentes de FlexiBeans se comunican a través de variables compartidas, así como del paso de eventos. Ambos mecanismos funcionan dentro de la misma máquina, así como a través de la red, para soportar el uso distribuido. Los componentes proporcionan "puertos" que aceptan o envían eventos específicos, y se vinculan para conducir el flujo de eventos de un componente al siguiente.

Los objetos compartidos son proporcionados por los componentes basados en la arquitectura distribuida Java RMI (*Remote Method Invocation*). El problema de la concurrencia en el acceso a objetos se aborda a través del uso de *synchronized* de Java, que sólo permite un hilo activo a la vez para pasar una sección crítica.

Los usuarios finales disponen de una herramienta de adaptación gráficamente interactiva, en la cual los puertos de los componentes se pueden conectar para construir la lógica de la aplicación. Los cambios realizados en la composición del componente se reflejan inmediatamente en todos los clientes que utilizan actualmente una instancia de la composición modificada.

2.2.2.3. DACIA

DACIA [Litiu 00, 99] es una plataforma para crear *groupware* adaptable para móvil, a través de la composición flexible de componentes separados. El *groupware* puede ser creado mediante la composición de Componentes de Procesamiento y Ruteo (PROC, *Processing and ROuting Components*), donde los datos se recopilan, manipulan y muestran en varios sitios. La configuración de PROC se puede modificar en tiempo de ejecución con el uso de una interfaz de gestión para la modificación interactiva de las configuraciones de los componentes. De esta manera, el sistema resultante es altamente configurable y adaptable, así como extensible, en tiempo de ejecución. Además, un PROC puede aplicar algunas transformaciones a flujos de datos de entrada, tales como sincronización de flujos de datos de entrada y división de los elementos de un flujo de datos de entrada en múltiples destinos.

Un motor (*Engine*) es otro elemento de DACIA (véase la Figura 8), que se ejecuta en cada host y es responsable de tareas administrativas, tales como mantener la lista de PROCs y sus conexiones, establecer y eliminar conexiones entre PROCs, migrar PROCs de un host a otro y establecer y mantener conexiones y comunicación entre hosts. Un motor sólo tiene conocimiento parcial sobre un PROC que se ejecuta en otros hosts y la configuración de la aplicación en sí.

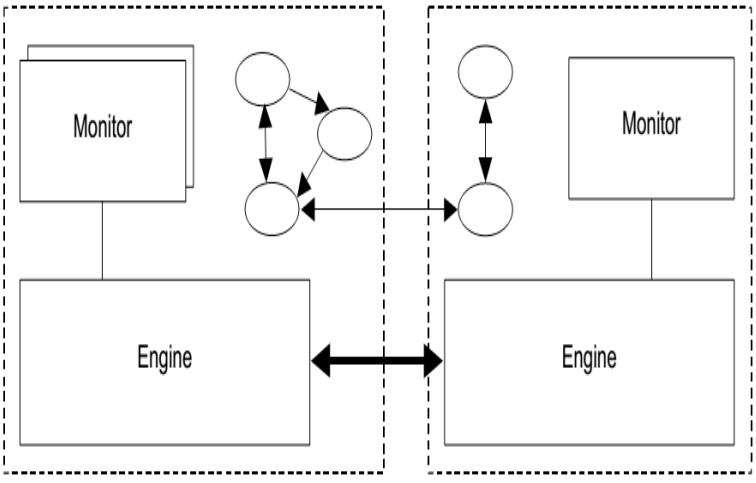


Figura 8. Esquema general de DACIA.

Un motor trabaja generalmente junto con uno o más monitores (*Monitor*). Un monitor (véase la Figura 2.8) representa la parte de la aplicación que reúne datos de rendimiento e implementa políticas de recolocación y reconfiguración. Sin embargo, los monitores son opcionales, ya que las aplicaciones también pueden reconfigurarse manualmente. Mientras que el motor y los PROC son de uso general, un monitor suele ser específico de la aplicación.

2.2.2.4. CoCoWare

CoCoWare [Slagter 01] es una plataforma dirigida a la composición y extensibilidad de *groupware* basada en CORBA (*Common Object Request Broker Architecture*). CoCoWare implementa una arquitectura de componentes duplicados denominada Gente y Sistemas Cooperativos (CooPS, *People and Systems Cooperative*). CooPS es una extensión del modelo propuesto por Ter Hofte en [Hofte 98]. Una aplicación de *groupware* de acuerdo con CooPS consta de cuatro tipos de componentes (véase la Figura 9):

- **Administrador de Conferencia (*Conference Manager*)**: Es responsable de administrar la existencia y participación de la conferencia, así como de las subconferencias existentes de una conferencia. Este componente permite a los usuarios finales iniciar y detener conferencias, unirse o salir de la conferencia e invitar o expulsar a los participantes de la conferencia. Hay una instancia de este componente para cada conferencia en la que el usuario participa.

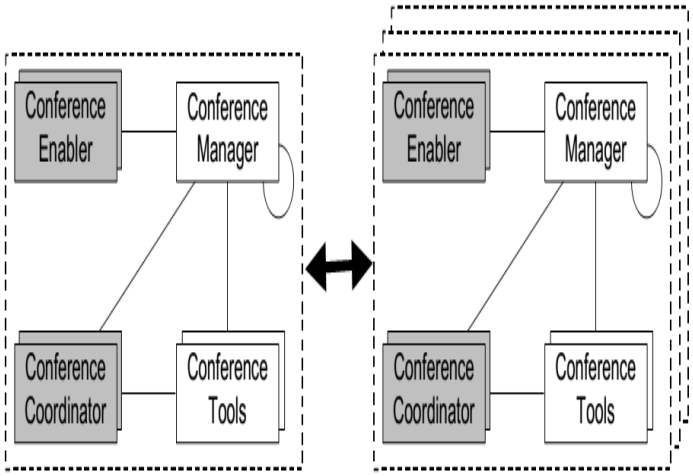


Figura 9. Esquema general de CooPS.

- **Herramienta de Conferencia (*Conference Tools*)**: Es el medio real para que los usuarios se comuniquen, se pueden usar varias herramientas en cada conferencia. CoCoWare contiene herramientas para conferencias de audio, videoconferencias, chat, navegación Web colaborativa y uso compartido de aplicaciones.
- **Coordinador de Conferencia (*Conference Coordinator*)**: Asigna los roles a los usuarios y promulga políticas de coordinación sobre las acciones del usuario basadas en estos roles y sobre el estado de la conferencia. El coordinador de la conferencia es un componente opcional. Una política de coordinación puede estar relacionada con varios componentes, pero un componente puede estar coordinado por al menos una política.
- **Habilitador de Conferencia (*Conference Enabler*)**: Proporciona a los usuarios finales y componentes de *groupware* información sobre los usuarios que serán invitados a una conferencia o conferencias. El habilitador de conferencias es un componente opcional, ya que la información proporcionada por este componente se puede obtener por otros medios, tales como correo electrónico o teléfono.

2.2.2.5. DreamTeam

DreamTeam [Roth 00] es una plataforma de *groupware* basada en componentes síncronos, que contiene:

- Un entorno de desarrollo, que consiste en soluciones específicas de *groupware*.
- Un entorno de tiempo de ejecución, proporcionando soluciones especiales de *groupware* y un entorno de simulación.

La arquitectura de DreamTeam es totalmente replicada. Una aplicación de DreamTeam consta de varios recursos, cada uno contiene una interfaz estándar. Los componentes son el tipo de recursos más importante (otros tipos de recursos incluyen interfaz de usuario y datos) y se llaman TeamComponents, que son recursos especiales que agregan recursos, excepto para otros TeamComponents.

Existen dos modos de comunicación entre recursos:

- *Comunicación intra-sitio:* En la que un recurso se comunica con su aplicación (mismo host), utilizando llamadas de método Java estándar.
- *Comunicación entre-sitios:* En la que un recurso se comunica con su recurso igual en otro host, mediante un sistema de comunicación subyacente basado en mensajes para distribuir los eventos.

La interfaz estándar de un componente proporciona varias llamadas de método para acceder a sus atributos, como datos de perfil, modo de integración, modo de colaboración, estado y notificación de eventos.

Un componente se integra perfectamente en su aplicación, porque la interfaz de usuario de componentes se convierte en parte de una ventana existente o como un vínculo. Además, un componente tiene también diferentes niveles de colaboración, tales como:

- Privado: No tiene datos compartidos.
- Exclusivo: Tiene algunos datos compartidos que pueden ser manipulados por un solo usuario a la vez.
- Compartido. Tiene datos compartidos que pueden ser manipulados por varios usuarios simultáneamente.

DreamTeam tiene varios mecanismos para el control de la concurrencia, como: bloqueo de transacción para la aplicación, bloqueo de monitor para cada recurso y bloqueos explícitos para variables y llamadas de método. DreamTeam también proporciona una serie de componentes básicos, incluyendo componentes de: dibujo, interfaz de usuario, navegador y audio.

2.2.2.6. Conclusión de *groupware* basado en componentes

De la revisión del desarrollo de *groupware* basado en componentes, se puede concluir (Tabla 6) que esta forma de desarrollo busca la adaptación de la aplicación de acuerdo a las necesidades del usuario final, usando principalmente un modelo de componentes fundamentado en JavaBeans, sólo CoCoWare utiliza CORBA y DreamTeam no maneja ningún modelo de componentes.

Tabla 6. Descripción General de *groupware* basado en componentes.

NOMBRE	ENFOQUE	ONTO_ LOGÍA	REPRESEN_ TACIÓN	PRINCIPAL CONTRIBUCIÓN
ANTS	Centrado en el usuario	NO	JavaBeans	Proporciona un *framework* colaborativo, multiusuario y genérico.
EVOLVE	Centrado en el usuario	NO	JavaBeans	Entorno en tiempo de ejecución para crear *groupware* y soportar la personalización del usuario.
DACIA	Centrado en el usuario	NO	JavaBeans	Crear *groupware* adaptable para móvil, a través de la composición flexible de componentes separados.
CoCoWare	Centrado en el usuario	NO	CORBA	Plataforma dirigida a la composición y extensibilidad de *groupware* basada en CORBA.
DreamTeam	Centrado en el usuario	NO	Team_ Components	Plataforma basada en componentes síncronos, con un entorno de desarrollo y entorno de ejecución.

En la Tabla 7 se puede apreciar que tres propuestas (EVOLVE, DACIA y CoCoWare) suministran comunicación síncrona y asíncrona en un modelo de distribución cliente-servidor, mientras dos (ANTS y DreamTeam) sólo comunicación síncrona, el primero mediante servicios centralizados y replicados y el segundo sólo con servicios replicados. El middleware que sustenta la comunicación varía en cada propuesta de desarrollo de *groupware* basada en componentes, y por tanto, también la manera en que activan la comunicación.

Tabla 7. **Aspecto de Comunicación de *groupware* basado en componentes.**

ToolKit	Comunicación	Modelo de Distribución	Middleware	Activación de la Comunicación
ANTS	Síncrona	Centralizado y replicado	*Middleware orientado a mensajes*, gestionado a través de un MVC	Usa componentes que interactúan a través del intercambio de mensajes.
EVOLVE	Síncrona y Asíncrona	Cliente-servidor	Java RMI	Los componentes de FlexiBeans se comunican a través de variables compartidas y paso de eventos.
DACIA	Síncrona y Asíncrona	Cliente-servidor	Usa PROCs para recopilar, manipular y mostrar datos	La composición de PROCs,
CoCoWare	Síncrona y Asíncrona	Cliente-servidor	CORBA	Herramienta de Conferencia permite a los usuarios comunicarse.
Dream-Team	Síncrona	Replicado	La interfaz del componente tiene varias llamadas de método para acceder a sus atributos	Dos modos de comunicación entre recursos: intra sitios y entre sitios.

En el caso del aspecto de Colaboración (Tabla 8) en este tipo de aplicaciones estudiadas tanto el manejo de interacción como de notificación se centra en la manera en que los componentes interactúan y son informados para que la aplicación compuesta con ellos funcione conforme al objetivo especificado por dichas aplicaciones.

Tabla 8. **Aspecto de Colaboración de *groupware* basado en componentes.**

Nombre	Usuario	Rol	Tarea	Manejo de Interacción	Notificación
ANTS	SI	SI	NO	El bus de información envía eventos, activando un conjunto de actuadores para responder a los mismos.	Sistema de notificación que informa a los demás componentes, qué eventos han ocurrido en el entorno compartido.
EVOLVE	SI	NO	NO	Los componentes proporcionan "puertos" que aceptan o envían eventos específicos, y se vinculan para conducir el flujo de eventos de un componente al siguiente.	Los cambios en la composición del componente se reflejan inmediatamente en todos los clientes que utilizan una instancia de la composición modificada.
DACIA	SI	NO	NO	Un motor mantiene la lista de PROCs y sus conexiones, establece y elimina conexiones entre PROCs, migra PROCs entre hosts.	Un motor conoce parcialmente sobre un PROC que se ejecuta en otros hosts y la configuración de la aplicación en sí.
CoCo-Ware	SI	NO	NO	Una política de coordinación está relacionada con varios componentes, pero un componente está coordinado por al menos un componente.	Capacitador de Conferencia proporciona a usuarios y componentes información de los usuarios, que pueden ser invitados a una conferencia o conferencias.
Dream-Team	SI	NO	NO	Componente tiene niveles de colaboración: Privado, Exclusivo y Compartido.	No especificada.

En cuanto al aspecto de Coordinación (véase la Tabla 9), sólo en ANTS y CoCoWare especifican el manejador de sesión con componentes concretos. Con respecto a la Política de Manejo de Sesión, DreamTeam no la específica, mientras que los demás lo hacen de manera implícita con mecanismos acordes a la funcionalidad de la aplicación. Con referencia a la concurrencia se establecen mecanismos de sincronización basados en políticas de coordinación, *tokens*, control de acceso, *floor control* y diferentes formas de bloqueo.

Tabla 9. **Aspecto de Coordinación de *groupware* basado en componentes.**

Nombre	Mecanismo de Concurrencia	Manejo de Sesión	Política de Manejo de Sesión
ANTS	Coordinador de control de acceso, *floor control* y bloqueo centralizado basado en *token*.	Proporciona el módulo CORE, que incluye soporte de sesiones.	El repositorio controla los permisos para obtener información de los componentes, insertar y removerlos.
EVOLVE	Uso de *synchronized* de Java, que permite un hilo activo a la vez.	No especificado	La tabla de usuarios relaciona archivos DCAT con usuarios, especificando qué aplicaciones son accesibles a éstos.
DACIA	Un PROC realiza sincronización de flujos de datos de entrada	No especificado	Un PROC ejecuta la división de elementos del flujo de datos de entrada en múltiples destinos
CoCoWare	Mediante el Coordinador de Conferencia que define políticas de coordinación.	A través del Administrador de Conferencias.	Coordinador de Conferencia asigna los roles a los usuarios y promulga políticas de coordinación.
DreamTeam	Bloqueo de transacción de la aplicación, de monitor de recurso y llamadas de método.	No especificado	No especificado

Para el aspecto de Entorno, los tres primeros que se presentan en la Tabla 10 denominan al entorno compartido Sesión y los dos últimos Conferencia. La interfaz de usuario para DACIA y CoCoWare no es especificada, mientras para los otros tres ésta se refiere a una ventana que permite personalizar la integración de los componentes.

Tabla 10. **Aspecto de Entorno de *groupware* basado en componentes.**

Nombre	Espacio	Interfaz de Usuario
ANTS	Sesión	Se desarrollan con JavaBeans. AWS es un servicio de conciencia genérico, que maneja los eventos recibidos del bus de colaboración.
EVOLVE	Sesión	Los usuarios finales disponen de una herramienta de adaptación interactiva, en la cual los puertos de los componentes se conectan para construir la lógica de la aplicación.
DACIA	Sesión	No especificada
CoCoWare	Conferencia	No especificada
DreamTeam	Conferencia	Un componente se integra en su aplicación, porque la interfaz de usuario de componentes se convierte en parte de una ventana existente o como un vínculo.

Finalmente, el desarrollo de *groupware* ya sea síncrono y/o asíncrono basado en componentes se centra en suministrar aplicaciones que sean adaptables y extensibles. Por tanto en este sentido los aspectos analizados se dirigen a proporcionar la comunicación, colaboración, coordinación y entorno (interfaz gráfica) de los componentes con la finalidad de que la composición de los mismos resulte en una aplicación fácil de adaptar y de extender incluso por los usuarios finales.

2.2.3. Desarrollo de *groupware* basado en modelos

En las dos últimas décadas, este es el tipo de desarrollo de *groupware* que ha incrementado en el dominio de CSCW, porque trata de proporcionar una especificación conceptual para dicho desarrollo.

2.2.3.1. Teoría de la coordinación

La coordinación se define como "la gestión de dependencias entre actividades" [Crowston 06]. La teoría de la Coordinación permite identificar y estudiar las dependencias comunes y sus mecanismos de coordinación asociados. Por esta razón, esta teoría proporciona:

- un *framework* teórico para el análisis de la coordinación,
- una tipología de dependencias y
- mecanismos de coordinación relacionados.

De esta manera, en primer lugar, se desarrolla una descripción de las actividades involucradas, que es registrada en una tabla; segundo, se identifican las dependencias en el proceso (de una manera simple, se debe anotar cuándo una actividad produce un recurso requerido por otra), creando un flujo de recursos entre actividades y dependencias resultantes; tercero, se establecen la taxonomía de dependencias y los mecanismos de coordinación relacionados.

Establece tres tipos genéricos de dependencias: (1) requisito previo, el producto de una actividad es requerido por otro; (2) recurso común, un recurso es requerido por múltiples actividades, y (3) simultaneidad, dos o más actividades deben ocurrir al mismo tiempo.

El mecanismo de coordinación se describe de acuerdo a las dependencias. Por ejemplo, en la dependencia (o flujo) del productor/consumidor, donde una tarea crea un recurso requerido por otra, se analizan tres subdependencias:

- *usabilidad*, significa que el recurso creado por la primera tarea debe ser adecuado a las necesidades de la segunda;
- *transferencia*, representa que el recurso debe ser trasladado desde donde fue creado hasta donde se realizará la tarea que lo consumirá; y
- *precedencia*, simboliza que el actor desempeñando la segunda tarea debe saber cuándo el recurso estará disponible, para poder iniciar la tarea.

Este mecanismo de subdependencias se puede utilizar para analizar todos los procesos llevados a cabo en una organización.

2.2.3.2. Cognición distribuida

La Teoría de la Cognición Distribuida [Hollan 00] proporciona un entendimiento sobre las interacciones entre personas y tecnologías, que se logra en términos de la organización y propagación de restricciones. De esta manera, el énfasis teórico se centra en eventos.

La teoría sostiene que la actividad cognitiva se construye a partir de los recursos, tanto internos como externos, y que los significados de las acciones se basan en el contexto de la actividad. Por lo tanto, es necesario saber cómo la mente procesa la información y cómo la información que se maneja está dispuesta en el mundo material y social. Esto implica que la forma en que los actores pueden lograr la coordinación con el comportamiento dinámico de los materiales de trabajo debería tomarse en cuenta en el diseño de herramientas conceptualmente significativas y en los entornos de trabajo.

En pocas palabras, la Teoría de la Cognición Distribuida trata con:

- las distribuciones de los procesos cognitivos a través de los miembros de los grupos sociales;
- la coordinación entre la estructura interna y externa;
- la manera en que los productos de los eventos anteriores pueden transformar la naturaleza de los eventos posteriores.

2.2.3.3. Análisis de la Tarea

Una tarea es una actividad realizada para alcanzar un determinado objetivo. Tres tipos de tareas (compleja, unidad y básica) se consideran en el análisis de tarea [Van Welie 98], que se centra en estudiar la interacción de los usuarios mediante la revisión de las actividades realizadas por una organización o grupo. El modelado de tarea se centra en el desarrollo de la estructura de la tarea, que deberá tomar en cuenta el flujo de tarea y de datos a través del tiempo, es decir, la relación de dependencia entre dos o más tareas. A partir de la estructura de la tarea pueden crearse las pantallas centrales y la configuración de navegación; de esta manera, se establecen las principales interfaces de usuario. En el análisis de tareas, generalmente se consideran fundamentalmente tres conceptos: Agente, Trabajo y Situación.

El primer concepto, Agente, se refiere a quién lleva a cabo la tarea; puede ser una persona, individuo o grupo, pero también un sistema. El agente puede desempeñar uno o varios roles, que indican ciertos

subconjuntos de tareas asignadas a éstos. La asignación de las tareas constituye la organización que los usuarios tendrán, de acuerdo con la delegación de responsabilidades de cada rol.

El segundo concepto, Trabajo, resalta los aspectos estructurales y dinámicos del trabajo; por lo tanto, está relacionado con la tarea, que debe definir un marco de referencia para comprender el significado de las acciones realizadas. La estructura de la tarea (que es a menudo, al menos parcialmente, jerárquica) suministra los flujos de trabajo y de datos, lo que permite saber cómo se llevará a cabo el trabajo en grupo. En el análisis de tarea son considerados los eventos, para establecer una condición de activación que permita ejecutar una tarea.

El último concepto, Situación, pretende detectar y describir el entorno (físico, conceptual y social), así como los objetos que lo componen. El entorno de trabajo es la situación actual para llevar a cabo una tarea determinada, que incluye la historia de los eventos para activar cada tarea. Comúnmente, la estructura de la tarea se modela con: un árbol de descomposición de tareas, diagramas UML, y/o *workflows*.

2.2.3.4. Teoría de la actividad

Teoría de la Actividad [Kuutti 91] es un marco de trabajo filosófico para el estudio de diferentes formas de praxis humana, como procesos de desarrollo interrelacionados, tanto a nivel individual como social.

La teoría considera el contexto cultural en el que se lleva a cabo una actividad. Este contexto incluye varios elementos que determinan la estructura de la misma. En este sentido, un sujeto (persona) utilizará una herramienta para manejar un objeto y alcanzar una meta. El sujeto no actúa de manera aislada, sino dentro de una comunidad (u organización) mediada por un conjunto de reglas que determinan la forma de actuar del sujeto. La comunidad transforma el objeto de acuerdo con la división del trabajo (*division of labour*), que indica las acciones que el grupo debe realizar en relación con los roles establecidos para cada sujeto.

En la vida real, los sujetos realizan un conjunto de actividades interconectadas, definiendo la estructura externa de la actividad. Cada actividad consiste en acciones o cadenas de acciones, que a su vez contienen operaciones, construyendo así una estructura interna (que es jerárquica). Esto es, cuando una actividad se ejecuta, el sujeto hace un proceso cognitivo para hacer las operaciones necesarias de manera apropiada.

2.2.3.5. Modelo Conceptual

El Modelo Conceptual [Ellis 94] caracteriza con tres modelos complementarios el trabajo en grupo a partir de la percepción de los usuarios: ontológico, coordinación e interfaz de usuario.

El primero es una descripción de los objetos (estructuras de datos sobre la cual los participantes -usuarios- operan) y operaciones (transformaciones que actúan sobre los objetos). Los objetos tienen atributos, que contienen uno o más valores que pueden ser atómicos u otros objetos; sobre los cuales se pueden aplicar cuatro operaciones: ver, crear, modificar y destruir.

Este modelo ontológico considera la semántica intencional (objetos que se pueden utilizar) y operacional (el conjunto de operaciones y restricciones que se pueden aplicar en el objeto), y establece los derechos de operación y acceso para manipular un objeto.

El segundo establece cómo lleva a cabo su trabajo el grupo; por lo tanto, se trata de una descripción de las actividades (y su orden) que pueden realizar los usuarios de la aplicación. Las actividades son un conjunto potencial de operaciones -y los objetos correspondientes- que un actor -puede ser un usuario, un sistema informático o un grupo- desempeñando un rol particular puede llevar a cabo con el fin de alcanzar la meta definida. Un procedimiento está constituido por un conjunto de actividades y el orden entre ellas. Una instancia de cada orden se llama esfuerzo (*endeavor*); una tarea es una instancia de una actividad asociada a un esfuerzo particular.

Hay dos tipos de tareas: inactiva (si aún no ha comenzado o ya ha finalizado) y activa (si se está ejecutando). Este modelo indica la existencia de s-acciones (inicio de una acción), y t-acciones (terminación de una acción), así como, la precedencia temporal de las actividades. Un rol se asigna a las actividades y t-

acciones (esto generalmente es estático) y a un participante se le establecen uno o más roles (lo cual puede ser dinámico).

También propone dos niveles de coordinación:

- **Nivel de Actividad:** Trata de la precedencia de actividades, por tanto, describe la secuencia de actividades de un procedimiento,
- **Nivel de Objeto:** Trata del acceso compartido a los mismos objetos, secuencial o simultáneamente; por lo tanto, para evitar la modificación simultánea del objeto, utiliza mecanismos de control de concurrencia.

El modelo de coordinación define una fase como una descripción general de las actividades que están activas; reflejando cada uno de los momentos de colaboración. Puede haber fase simple (que no proporciona ninguna secuencia temporal de actividades) y compleja (cuando existe una secuencia temporal de actividades). Considera varios mecanismos para inspeccionar las fases: basada en participante (el sistema proporciona información de cada tarea activa); basada en esfuerzo (proporciona la información global de la fase de un *endeavor* en particular); total (que combina las dos anteriores); y de segundo orden (suministra información estadística).

Por otra parte, la fase se puede ajustar al mecanismo de coordinación establecido. En conclusión, se contemplan cuatro niveles de simultaneidad:

- **Secuencial:** Una tarea va después de otra.
- **Paralelo:** Las tareas ocurren al mismo tiempo, pero utilizan diferentes objetos, y no sucede ninguna interferencia entre ellos.
- **Aditivo concurrente:** Las tareas están activas al mismo tiempo, pero no hay ninguna modificación simultánea de cualquier objeto, sin embargo, se presenta un problema, se deben actualizar los objetos que son accedidos por los participantes, por lo tanto, se realizará un proceso de notificación.
- **Totalmente concurrente:** Cuando dos o más tareas simultáneas modifican a un mismo conjunto de objetos, se debe proporcionar un mecanismo de concurrencia para hacer frente a la modificación simultánea de los objetos.

El tercero es una descripción de la interfaz de usuario del sistema; que tiene tres componentes:

- **Vista de Objetos de Información:** Muestra los objetos y las operaciones del modelo ontológico, lo cual puede ser local o grupal; en el último caso, se debe aplicar un proceso de notificación a los participantes apropiados.
- **Vista de Participantes:** Proporciona un medio conveniente para conocer lo que hacen y están haciendo otros participantes.
- **Vista de Contexto:** Implica objetos y condiciones que están inmersos en todas las actividades colaborativas que han sido desarrolladas, así que en esta vista se despliega todo el material histórico de la sesión.

2.2.3.6. AMENITIES

AMENITIES [Garrido, 2003] considera y define conceptos relevantes para un sistema colaborativo, los cuales reflejan algunas ideas y propuestas de varios de los trabajos analizados en la sección anterior. Esta metodología está basada en modelos de tareas y contempla aspectos dinámicos sobre cómo los usuarios pueden pasar de desempeñar un rol a desempeñar otro por medio de una serie de leyes y capacidades. También se basa en un conjunto de modelos que están sustentados en UML.

AMENITIES presenta cinco etapas, que siguen un proceso iterativo [Garrido, 2003]:

1. Análisis del sistema y obtención de requisitos.
2. Modelado del sistema cooperativo.
3. Análisis del modelo cooperativo.
4. Diseño del sistema.
5. Desarrollo del sistema software.

La metodología se basa en cuatro modelos (véase la Figura 10) [Garrido, 2003]:

- **Modelo de Requisitos:** Obtiene y especifica los requisitos por medio de técnicas como la etnografía aplicada y los diagramas de casos de uso de UML.
- **Modelo Formal:** Realiza un análisis automatizado del sistema por medio de Redes de Petri Coloreadas (*Color Petri Nets —CPN*).
- **Modelo Cooperativo (COMO): M**odelo conceptual para describir y representar un sistema cooperativo desde su estructura y funcionamiento. Este modelo presenta cuatro vistas:
  - *Vista de Grupo:* Describe la organización y las restricciones que se imponen dentro de esa asociación, resaltando los conceptos de usuario, tarea y rol. Las relaciones entre usuarios dependen de sus capacidades para desempeñar un rol y las leyes que determinan cómo se debe comportar en esa estructura organizacional.
  - *Vista Cognitiva:* Representa el conocimiento que adquiere o posee un usuario. De acuerdo a este conocimiento tendrá que realizar un conjunto de tareas, lo que lleva a hacer un análisis de tareas. Primero se identifican las tareas, sus características (su naturaleza cooperativa, su mecanismo de activación y su modo de sincronización), las relaciones con otras tareas (si puede ser interrumpida por otra tarea) y con el entorno.

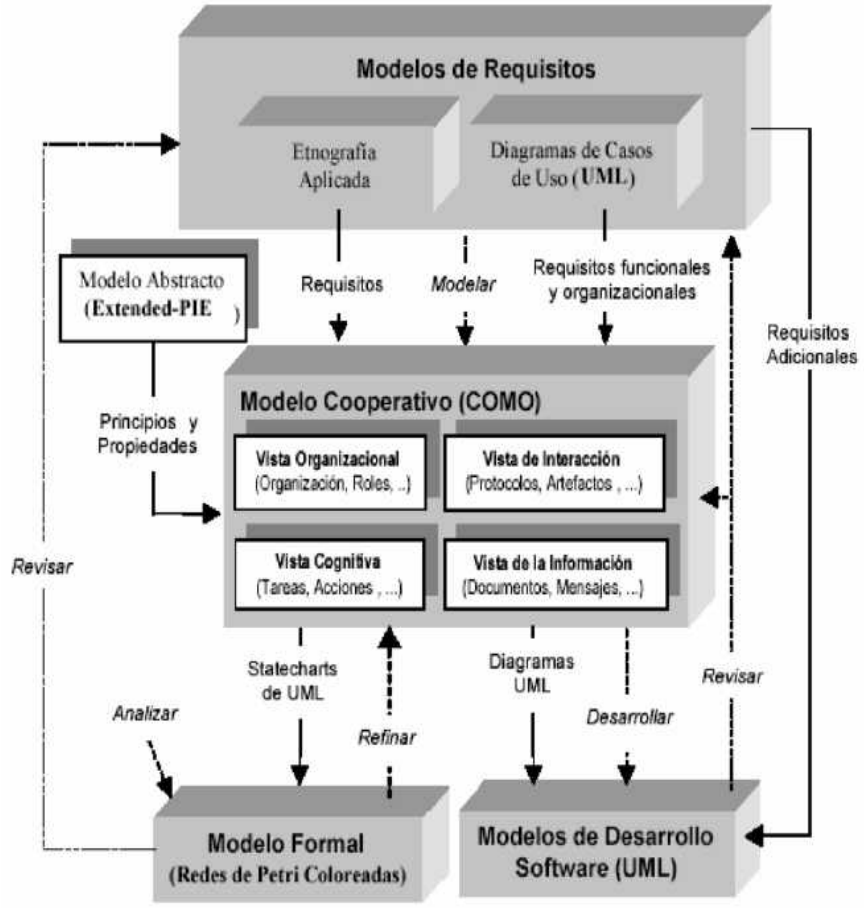


Figura 10. Esquema general de AMENITIES [Garrido, 2003].

  - *Vista de Interacción:* Estudia los procesos que implican un diálogo entre los participantes, estableciendo el modo de diálogo mediante protocolos o los requisitos que se imponen sobre los medios a utilizar.
  - *Vista de Información:* Obtiene toda la información que se comparte sobre el escenario colaborativo, que puede ser implícita (en actividades y acciones) o explícita (como flujo entre actividades).
- **Modelo de Desarrollo Software:** Punto de conexión de la metodología con el desarrollo del software propiamente dicho, haciendo uso de notaciones de UML.

Además, AMENITIES es una metodología para desarrollar *groupware* centrada en un modelo dirigido por arquitecturas, que comprende una ontología de dominio, de la que pueden derivarse ontologías de aplicación [Garrido 07, Noguera 09, Noguera 10]; suministrando modelos independientes para cada sistema.

2.2.3.7. CIAM

CIAM [Molina 07, 09] propone un marco metodológico para el diseño de interfaz de usuario interactiva; que se pueda extender o incluso ser parte de otras metodologías existentes, de tal manera que reduzcan las carencias presentes en el desarrollo de *groupware*. Este marco plantea cuatro vistas para el desarrollo de interfaces de usuario [Molina 07]:

1. **Vista de Organización.** La organización está basada en roles y toma en cuenta conceptos como: *organización*, *rol*, *grupo*, *agente software*, *actor* y *equipo de trabajo*. Las relaciones que se identifican entre los roles son: jerárquicas y generalización. Para abordar el modelado de la organización se emplea una notación análoga a los diagramas de clases de UML.
2. **Vista de Inter-Acción.** Modela la estructura y el flujo de trabajo, donde el concepto principal es la tarea. Una tarea es una actividad desarrollada por actores para alcanzar un cierto objetivo. Esta vista considera conceptos como *proceso CSCW*, *tarea evento*, *interdependencia*, *tarea de trabajo en grupo*, *tarea individual*, *tarea colaborativa* y *tarea cooperativa*. Emplea los operadores temporales del Árbol de Tareas Concurrente (CTT, *Concur Task Tree*) para la representación del flujo de trabajo, además de tablas de participación, diagramas de estados y diagramas de actividad.
3. **Vista de Datos.** Modela los datos manejados durante el trabajo en grupo, considerando los siguientes conceptos: contexto compartido, objeto, atributos, operaciones, objeto compuesto y objeto simple.
4. **Vista de Interacción.** Describe las herramientas y los aspectos interactivos de la aplicación. Maneja los conceptos de: herramienta, herramienta dependiente de la tarea, herramienta independiente de la tarea, interfaz de usuario, tarea interactiva, tareas de interacción, tareas de aplicación, relación entre tareas, interfaz de usuario colaborativa y áreas de visualización. El modelado de esta vista se realiza con CIAN *(Collaborative Interactive Applications Notation)*, una extensión de CTT que incluye tres áreas de visualización:
    a) *Área de visualización colaborativa:* Representa el contexto compartido.
    b) *Área de visualización individual:* Representa la interacción individual de cada miembro del grupo.
    c) *Segmento exclusivo de edición*: Representa el contexto compartido exclusivo para realizar alguna modificación.

La metodología CIAM consta de las siguientes etapas (Figura 11):

- **Modelado del grupo:** Crea el sociograma, que representa la estructura organizacional y las relaciones (herencia, desempeño —entre actores y roles— y asociación —indica si los roles cooperan o colaboran entre sí) que existen entre los diferentes miembros de la misma;
- **Modelado del proceso:** Que puede ser cooperativo, colaborativo y de coordinación, especificando las tareas cooperativas, colaborativas o individuales y definiendo una serie de relaciones entre procesos: dependencia temporal, dependencia de datos, notificación; así como una serie de condiciones: de terminación, de plazo y de ejecución.
- **Modelado de responsabilidades:** Asigna responsabilidades individuales a cada rol.
- **Modelado de tareas de trabajo en grupo:** Define con mayor nivel de detalle las tareas cooperativas (modeladas con grafos de descomposición de responsabilidades) y colaborativas (modeladas mediante la especificación del contexto compartido) identificadas en las etapas anteriores.
- **Modelado de Interacción**: Modela únicamente aspectos de interacción humano-computadora por medio de los árboles de descomposición de tareas interactivas en notación CTT.

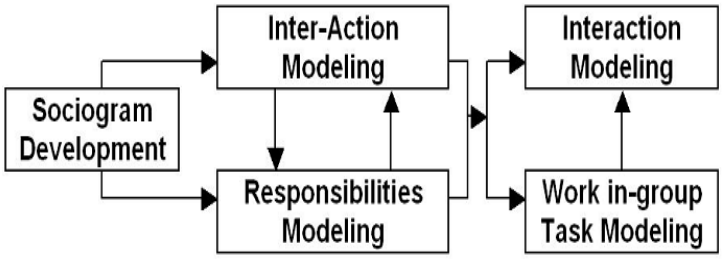


Figura 11. Etapas de CIAM [Molina 07].

2.2.3.8. TOUCHE

TOUCHE [Penichet 07a, 08] propone una metodología para desarrollar interfaces de entornos colaborativos, mediante el uso de modelos integrados al mismo, que definen las necesidades del sistema desde diferentes puntos de vista complementarios llevando a una especificación completa y coherente del sistema colaborativo. Además, plantean una ontología para evitar incoherencias, imprecisiones y ambigüedades relacionadas con los conceptos utilizados en el desarrollo de *groupware*. TOUCHE proporciona los siguientes modelos [Penichet 07b]:

- **Modelo de Organización**. Representa la estructura organizacional de los usuarios de un entorno colaborativo, presentando actores del sistema y relaciones de colaboración entre ellos.
- **Modelo de Tareas**. Especifica acciones individuales o colectivas que deben llevar a cabo los usuarios para alcanzar una meta determinada. Una tarea puede ser abstracta, de aplicación, de interacción o de usuario, estos tipos de tarea fueron definidos en CTT.
- **Modelo de Objetivos**. Representa las metas de los grupos y los fines para los que se llevan a cabo las tareas a cualquier nivel de granularidad.
- **Modelo de Sesión**. Proporciona una vista del sistema en un momento dado, presentando un escenario concreto.

TOUCHE sigue el proceso de desarrollo tradicional de la Ingeniería del Software [Penichet 07a]:

1. **Elicitación de Requisitos.** Esta etapa se realiza en cinco pasos (véase la Figura 12):
   a) Adquisición del conocimiento del dominio del problema;
   b) Identificación de la estructura organizacional y los actores del sistema.
   c) Elicitación de Objetivos del Sistema.
   d) Elicitar los Requisitos.
   e) Priorización de Objetivos y Requisitos.
2. **Análisis.** Ésta se corresponde con el estudio del dominio, para lo cual realizan los siguientes pasos:
   a) Identificación y Descripción de Roles del problema.
   b) Identificación y Descripción de Tareas.
   c) Estructura y Comportamiento.
   d) Trazabilidad Inter- e Intra-etapa.
3. **Diseño.** Presenta el Modelo de Navegación (que representa el camino que toman los usuarios entre las distintas interfaces en función de sus acciones, utilizando un diagrama de interacción de contenedores abstractos) y el Modelo de Presentación (que simboliza las interfaces de usuario con las cuales interaccionan los usuarios, empleando un diagrama de interfaces de usuario abstractas).
4. **Implementación.** Acciones individuales o colectivas que se realizan por parte de los usuarios para alcanzar una meta determinada. Consiste en generar la interfaz de usuario final de acuerdo a los componentes definidos durante el diseño. Es un proceso de reificación de estos componentes, que concretizarán los elementos a emplear según detalles de implementación, plataforma, objetivo, personalizaciones de usuarios, etc.

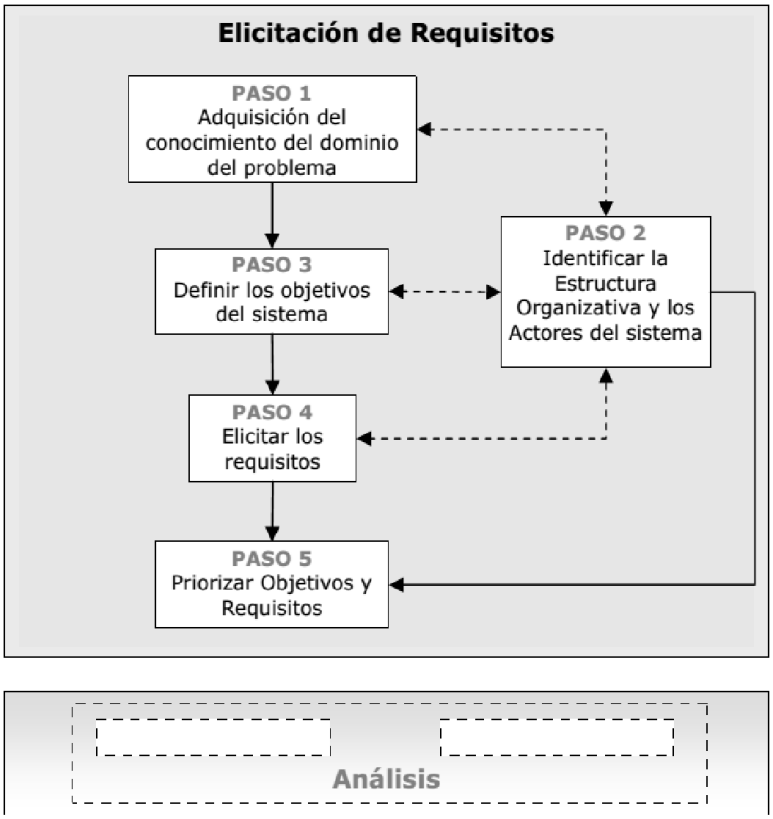


Figura 12. Elicitación de Requisitos de TOUCHE [Penichet 07a].

2.2.3.9. MOLICC

MoLICC [De Souza15] es la extensión de MOLIC **(***Modeling Language for Interaction as Conversation)* para permitir la colaboración entre usuarios basada en el modelo 3C (Comunicación, Colaboración y Coordinación) [Fucks 04]. MOLIC [Barbosa 03] es un lenguaje para el diseño de la interacción basado en la ingeniería semiótica, que percibe la interacción usuario-sistema como una conversación entre el diseñador (sistema) y el usuario. De este modo, permite la representación de la interacción como un conjunto de conversaciones que el usuario puede tener con el diseñador, esperando que éste presente claramente el meta-mensaje. Así el lenguaje, también, sirve como una herramienta epistémica, llevando a los diseñadores a mejorar su comprensión sobre el problema que se va a resolver y el artefacto que se va a crear.

En MOLIC el proceso de diseño comienza con la definición de los roles de usuario, sus objetivos, escenarios de análisis e interacción y los signos utilizados en dichos escenarios. Por tanto, el modelo se crea para representar cómo los escenarios pueden ocurrir, llevando a los usuarios a alcanzar sus metas durante la interacción. En el proceso, el diseñador es llevado a reflexionar sobre el meta-mensaje, creando mejores maneras de resolver problemas de comunicación y explorando alternativas antes de crear la interfaz de usuario final.

En un diagrama MoLIC, el diseñador puede definir los diferentes posibles temas de conversación y las pronunciaciones de los usuarios y diseñadores para tomar el turno, que pueden emitir para avanzar en la conversación hacia una meta o cambiar de tema para lograr otra. La Figura 13 presenta los principales elementos del lenguaje MoLIC, donde: **a** es el punto de apertura; **b** es un acceso ubicuo; **c** es un interés del usuario; **d** es una escena de conversación; **e** es un procesamiento del sistema; **f** es un interés del diseñador, **g** es un interés de ruptura; **h** es su cierre.

MoLLIC para representar la colaboración basada en el Modelo 3C, agrega tres elementos:

- **El Indicador de Mensaje Entrante (*Incoming Message Indicator*, IMI)**: Indica qué mensaje puede recibir el usuario durante su conversación sobre una escena, y las transiciones de IMI indican que cuando el usuario recibe algún mensaje, puede o debe cambiar su tema de conversación actual para atender el mensaje. El IMI está vinculado obligatoriamente a una escena, lo que indica que el usuario recibirá una notificación para algunos mensajes sólo cuando un IMI adjunto a la escena tiene definido ese mensaje.

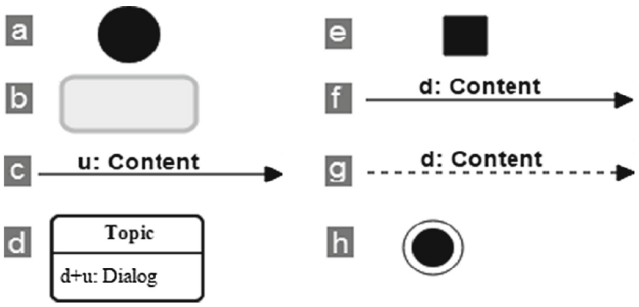


Figura 13. Elementos de MOLIC [De Souza15].

- **El Indicador de Mensaje Saliente (*Outgoing Message Indicator,* OMI):** Indica que cuando el usuario hace una solicitud, el sistema enviará un mensaje a otros usuarios, que se recibirán mediante el IMI. La OMI está obligatoriamente unida a un procesamiento del sistema, indicando que el mensaje a enviar debe ser desempeñado por el sistema, ya sea síncrono o asíncrono.
- **El Indicador de Espacio Compartido (*Shared Space Indicator,* SSI):** Representa un área de espacio compartido para todos los usuarios, dónde ellos son capaces de conocer quiénes más están durante su comunicación con el sistema. Los elementos del SSI son: el texto en la parte superior izquierda, que describe el nombre del espacio, la parte inferior izquierda, que muestra los roles de usuario que pueden participar en ese espacio y; la parte inferior derecha, que muestra los objetos de cooperación que comparten los usuarios dentro del espacio. Las modificaciones en un objeto de cooperación son vistas por todos los usuarios dentro del espacio, no especificando el tipo de acceso sincronizado al modificarlo. La concurrencia para acceder a estos objetos puede cambiar en función de diferentes implementaciones, y el modelo de interacción no necesita ser especificado en el diseño de la fase de interacción. Los elementos de expresión para interactuar con los objetos de cooperación se definen por el propio espacio SSI y las conversaciones que el usuario puede tener con el diseñador dentro del espacio.

2.2.3.10. Meta-modelo genérico

El meta-modelo genérico consiste de clases abstractas (véase la Figura 14), que describen aplicaciones cooperativas centradas tanto en ayudar a los trabajadores a organizar su trabajo y hacerlo (*groupware*), como en mantener la cooperación en casos de uso de ocio o no profesional (redes sociales) [Jeners 12]. Cada sistema implementa cada clase abstracta y su herencia.

Este meta-modelo, primero define las acciones elementales que el usuario puede realizar en el sistema existente. Luego se categorizan éstas para generalizarlas en el meta-modelo del sistema. A continuación se instancian las clases abstractas con las técnicas aplicadas en dicho sistema.

El meta-modelo presenta tres superclases: acción, objeto y sistema.

- **Acción.** Denota todas las actividades y procesos realizados por los usuarios del sistema o por el propio sistema. Por tanto, la clase Acción está especializada en dos subclases actividad y proceso. La primera es realizada por personas que tienen un rol específico y la segunda puede ser ejecutada por el propio sistema, controlada por una regla. Ambas subclases pueden transformar (crear, modificar y destruir) objetos específicos.
- **Objeto.** Representa todas las entidades virtuales: persona, contenedor, artefacto y atributo del sistema.

- *Persona.* Representa a cada actor humano en el sistema, es decir, el usuario, especificado por los roles.
- *Contenedor.* Colección de objetos. Pueden contener contenedores propios, personas y artefactos.
- *Artefacto.* Denota las entidades básicas del sistema, tales como mensajes, archivos o cualquier otro objeto con el que los usuarios puedan interactuar.
- *Atributo.* Pertenece a objetos y no pueden permanecer solos. Están siempre unidos a otros objetos (persona, contenedor y artefacto) Los dos atributos ya implementados son el rol y la regla. Los primeros especifican a la persona en términos de las actividades que pueden llevar a cabo, estableciendo cuatro tipos de roles:
    1. Organizacionales: Describen a la gente en el contexto de su jerarquía de trabajo, por ejemplo, jefe, colega y socio.
    2. De Actividad: Especifican las actividades que una persona puede realizar, por ejemplo, autor, lector.
    3. De Derechos: Muestran los derechos que una persona puede tener, por ejemplo: gerente, propietario, etc.
    4. El último tipo de roles son los roles culturales.
- Las reglas son expresiones que pueden adherirse a los objetos y si esta regla es válida, se realizará un proceso. Un ejemplo es un mecanismo de notificación en espacios de trabajo compartidos.

➢ **Sistema.** Contiene todos los sistemas representados en el meta-modelo. Cada clase sistema implementada pertenece a uno o varios sistemas.

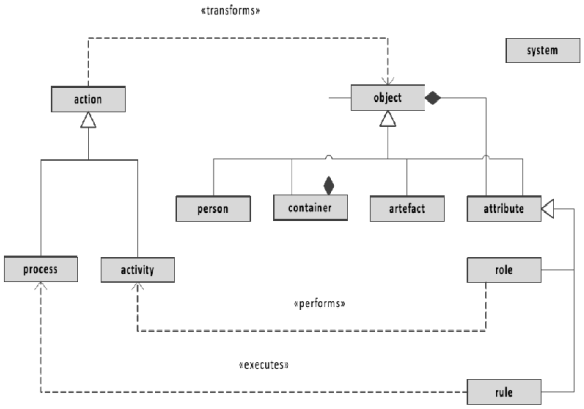


Figura 14. Meta-modelo de sistemas cooperativos [De Souza 15].

2.2.3.11. Meta-modelo de entornos virtuales

Este meta-modelo [Jeners 13] intenta modelar los entornos virtuales, que consisten de sistemas cooperativos, redes sociales, así como de sistemas de información personales y empresariales. Por tanto, busca un modelo unificado de dichos entornos, aplicando los conceptos utilizados tanto en *groupware* como en redes sociales.

El meta-modelo emplea una notación extendida del modelo entidad-relación, pero centrado en cubrir el modelo mental de los usuarios, a partir de la perspectiva de la interfaz de usuario mientras trabajan con el sistema y usan sus funciones. Además parte de la idea, que un sistema cooperativo se caracteriza por personas y artefactos, así como por la comunicación entre las personas y las actividades realizadas por las mismas sobre los artefactos [Dix 03]. Por tanto, dicho modelo propone que un sistema cooperativo (C) es modelado por un cuádruple $C = \{P, A, Rel, Rul\}$ compuesto por un conjunto de Personas (P), Artefactos (A), Relaciones (Rel) y Reglas (Rul).

El conjunto $P = \{p_1, p_2, \ldots, p_n\}$ representa personas físicas que utilizan un sistema (usuarios). Las instancias de $p_1, p_2, \ldots p_n$ son personas simples pero representantes de un tipo de persona, llamado rol. Un

rol contiene ciertas características, tal como el derecho a realizar determinada actividad (autor, administrador, gerente y propietario).

Todo lo que no es una persona pertenece al conjunto de artefactos $A = \{a_1, a_2, \ldots, a_n\}$. Las instancias de $a_1, a_2, \ldots, a_n$ pueden ser todos los artefactos de trabajo que están virtualmente representados por el sistema y el propio sistema.

El conjunto $Rel = \{r_1, r_2, \ldots, r_n\}$ incluye tres tipos de relación:

- **Social.** Relación análoga entre dos personas, que establece una estructura de personas. Esta relación puede ser simétrica ("Alice es una amiga de Bob"), o asimétrica ("Alice está siguiendo a Bob"). Las relaciones sociales entre los roles se heredan a las personas individuales.
- **Estructural.** Relación entre dos artefactos, que establece una estructura de contenido. Es decir, estructura los artefactos relacionándolos uno con otro, p. "Una carpeta contiene un archivo".
- **Derechos**. Cada entidad (persona, artefacto) puede poseer el derecho de cambiar, ya sea una entidad o una relación. Una persona tiene derechos particulares para realizar ciertas actividades sobre los artefactos. Las personas y los artefactos pueden tener el derecho de cambiar una relación social (convertirse en amigo de otra persona), una relación estructural (guardar un archivo en una carpeta determinada) o un derecho mismo (invitar a una persona como editor a un documento). Los derechos de artefactos se presentan en términos de notificaciones y otros tipos de procesos realizados por el sistema.

El conjunto de reglas $\mathscr{R} = \{g_1, g_2, \ldots, g_n\}$ modela las dependencias de las relaciones. Los derechos pueden derivarse de los roles de las personas, las relaciones sociales o las relaciones estructurales. Las relaciones sociales determinan los derechos (un amigo de un autor tiene el derecho de leer los artefactos creados por el autor). Las relaciones estructurales determinan derechos (cada miembro de una carpeta compartida tiene acceso a todas las subcarpetas y archivos).

El modelo se centra principalmente en el contexto interno de un sistema cooperativo, pero está enmarcado en la interfaz de usuario gráfica para denotar la importancia de la misma. Además, una interfaz apropiada es esencial para proporcionar características de sistemas cooperativos, como por ejemplo, presencia y conciencia, y hacerlos accesibles a los usuarios.

Este meta-modelo considera dos aspectos importantes de los sistemas cooperativos: las relaciones entre personas o red de personas, que establece la estructura organizacional o social de estos sistemas, y el intercambio de información, que establece la base de las actividades que realizan las personas.

- **La relación entre personas.** La cooperación siempre implica una relación entre las personas (que pueden estar distribuidas o co-ubicadas de manera explícita) implicadas en la misma. Esta relación se asigna a la lógica de aplicación del sistema. El meta-modelo identifica tres tipos básicos de relación: amigos, seguidores y grupos (derivados de la teoría de grafos y una perspectiva de análisis de redes sociales).
- **El intercambio de información.** Independientemente de realizar comunicación, cooperación o coordinación, cada actividad puede ser reducida a pasos de intercambio de información. Dentro de este concepto, el meta-modelo diferencia dos sub-conceptos: Enviar y compartir. Éstos existen en diferentes formas y pueden caracterizarse por varias dimensiones.
  - **Reciprocidad.** Indica si una comunicación es bilateral (conversación) o en solo sentido (difusión).
  - **Direccionalidad.** Indica si un mensaje está dirigido a una persona o no.
  - **Visibilidad.** Indica si una conversación es visible para dos personas o un grupo (privado) o visible para todo el mundo (público).
  - **Cardinalidad.** Indica cuántas personas están involucradas en el intercambio de información.

2.2.3.12. Modelo de clasificación socio-técnico

El modelo de clasificación socio-técnico [Cruz 12] intenta abordar la evidente falta de consenso respecto a la estructura conceptual del trabajo cooperativo y de *groupware* en una perspectiva combinada, que comprende los requisitos técnicos y las dimensiones de trabajo en un modelo de clasificación unificado. Por ello, el modelo categoriza los requisitos de colaboración para un desarrollo de *groupware* con una orientación más social.

Los elementos taxonómicos del modelo se basan en la literatura genérica CSCW y de grupo, tomando en cuenta su persistencia temporal, impacto bibliométrico, complementariedad y consistencia lógica. Los "bloques" y "metabloques" de este modelo establecen un conjunto de dominios según su granularidad, estando estructurados de forma jerárquica (véase la Figura 15).

1. **El modelo 3C.** Es la primera categoría, que es sistematizada en un ciclo interactivo a través de los modos de colaboración conocidos.
   a. *La comunicación.* Proceso de interacción entre las personas [MacGrath 84], que implica un intercambio de información explícita o implícita, en un canal privado o público. Los mensajes de los usuarios pueden ser identificados o anónimos, y la conversación puede ocurrir sin soporte, con soporte de proceso estructurado e intelectual, o con protocolos asociados. Como requisito, el *groupware* debe ser capaz de soportar la conversación entre dos o más personas, en una configuración uno a uno, uno a muchos o muchos a muchos.
   b. *La coordinación.* Gestiona las interdependencias entre las actividades ejecutadas por múltiples actores, que se basan en los objetos comunes intercambiados en las actividades [Malone 94]. Algunas categorías relacionadas con la coordinación son: planificación, modelos de control, relación tarea/subtarea, gestión de información, ajuste mutuo, estandarización, protocolo de coordinación, modos de operación, etc. Con el fin de apoyar eficazmente la coordinación, *groupware* necesita cumplir con tres requisitos importantes: gestión del tiempo, de recursos o artefactos compartidos producidos a lo largo de las actividades.
   c. *La cooperación.* Los acuerdos de trabajo cooperativo aparecen y desaparecen [MacGrath 84]. La cooperación se produce cuando un grupo trabaja hacia un objetivo común [Malone 94] con altos grados de interdependencias de tareas, compartiendo la información disponible por algún tipo de espacio compartido [Grudin 94]. Las categorías de cooperación pueden abarcar desde la producción (co-autoría), almacenamiento o manipulación de un artefacto, hasta la concurrencia, acceso o *floor control*. Tecnológicamente, la cooperación es apoyada por sistemas con capacidad para enviar o recibir mensajes, de forma síncrona y/o asincrónicamente [Mittleman 08, Mentzas 93], y también, de desarrollar o compartir documentos [Grudin 97], que se identifican como requisitos en este modelo.
2. **Tiempo/Espacio.** La colaboración ocurre en un tiempo (síncrono y/o asincrónico) y espacio (co-localizado o remoto) específico, presentando altos o bajos niveles de predictibilidad.
3. **Granularidad.** Si granulamos la categoría Tiempo/Espacio, se puede tener un conjunto de subdominios: persistencia de sesión, retardo entre canales de audio/vídeo, reciprocidad y homogeneidad de canales, retraso del mensaje enviado y espontaneidad de colaboración. Complementariamente, puede ser útil definir temas contextuales para mejorar la dinámica de trabajo.
4. **Conciencia.** Los mecanismos de conciencia de grupo son esenciales en los sistemas de colaboración para reducir las pérdidas de trabajo La conciencia de grupo es la percepción del grupo sobre lo que cada miembro desarrolla y el conocimiento contextual que tienen acerca de lo que está sucediendo dentro del grupo. La conciencia está caracterizada por el espacio y atmósfera, actividad, objeto, ser humano y meta-dimensiones como la presencia, influencia y habilidades.
5. **Nivel de aplicación.** Identifica un conjunto de tipologías para sistemas de *groupware*. En esta categoría se consideran las siguientes subcategorías:

a. *Regulación.* Significa la representación de mecanismos que permiten a los participantes organizarse en un entorno compartido, donde la regulación de las actividades de colaboración se refiere a la definición y evolución de normas de trabajo para asegurar la conformidad entre la actividad y los objetivos grupales. Cabe destacar que algunas de las dimensiones de la regulación obtenidas de la literatura son: localización, actores (roles, lugares y posiciones); herramientas (regulativas o no); roles (temáticos o causales); reglas (restricciones, normas o de trabajo); tipos de interacción; escenarios interactivos; y objetos (medios de comunicación y productos de la colaboración).
b. *Propiedades de groupware.* Pueden ser constituidas por propiedades funcionales de herramientas de colaboración: arquitectura, propiedades funcionales y de calidad, soporte de procesos de grupo, interfaz de colaboración (portal, dispositivos o espacio de trabajo físico), relaciones (colección, lista, árbol y gráfico) funcionalidad central, contenido (texto, enlaces, gráfico o flujo de datos), acciones soportadas (recibir, añadir, asociar, editar, mover, eliminar o juzgar), identificabilidad, control de acceso, mecanismos de alerta, componentes de software inteligentes/semi-inteligentes, indicadores de conciencia y plataforma.
c. *Elementos GDSS* (*Group Decision Support Systems, Sistemas de Soporte de* Decisión *de Grupo*). Pueden incluir hardware, software, organización y soporte de personas.

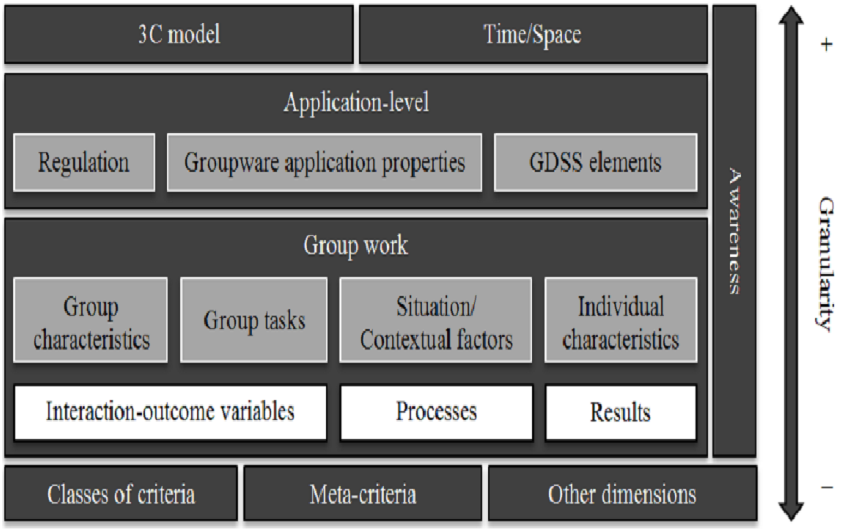


Figura 15. Modelo de clasificación socio-técnico [Cruz12].

6. **Grupo.** Pueden ser definidos como "agregaciones sociales de individuos" con conciencia de su presencia, conducidos por sus propias normas y apoyados por interdependencias de tareas hacia un objetivo común en un propósito o contexto de trabajo compartido [Pumareja 02]. En este sentido un grupo cuenta con:
   a. *Características.* Tales como: tamaño (3 a 7,> 7), composición, ubicación, proximidad, estructura (liderazgo y jerarquía), formación, conciencia grupal (baja o alta y cohesividad), comportamiento (cooperativo o competitivo), autonomía, sujeto y confianza. Los miembros del grupo tienen experiencia (personal, de trabajo, capacitación y educación), habilidades, motivación, actitud hacia la tecnología, experiencia previa, satisfacción, conocimiento y personalidad.
   b. *Tareas.* Estas pueden subdividirse en: creatividad, planificación, intelecto, toma de decisiones (elección, evaluación y análisis, búsqueda, informe y encuesta), conflictos cognitivos, motivos mixtos, concursos/batallas/competencia y performances/ psicomotor, teniendo una complejidad específica asociada a cada tarea [MacGrath 84]. Las subcategorías pueden ser soportadas por impacto cultural, objetivos, interdependencia o necesidades de intercambio de información, cuellos de botella o procesos de ganancia y pérdida.

c. *Factores contextuales o situacionales.* Pueden ir desde el apoyo organizacional (recompensas, presupuesto y capacitación), contextos culturales (confianza o equidad), entorno físico, ambiente (competencia, incertidumbre, presión de tiempo y evaluación) y dominio de negocios de manera organizacional.
d. *Variables de interacción.* Están relacionadas con factores de grupo:
    i. *Variables de interacción de resultado.* Tales como resultados de grupo (calidad del desempeño del grupo, procesos de colaboración y desarrollo de grupo), resultados individuales (expectativas y satisfacción en el uso del sistema, apreciación de la pertenencia al grupo y los problemas individuales en el uso del sistema) y resultados del sistema (mejoras y asequibilidad);
    ii. *Procesos*: Incluyendo las dimensiones individuales, de interpretación, de motivación y de desempeño;
    iii. Resultados. Específicamente premios individuales, vitalidad grupal y resultados organizacionales.

7. Las variables independientes se centran en clases de criterios (funcional, técnico, usabilidad y ergonomía), meta-criterios (escalabilidad y ortogonalidad) y dimensiones complementarias sin un dominio específico. Algunas de las otras dimensiones que pueden caracterizar un escenario de colaboración socio-técnica son: acoplamiento de trabajo, tareas y metas compartidas, riqueza y tipo de información, centralización de control, actividades, división del trabajo, patrones, técnicas, scripts, asistencia, monitorización del aprendizaje, grado de interacción, aserción, eventos, estrategia, conectividad social, gestión de contenidos, integración de procesos, compartir (visión/opinión, conocimiento/información y trabajo/operación), protección, pérdida de procesos distribuidos o profundidad de la mediación.

2.2.3.13. MoCA

MoCA [Lee 15] es un *framework* extendido para describir situaciones y ambientes colaborativos complejos, que trata de abarcar también las redes sociales e infraestructuras a gran escala y de largo plazo dentro del ámbito de *groupware*. Para ello, este modelo se centra en la "Acción Coordinada" de tal manera que la acción realizada por el grupo para alcanzar una meta sea tanto en un entorno de trabajo formal (*groupware* clásico) como de ocio (red social). Para MoCA la acción coordinada es el núcleo de CSCW y está constituida por la interdependencia de dos o más actores que, en sus actividades individuales, están trabajando hacia una meta particular a través de uno o más campos de acción solapados. Al centrar este modelo en la colaboración, se extiende el espacio de investigación, cómo múltiples tecnologías pueden ser asignadas a una sola acción colaborativa: una agregación socio-técnica de actores y tecnologías.

MoCA consta de siete dimensiones de acción coordinada.

1. **Sincronización**. Parte de la matriz original de Johansen sobre reuniones electrónicas [Schmidt 13]. Sincronización se refiere a una acción coordinada continua que se realiza de forma síncrona (al mismo tiempo) hasta asíncrona (diferente tiempo). La continuidad de esta dimensión permite acciones coordinadas para ser una mezcla de asincrónico y síncrono, en la cual la apreciación de la previsibilidad del tiempo, o falta de ella, está implícita en las dimensiones descritas más adelante.
2. **Distribución física.** Tomada también de la matriz original de Johansen, es un continuo que se refiere a si las acciones coordinadas tienen lugar en la misma ubicación geográfica (en un extremo del continuo) o en ubicaciones geográficas completamente diferentes (en el otro extremo). Al igual que con la sincronización, en esta dimensión la apreciación de la previsibilidad, o falta de ella, está implícita en las dimensiones descritas más adelante. La naturaleza de algunos tipos de trabajo (fuerte o débilmente acoplado) se ven afectados por la distancia incluso a pesar de los avances tecnológicos.

3. **Escala.** Es el número de participantes involucrados en la colaboración. Se debe considerar que un aumento en los participantes requiere escalar los mecanismos de coordinación para soportar organizaciones sociales complicadas y nuevas.
4. **Número de comunidades de práctica.** NCoP (*Number of Communities of Practice*) se refiere a la diversidad en normas, prácticas, herramientas y lenguaje. Hacia un extremo de la dimensión se encuentran equipos pequeños y homogéneos compuestos por personas con antecedentes y formación similares; en el otro, se encuentran equipos grandes y heterogéneos compuestos por personas con antecedentes y formación diferentes.
5. **Naciente.** Se refiere a acciones coordinadas no establecidas (nuevas) contra. establecidas (antiguas). Naciente no es sólo un sinónimo de "novedad", sino que implica "presentar" cambios inesperados. En este caso, para soportar este tipo de cambios en la colaboración, es necesario crear y usar artefactos temporales no disponibles.
6. **Permanencia planificada.** Se refiere a la permanencia prevista o planificada de una acción coordinada, es decir, la colaboración será de corto o largo plazo. Una colaboración temporal no es necesariamente "más fácil" que una colaboración permanente. Mientras que una permanente tiene un alto costo en la planificación y creación de las prácticas y herramientas administrativas así como de coordinación necesarias para las colaboraciones a largo plazo, las colaboraciones temporales deben esforzarse debido a la falta de argot, métodos, prácticas y comprensión común.
7. **Volumen de negocios.** Se refiere a la rapidez con que los participantes entran y salen en una colaboración. Dentro de esta dimensión están implícitas colaboraciones que van desde colaboraciones cerradas y privadas donde los participantes parten lentamente, o en todo caso, colaboraciones totalmente abiertas y públicas donde "las masas" pueden participar.

2.2.3.14. Conclusión de *groupware* basado en modelos

De los trece modelos revisados, cinco (Análisis de Tarea, Modelo Conceptual, Teoría de la Coordinación, de la Cognición Distribuida y de la Actividad) son teóricos, otros cinco (MOLLIC, Meta-modelo, Modelo de Clasificación Socio-Técnico, Meta-modelo de Entornos Virtuales y MoCA,) especifican *groupware* per se y de red social, y tres (AMENITIES, TOUCHE y CIAM) son metodologías.

Estos modelos se analizaron desde una perspectiva general, sus características, la colaboración, la coordinación y el entorno. En el primero (véase la Tabla 11) se considera su principal contribución, la especificación utilizada y los pasos utilizados para aplicar el modelo.

La contribución de los modelos examinados se centra en sustentar la especificación del desarrollo de *groupware,* pero se diferencian en la manera en que lo consiguen. Los cinco primeros modelos revisados son especificados teóricamente, AMENITIES semi-formalmente (utilizando redes de Petri coloreadas) y el resto informalmente. Además, tres (AMENITIES, TOUCHE y CIAM) se basan en modelos de tareas, otros tres (MOLIC, modelo de clasificación socio-técnico y MoCA) en el modelo 3C, cuatro (AMENITIES, TOUCHE, CIAM y Meta-modelo) en UML y el Meta-modelo de entornos virtuales en el modelo entidad-relación. Solamente AMENITIES, TOUCHE y CIAM establecen explícitamente los pasos que se deben seguir para desarrollar *groupware*.

En la Tabla 12 se consideran las características tomando en cuenta los elementos que los constituyen y el producto obtenido. Tanto los elementos como el producto obtenido difieren en todos los modelos analizados, debido a que ambos corresponden al enfoque y sustento de cada modelo.

Tabla 11. **Perspectiva general de *groupware* basado en modelos.**

NOMBRE	PRINCIPAL CONTRIBUCIÓN	ESPECIFI_ CACIÓN	PASOS
Teoría de la Coordinación	Identifica y estudia inter-dependencias comunes y sus mecanismos de coordinación.	Conceptual Teórica	1) Registra la descripción de las actividades. 2) Identifica inter-dependencias. 3) Establece la taxonomía de inter-dependencias y mecanismos de coordinación.
Teoría de la Cognición Distribuida	Suministra un entendimiento de las interacciones entre personas y tecnologías.	Conceptual Teórica	1) Distribución de procesos cognitivos a través de los miembros de grupos sociales; 2) Coordinación entre la estructura interna y externa; 3) Forma en que los eventos anteriores transforman a los posteriores.
Análisis de Tarea	Estudia la interacción de usuarios a través de la revisión de las tareas realizadas.	Conceptual Teórica	Se centra en la estructura de la tarea, que considera su flujo y la relación de dependencia entre ellas, a partir de dicha estructura se crean las interfaces de usuario.
Teoría de la Actividad	Framework que estudia diferentes formas de praxis humana.	Conceptual Teórica	Usa herramientas para tratar un objeto y alcanzar meta en una comunidad mediada por reglas. La comunidad transforma el objeto conforme a la división del trabajo.
Modelo Conceptual	Especifica tres modelos: ontológico, coordinación e interfaz de usuario.	Conceptual Teórica	Describe objetos y sus operaciones, considerando acceso y derechos de operación. Establece como el grupo lleva a cabo su trabajo. Describe la interfaz de usuario.
AMENITIES	Metodología que considera y define conceptos relevantes para un sistema colaborativo.	Semiformal, MT, CPN y UML	1) Análisis y obtención de requisitos. 2) Modelado del sistema cooperativo. 3) Análisis del modelo cooperativo. 4) Diseño del sistema.5) Desarrollo del software.
NOMBRE	**PRINCIPAL CONTRIBUCIÓN**	**ESPECIFI_ CACIÓN**	**PASOS**
CIAM	Marco metodológico para el diseño de interfaz de usuario interactiva.	Informal, CTT, UML y CIAN	1) Modelado del grupo, 2) Modelado del proceso, 3) Modelado de responsabilidades, 4) Modelado de tareas de trabajo en grupo, 5) Modelado de interacción.
TOUCHE	Metodología para interfaces de entornos colaborativos.	Semiformal, CTT y UML	1. Elicitación de Requisitos. 2. Análisis. 3. Diseño.4. Implementación.
MoLICC	Modela sistemas cooperativos, percibiendo la interacción usuario-sistema como una conversación entre ellos,	Informal, MOLIC	Se definen roles, objetivos, escenarios de análisis e interacción. El modelo representa cómo los escenarios ocurren, llevando a los usuarios a alcanzar sus metas; identificando y agregando colaboraciones entre usuarios.
Meta-modelo	Modela sistemas cooperativos tanto de *groupware* como de redes sociales.	Informal, UML	1) Define las acciones del usuario. 2) Categoriza éstas para generalizarlas. 3) Instancia las clases abstractas con las técnicas aplicadas en dicho sistema.
Meta-modelo Entorno Virtual	Modela sistemas cooperativos y de información.	Informal, modelo entidad-relación	Especificar los elementos que constituyen un sistema cooperativo: conjunto de Personas (P), Artefactos (A), Relaciones (Rel) y Reglas (Rul).
Modelo de Clas. Socio-Técnico	Categoriza los requisitos para *groupware* social.	Informal, Taxonómico.	Se especifican cada una de las categorías que constituyen al Modelo de Clasificación Socio-Técnico.
MoCA	Framework extendido para describir *groupware*.	Informal, Taxonómico	Especificar las siete dimensiones establecidas por MoCA.

Tabla 12. **Características de *groupware* basado en modelos.**

NOMBRE	Elementos	Producto
Teoría de la Coordinación	Análisis de la coordinación, tipología de interdependencias y mecanismos de coordinación.	Flujo de recursos entre actividades y dependencias resultantes
Teoría de la Cognición Distribuida	La actividad cognitiva se hace con recursos y los significados de las acciones se basan en el contexto de la actividad.	La coordinación con el comportamiento dinámico del material de trabajo.
Análisis de Tarea	Agente es quién realiza la tarea. *Trabajo* relacionado a la estructura de la tarea. *Situación* detecta y describe el entorno y los objetos que lo componen.	A partir de la estructura de la tarea pueden crearse las pantallas centrales y la configuración de navegación de la aplicación.
Teoría de la Actividad	La teoría considera el contexto de una actividad, que incluye varios elementos que determinan su estructura.	Cuando una actividad se ejecuta, el sujeto realiza un proceso cognitivo para hacer las operaciones.
Modelo Conceptual	Define tres modelos: ontológico, coordinación e interfaz de usuario.	El trabajo en grupo caracterizado por tres modelos complementarios.
AMENITIES	Modelo de Requisitos (etnografía y casos de uso). Modelo Formal (RPC). Modelo Cooperativo (COMO). Modelo de Desarrollo Software (UML).	Considera y define conceptos relevantes para analizar y diseñar un sistema cooperativo.

NOMBRE	Elementos	Producto
CIAM	Cuatro vistas: De organización, Inter-Acción, Datos e Interacción.	Un marco metodológico que se pueda extender o incluso ser parte de otras metodologías existentes.
TOUCHE	Contiene los modelos de: Organización, Tarea, Objetivos y Sesión.	Integra modelos, obteniendo una especificación completa y coherente del sistema colaborativo.
MoLICC	Punto de apertura, acceso ubicuo, interés del usuario y diseñador, escena de conversación, procesamiento del sistema, interés de ruptura, cierre, IMI, OMI y SSI.	Permite la representación de la interacción como un conjunto de conversaciones.
Meta-modelo	Tres clases de nivel superior del meta-modelo: acción, objeto y sistema.	Un enfoque para modelar cualquier tipo de aplicación colaborativa.
Meta-modelo Entorno Virtual	Un conjunto de Personas (P), Artefactos (A), Relaciones (Rel) y Reglas (Rul).	Modela los entornos virtuales de manera completa.
Modelo de Clasificación Socio-Técnico	Los bloques y metabloques del modelo establecen un conjunto de dominios según su granularidad, que son estructurados de forma jerárquica.	Modelo de clasificación unificado que subsane la falta de consenso de la estructura conceptual del trabajo cooperativo y de *groupware*.
MoCA	Sincronización, distribución, escala, naciente, vol. de negocios, permanencia. comunidades de práctica, y	La acción coordinada es el núcleo de CSCW y está constituida por interdependencia de los actores.

En la tabla 13 se aprecia que los modelos analizados gestionan la interacción de manera explícita, obviamente, la llevan a cabo de manera diferente; mientras la notificación es realizada de forma implícita, aplicándola de modo distinto. En cuanto a los conceptos que manejan para la colaboración, existen muchas similitudes y diferencias, que les permiten ser utilizados en diversos contextos. Una ontología para disminuir la diversidad e incluso ambigüedad de los conceptos, se presenta en TOUCHE. La metodología propuesta en esta tesis, toma como base de conocimiento varios de los conceptos aportados por los trabajos analizados para desarrollo de *groupware* basado en modelos.

Tabla 13. **Colaboración en *groupware* basado en modelos.**

Nombre	CONCEPTOS	Manejo de Interacción	Notificación
Teoría de la Coordinación	Meta, actividad, actor, recurso e interdependencia.	Gestión de dependencias entre actividades.	Taxonomía de dependencias.
Teoría de la Cognición Distribuida	Actor, actividad cognitiva, recurso, significado de la acción, comunidad.	El significado de la acción se basa en el contexto de la actividad.	Coordinación entre la estructura interna y externa.
Análisis de Tarea	Agente, trabajo, situación, tarea, actividad, entorno, rol, organización/grupo, estructura tarea e interfaz de usuario.	La revisión de las actividades realizadas por una organización o grupo.	El flujo de tarea y de datos por la relación de dependencia entre tareas.
Teoría de la Actividad	Objeto, actividad, acción, operación, estructura interna y externa.	Actividades (estructura externa), operaciones (estructura interna).	Acciones que el grupo realiza en relación a roles.
Modelo Conceptual	Objeto, operación, atributo, semántica, tarea, derecho de operación y acceso, fase, coordinación secuencial, paralelo, aditivo concurrente y totalmente concurrente, contexto, conciencia y memoria de grupo.	Contempla cuatro niveles de simultaneidad: secuencial, paralelo, aditivo concurrente y totalmente concurrente.	Proceso de notificación para actualizar los objetos compartidos que son accedidos por los participantes.
AMENITIES	Usuario, rol, entorno, ley, tarea, estructura organizacional, vista, diálogo, protocolo, información, actividad, acción, escenario, capacidad, desempeño, contexto, conciencia y memoria de grupo.	Establece el modo de diálogo mediante protocolos o requisitos que indican los medios a utilizar.	Las relaciones entre usuarios dependen de sus capacidades y leyes que determinan su comportamiento.
CIAM	Organización, rol, grupo, agente, actor, equipo, proceso, interdependencia, evento, tarea, operación, atributo, contexto compartido, objeto, herramienta, interfaz de usuario y área de visualización.	A través del sociograma y vista de interacción que incluye: a) Área de visualización. b) Área de visualización individual. c) Segmento exclusivo de edición.	A través de la Vista de Interacción.
TOUCHE	Actor, usuario, rol, meta, estructura, entorno, sesión, acción, tarea, modelo de navegación y presentación.	A través del Modelo de Navegación.	Por medio de estructura y comportamiento de la fase de análisis.
MoLICC	Punto de apertura; acceso ubicuo; escena de conversación; procesamiento del sistema; interés del usuario, del diseñador; de ruptura; cierre; IMI; OMI y SSI	A través del IMI y el OMI.	Notificación para mensajes con un IMI adjunto a la escena, tiene definido ese mensaje

Nombre	CONCEPTOS	Manejo de Interacción	Notificación
Meta-modelo	Acción, actividad, proceso, rol, persona, contenedor, regla, objeto, artefacto, atributo y sistema.	A través de la clase acción.	A través de las reglas
Meta-modelo Entornos Virtuales	Persona, artefacto, relación, regla, rol, interfaz de usuario gráfica, enviar, compartir e intercambiar información.	A través del intercambio de información	A través de reciprocidad, direccionalidad, visibilidad y cardinalidad.
Modelo de Clasificación Socio-Técnico	Modelo 3C, conciencia, factor contextual o situacional, nivel de aplicación, tarea, granularidad, regulación, propiedades *groupware*, característica GDSS, variable de interacción, criterio y metacriterio.	Las categorías 3C, Grupo, tiempo/Espacio, Granularidad, Conciencia y Variables de Interacción permiten gestionar la interación del *groupware*.	La interacción se gestiona con: 3C, Grupo, Granularidad, Conciencia y Variables de Interacción.
MoCA	Sincronización, distribución física, escala, número de comunidades de práctica, naciente, permanencia planificada y volumen de negocios.	Establecida por las dimensiones de sincronización, distribución física, escala, número de comunidades de práctica, y volumen de negocios.	Establecida por la sincronización, distribución física, permanencia y volumen de negocios.

En la tabla 14 se considera que los modelos revisados se contextualizan, principalmente, sobre el grupo, aunque los modelos presentados antes del 2010 se centran en el grupo/comunidad clásico o per se, en el cual la idea central es realizar trabajo formal y planeado, mientras que después de ese año los modelos se orientan en especificar reuniones informales o de ocio grupo/comunidad social. También, se contextualiza en el proceso (Teoría de la Coordinación), la actividad (Teoría de la Cognición distribuida) y la situación (Análisis de Tarea). Todos los modelos tienen tanto política de manejo de sesión como mecanismo de concurrencia, que es soportado de diferente forma y presentado de manera explícita e implícita.

Tabla 14. **Coordinación en *groupware* basado en modelos.**

Nombre	Contexto	Mecanismo de Concurrencia	Política de Manejo de Sesión
Teoría de la Coordinación	Proceso	Mecanismo de coordinación conforme a dependencias.	Modelos de tareas y aspectos dinámicos para que un usuario pase de un rol a otro.
Teoría de la Cognición Distribuida	De la actividad	Distribución del proceso cognitivo a través del grupo social; coordinación entre estructura interna y externa.	Distribución de los procesos cognitivos a través de los miembros de los grupos sociales.
Análisis de Tarea	Tarea	La coordinación entre la estructura interna y externa.	La asignación de tareas determina como se organizan los usuarios, de acuerdo al rol.
Teoría de la Actividad	Comunidad	Reglas que determinan la forma de actuar del sujeto en una comunidad.	División del trabajo, acciones que el grupo realiza en relación con los roles del sujeto.
Modelo Conceptual	Grupo	Dos niveles de coordinación (actividad y objeto) y concepto de fase.	El modelo de coordinación describe las actividades (y su orden) que realizan los usuarios.
AMENITIES	Grupo	Identificando de cada tarea su naturaleza cooperativa, mecanismo de activación y modo de sincronización.	Un modelo conceptual para describir y representar un sistema cooperativo.
CIAM	Grupo	A través de la Vista de Inter-Acción, Modelado de proceso y tareas.	A través de la Vista de Organización y del Modelado de responsabilidades.
TOUCHE	Grupo	Por medio del Modelo de Tareas.	A través del Modelo de Organización
MoLICC	Grupo	La concurrencia cambia en función de diferentes implementaciones.	En un área de espacio compartido, cada usuario conoce quiénes están en el mismo.
Meta-modelo	Grupo	A través de la clase acción.	A través de los roles.
Meta-modelo Entornos Virtuales	Grupo	A través del subconcepto compartir.	A través de la relación entre personas
Modelo de Clasificación Socio-Técnico	Grupo per se y social	La categoría 3C a través de la subcategoría *cooperación* permite establecer la concurrencia.	La categoría 3C a través de la subcategoría *colaboración* permite establecer la concurrencia.
MoCA	Grupo per se y social	Establecida por: sincronización, número de comunidades de práctica, escala, distribución física, permanencia planificada y volumen de negocios.	Establecida por las dimensiones de sincronización, distribución física, escala, número de comunidades de práctica, naciente, permanencia planificada y volumen de negocios.

En el entorno de los modelos analizados (véase la Tabla 15) se considera el nombre asignado al espacio compartido y la interfaz de usuario. El primero en la mayoría de los casos es sesión, mientras que en los modelos teóricos y MoCA este nombre no se especifica. El segundo tampoco se especifica en los modelos teóricos y MoCA, y en los restantes su representación es diferente en cada uno.

Finalmente, se puede resumir que los modelos han buscado especificar la interacción de los miembros (usuarios) del grupo, ya sea para trabajo organizado y planificado o para reuniones con un sentido de ocio (redes sociales); sin embargo, dicha especificación en la mayoría de los modelos analizados es informal y sólo uno es semi-formal (AMENITIES), aunque TOUCHE utiliza una ontología esta no es para el modelado de *groupware* sino para homogenizar y evitar ambigüedades en los conceptos aplicados en el dominio CSCW. El rango de estos conceptos es bastante amplio, presentan ambigüedades, por eso algunos esfuerzos como el ya mencionado de TOUCHE. Algunos conceptos son básicos (están en los modelos analizados y deberían ser parte de nuevas propuestas) y otros podrían sustentar y ayudar a estas propuestas a delinear un conjunto de pasos que sirvan de guía en el desarrollo de *groupware.*

Así como también, presentan mecanismos que son parte medular para suministrar la comunicación, coordinación y colaboración, tal como: gestión de la estructura organizacional, notificación y concurrencia. Sin olvidar, la sesión que suministra el espacio de trabajo compartido mostrado a través de una interfaz de grupo, que debe permitir visualizar tanto la interacción del usuario con la aplicación, como entre los usuarios. La interfaz de usuario compartida considera tres vistas: Vista de Información, Vista de Participante y Vista de Contexto.

Tabla 15. **Entorno en *groupware* basado en modelos.**

Nombre	Espacio	Interfaz de Usuario
Teoría de la Coordinación	No especificado	No especificada
Teoría de la Cognición Distribuida	No especificado	No especificada
Análisis de Tarea	No especificado	No especificada
Teoría de la Actividad	No especificado	No especificada
Modelo Conceptual	Sesión	Presenta el modelo interfaz de usuario, que contiene tres vistas: de Objetos de Información, de Participantes y de Contexto.
AMENITIES	Sesión	Determinada por la Vista de información, que obtiene la información que se comparte sobre el escenario colaborativo, que es implícita o explícita.
CIAM	Sesión	Vista de Interacción. Describe las herramientas y los aspectos interactivos de la aplicación.
TOUCHE	Sesión	A través del Modelo de Presentación y del Modelo de Sesión.
MoLICC	Sesión	Es presentada por medio de los elementos del SSI.
Meta-modelo	Sesión	Es presentada a través de la clase sistema.
Meta-modelo Entornos Virtuales	Sesión	A través de la interfaz de usuario gráfica.
Modelo de Clasificación Socio-Técnico	Sesión	La categoría nivel de aplicación permite presentar la interfaz de usuario.
MoCA	Sesión	No especificada.

2.2.4. Conclusiones generales del desarrollo de *groupware*

En los últimos años, ha habido una tendencia centrada en el desarrollo de *groupware* basado en el modelo conceptual. Sin embargo, se han encontrado algunas limitaciones sobre esta forma de desarrollo [Schmidt 90, Weiseth 06, Giraldo 08, Penichet 07b, Verginadis 10, Rodríguez 12, Groudin 12]:

A. Falta de modelos teóricos y computacionales que permitan especificar las actividades grupales.
B. Dificultad para abordar el modelado integral de los aspectos interactivos.
C. Carencia de artefactos de especificación conceptual y formal para modelar tareas colaborativas.
D. Falta de un marco práctico y holístico que conduzca a organizaciones y otras entidades sociales en un esfuerzo por especificar, desarrollar y evaluar sistemas de colaboración que apoyen sus necesidades.

E. Carencia de herramientas de colaboración que contengan una investigación sobre el contexto actual, de tal manera que incluyan los efectos significativos de las implementaciones CSCW que han tenido éxito en la sociedad.
F. Falta de investigaciones para desarrollar o validar estructuras ontológicas de procesos de colaboración para tareas relacionadas con la especificación del trabajo en grupo (parte medular de la interacción del grupo).

Además, en este trabajo de tesis se formula, que el desarrollo de *groupware* debería estar fundamentado en la especificación y modelado formal de estructuras organizacionales acordes a las necesidades y estilos dinámicos de cada grupo.

Por tanto, se propone un enfoque metodológico semántico basado en un modelo arquitectónico, que sirva como una guía para el análisis, especificación y desarrollo de *groupware*, de una manera formal y explícita. El enfoque contendrá los siguientes conceptos, que se han obtenido del estudio realizado en este capítulo y/o serán adaptados a la propuesta metodológica:

- **Estructura Organizacional de Grupo.** La estructura de la tarea requiere establecer una estructura organizacional, que permita definir una división del trabajo (*division of labour*) de acuerdo con las necesidades cambiantes del trabajo en grupo y de la propia aplicación. Este concepto se establecerá como un modelo ontológico en este trabajo, para facilitar la adaptación a diferentes estructuras organizacionales y necesidades cambiantes del grupo.
- **Política.** Corresponde a la Política de Manejo de Sesión, que define la estructura organizacional del grupo (basada en roles, para brindar una división de labor adecuada para *groupware*), que debe ser fácilmente adaptable a las necesidades cambiantes del grupo.
- **Usuario.** También denominado Actor, Participante o Miembro, que puede ser una persona, agente o aplicación, que desempeña uno o más roles para realizar alguna tarea en la aplicación.
- **Rol.** Determina la división de labor de los usuarios en dicha aplicación.
- **Derecho.** También conocido como Obligación o Permiso, que se le concede a un usuario a partir del rol que desempeña.
- **Estatus.** Indica el nivel de privilegios que un rol, tiene en la aplicación.
- **Fase.** Ayuda a estructurar el trabajo en grupo en períodos, en cada una de ellas pueden ser ejecutadas un conjunto de tareas. Esto simplifica el acceso de los usuarios, permitiendo gestionar y controlar el trabajo de grupo.
- **Precedencia de la Fase.** Establece el orden en que se va a ejecutar cada Fase que conforma la Política de Manejo de Sesión.
- **Evento.** Activa una tarea.
- **Tarea.** Indica las actividades que un rol puede desempeñar.
- **Precedencia de la Tarea.** Establece el orden en que se va a ejecutar cada Tarea en una Fase determinada.
- **Actividad**. Instaura las acciones que un Rol puede llevar a cabo. Una Tarea puede estar conformada por una o más actividades.
- **Recurso.** Es el elemento manejado para poder realizar una actividad.
- **Tarea Secuencial.** Se refiere a la Tarea que se realiza después de que se haya concluido otra Tarea.
- **Tarea Paralela.** Se refiere a Tareas que se realizan al mismo tiempo pero no comparten recurso.
- **Tarea Parcialmente Concurrente.** Son Tareas que se ejecutan al mismo tiempo pero no modifican a la vez el recurso.
- **Tarea Totalmente Concurrente.** Tareas que se ejecutan al mismo tiempo y acceden a la vez al mismo recurso.
- **Sesión.** Se refiere al espacio compartido donde se llevará a cabo la interacción.
- **Notificación.** Indica al resto de usuarios, lo que uno de ellos están realizando en el espacio compartido; proporcionando la *Conciencia de Grupo* y la *Memoria de Grupo*, que simplifican la comunicación, colaboración y coordinación.
- **Concurrencia**. Establece el uso mutuamente excluyente de los recursos compartidos, evitando conflictos e inconsistencias en la aplicación.

- **Interfaz de usuario compartida.** Muestra tres tipos de vistas: Vista de Información —muestra los objetos—, Vista de participante —conciencia de grupo— y Vista de Contexto —memoria de grupo— en *groupware*.
- **Vista de Información:** Consiste en una interfaz de usuario donde se muestran los objetos colaborativos y las operaciones sobre éstos. Cada acción colaborativa debe ser notificada a la interfaz de cada miembro del grupo (en algunos casos, solo a parte del grupo o a un solo usuario).
- **Vista de Participantes:** Facilita la comunicación persona-a-persona, suministrando *widgets*, audio, vídeo, etc., los cuales permiten saber: qué personas están participando y qué están haciendo, es decir, proporciona **conciencia de grupo** entre sus integrantes, permitiendo observar las actividades del resto de participantes. En *groupware*, la conciencia de grupo es vital para lograr la comunicación, colaboración y coordinación de los miembros del grupo.
- **Vista de Contexto:** Proporciona todo el material histórico de la sesión que sea útil para realizar de manera efectiva el trabajo en grupo, lo que se conoce como *contexto o memoria de grupo*. Se crea con la finalidad de proveer entendimiento y razonamiento sobre el proceso colaborativo, para hacer un seguimiento exacto del mismo. Una sesión no es un evento aislado, de forma que el contexto que proporciona puede hacer la diferencia entre el éxito frente al fracaso del *groupware*. El contexto implica objetos y condiciones que están inmersos en todas las actividades colaborativas que han sido desarrolladas.

3. Modelos arquitectónicos

En este capítulo, primero se introduce la disciplina de la arquitectura de software, para después describir los principales modelos arquitectónicos existentes para el desarrollo de software. A continuación, se presentan propuestas de arquitecturas de software más relevantes, clasificadas en los diferentes estilos arquitectónicos utilizados en el dominio CSCW. Finalmente, se exponen las principales conclusiones relativas a lo presentado en el capítulo.

3.1. Conceptos básicos

En los años sesenta surge el concepto de ***arquitectura de software*** en los entornos de investigación. Sin embargo, en los noventa fue cuando toma popularidad este término, tras originarse la "crisis del software" y como un área interesante de la naciente disciplina de la Ingeniería del Software, de la cual es parte de estudio, ya que se centra de describir y el procesar los componentes de la estructura de una aplicación, sus interrelaciones, así como los principios y reglas que gobiernan su diseño y evolución a lo largo del tiempo [Garlan 95, Addy 98, Cuesta 00]. Normalmente, se la vincula con la etapa de diseño, aunque su principal objetivo es hacer énfasis en la importancia de la descripción estructural de los sistemas de software [Cuesta 03]. La arquitectura de software tiene como objetivos [Paderewski 03]:

- Comprender y manejar la estructura de las aplicaciones complejas.
- Reutilizar estructuras (o partes de ellas) para resolver problemas parecidos.
- Planificar la evolución de la aplicación, identificando las partes susceptibles de modificación, reduciéndose así los costos relacionados con los cambios.
- Analizar el buen funcionamiento de la aplicación y el grado de cumplimiento de sus requisitos, tanto iniciales como los que surgen durante su funcionamiento.
- Permitir el estudio de una o más propiedades específicas del dominio de la aplicación.

La arquitectura de software representa diferentes aspectos del software, utilizando distintas vistas, que constituyen una representación parcial de una misma arquitectura y todas éstas proporcionan su descripción global. En cualquier arquitectura existen al menos tres vistas [Garlan 95, Kazman 94]:

- **Visión estática.** Describe qué componentes tiene la arquitectura.
- **Visión funcional.** Describe qué hace cada componente.
- **Visión dinámica.** Describe cómo se comportan los componentes a lo largo del tiempo y cómo interactúan entre sí.

La representación de las vistas de una arquitectura ha sido llevada a cabo a través de diagramas de flujo de datos, diagramas UML (*Unified Modeling Language*) [Booch 06], formalismos, patrones y lenguajes de descripción de arquitecturas o ADL (*Architecture Description Language*) [Medvidovic 00], entre muchos otros. Los ADL proporcionan modelos, notaciones y herramientas que permiten describir los componentes y conectores presentes en la estructura de una determinada aplicación [Medvidovic 00].

Otro concepto importante dentro del dominio de la arquitectura es el de ***patrón de software***, ya que es un elemento que ayuda a simplificar la adaptación de un modelo arquitectónico. Se presentan tres tipos de patrones de software [Gamma 95, Buschmann 96, Bayle 98, Ericksson 00, Borchers 01, Cuesta 01, Fowler 02, Martin 05]:

- **Estilo arquitectónico.** Presenta soluciones de organización a nivel del sistema, que serán descritas en la sección 3.2.
- **Patrón de diseño.** Captura la esencia de una solución exitosa a problemas que ocurren comúnmente en el diseño de software. En consecuencia, puede ser visto como un conjunto claro y genérico de instrucciones, permitiendo personalizar el patrón para resolver problemas

específicos. La importancia del enfoque es su potencial para reducir la brecha entre los requisitos de alto nivel y el diseño. Por esta razón, los patrones de diseño pueden ser usados para la obtención de requisitos, determinando los elementos necesarios para el desarrollo de software y cómo estos se aplican al patrón. Además, pueden guiar el proceso de diseño, asistiendo a los diseñadores en el desarrollo de prototipos iniciales. Por estas razones, los patrones de diseño han sido muy utilizados en la Ingeniería de Software. En lo que respecta a la comunidad de la interacción humano-computadora o HCI (*Human-Computer Interaction*), desde hace varios años, los patrones de diseño han sido de interés para los miembros de la misma, aunque el número de personas involucradas ha sido relativamente pequeño. Así, los patrones se han elaborado para una variedad de dominios de aplicación, como Ingeniería de Software, HCI y desarrollo Web. En el área de CSCW, la cantidad de estudios realizados sobre los patrones ha servido para resolver cuestiones de interés para especificar la interacción de los usuarios con dichos sistemas.

- **Idioms.** Describe cómo implementar aspectos particulares de elementos o de las relaciones entre éstos, usando las características de un lenguaje de programación.

3.2. Estilos arquitectónicos

Un estilo arquitectónico es una familia de sistemas en términos de un patrón de organización estructural [Sommerville 15, Garlan 95, Shaw 95] que describe las características básicas de una arquitectura y establece los límites para su evolución [Shaw 96]. Un estilo puede ser definido a partir de [Sommerville 15, Shaw 96, Shaw 95, Garlan 95]:

- Un vocabulario de componentes y conectores de los que se dispone.
- Un conjunto de restricciones sobre la forma en que éstos pueden ser combinados.
- Un modelo semántico que determina cómo se pueden deducir las propiedades globales del sistema a partir de las propiedades de cada uno de sus componentes.

En los últimos años, una variedad de estilos arquitectónicos acordes a las tecnologías de la información y comunicación han surgido. Por ello, en este trabajo se presentarán estilos arquitectónicos actuales y que se utilizan en el dominio CSCW o han incidido en esta disciplina, siendo un factor importante en el desarrollo de aplicaciones colaborativas. De esta forma, se describirán a continuación los siguientes estilos: *centrado en datos* (*repositorios*), *llamada y retorno* (*orientados a objetos, sistemas en capas, Modelo-Vista-Controlador* (MVC) *y basado en componentes*), *sistemas distribuidos* (*cliente-servidor, objetos distribuidos y peer-to-peer —basado en eventos y arquitectura orientada a servicios*).

3.2.1. Estilo centrado en datos

Este estilo se centra en sistemas relacionados con acceso y procesamiento de datos alojados en estructuras de almacenamiento. Los sub-estilos característicos serían los repositorios, bases de datos, arquitecturas basadas en hipertextos y en pizarra. En este trabajo sólo se hará referencia a los ***repositorios*** [Shaw 96], porque cualquier aplicación colaborativa requiere de una estructura de almacenamiento para su buen funcionamiento. Este sub-estilo se denomina *repositorio* porque existe una estructura de datos central, que contiene un conjunto de componentes independientes, operando sobre dicha estructura. Dos grandes subcategorías son generadas por la forma de controlar las interacciones entre el repositorio y sus componentes [Shaw 96]:

1. El *repositorio* es una *base de datos* tradicional, cuando los tipos de transacciones a la estructura de datos central determinan el o los procesos a ejecutar.
2. El *repositorio* es una *pizarra* (*blackboard*), cuando el estado actual de la estructura de datos central es el activador para la selección de los procesos a ejecutar. Este modelo presenta tres partes [Corkill 91]:

- **Fuentes de conocimiento.** Espacios independientes del conocimiento propio de la aplicación que interactúan por medio de la pizarra. La invocación de una fuente de conocimiento se dispara por el estado de la pizarra.
- **Estructura de datos de la pizarra.** Contiene los datos del estado que permiten resolver el problema y que están organizados en una jerarquía dependiente de la aplicación. El proceso para obtener una solución es a través de los cambios realizados por las fuentes de conocimiento en la pizarra.
- **Control.** Coordinado por el estado de la pizarra, ya que las fuentes de conocimiento responden a los cambios de la pizarra que les afectan. Los puntos de control actuales y su implementación están en las fuentes de conocimiento, en la pizarra, en un módulo separado o en una combinación de ellos.

Los sistemas de pizarra se han utilizado tradicionalmente para el procesamiento de señales, en el reconocimiento de patrones y en sistemas que involucran el acceso compartido de datos con agentes débilmente acoplados.

Las ventajas de este estilo son [Shaw 96]:

- El acceso a la estructura de datos compartida mantiene la integridad del sistema, aunque los componentes del sistema sean independientes.
- Mientras la estructura de datos es consultada y modificada, un tratamiento incremental de la información se lleva a cabo.

Sin embargo, este estilo presenta las siguientes desventajas [Shaw 96]:

- Cualquier modificación de la estructura de datos compartida afecta a los componentes que interactúan con ésta.
- Cualquier nuevo componente que se incorpora al sistema debe adaptarse para trabajar con la estructura de datos compartida.

3.2.2. Estilo de llamada y retorno

Este estilo pondera la modificabilidad y la escalabilidad; por tanto, se recomienda para sistemas complejos y con una gran cantidad de usuarios. Los sub-estilos que se describirán en esta sección son: sistemas orientados a objetos, sistemas basados en componentes, sistemas basados en capas y MVC. Esto se debe a que, en general, el desarrollo de software se basa principalmente en diseñar e implementar objetos y en integrar componentes de otras aplicaciones, así como en realizar un desarrollo modular dividido en capas, lo cual es también aplicable al desarrollo de *groupware*. Por otra parte, este tipo de aplicaciones son inherentemente Web, y hoy en día su construcción se fundamenta en el patrón de diseño MVC o variantes del mismo. Además, es importante resaltar que el enfoque metodológico semántico que se presenta en esta tesis está basado en un MVC personalizado para aplicaciones colaborativas, que se originó a partir de una arquitectura por capas.

3.2.2.1. Sistema orientado a objetos

En este estilo, las representaciones de los datos y sus operaciones primitivas están encapsuladas en un objeto [Sommerville 15]. Los objetos interactúan mediante invocaciones explícitas a métodos (u operaciones) que se encuentran en la interfaz de cada componente. Cada objeto es responsable de preservar la integridad de la representación de sus datos (la cual se oculta a otros objetos) [Sommerville 15]. Las ventajas de este estilo son:

- Cambiar su implementación sin afectar a sus clientes, ya que un objeto oculta su representación.
- Reutilizar clases y objetos existentes, ya que éstos pueden ser invocados en otras aplicaciones mediante interfaces adecuadas.

La desventaja principal de este estilo es que los objetos requieren conocer a priori la identidad de los demás objetos con los que se relacionan; por tanto, si se modifica la identidad de un objeto entonces se deben adecuar todos aquellos que lo invocan.

3.2.2.2. Sistemas basados en componentes

Este estilo se sustenta en principios definidos por la ingeniería de software basada en componentes (*Component-Based Software Engineering,* CBSE) [Brown 98]. Un componente de software es una unidad de composición con interfaces especificadas contractualmente y dependencias del contexto explícitas [Szyperski 02]. Por tanto, se puede utilizar un componente propio o desarrollado por otro (de terceros) en una o varias aplicaciones, para ello, es necesario buscar, enlazar y registrar dicho componente. En un estilo de este tipo [Szyperski 02]:

- Los componentes son componentes en el sentido de CBSE y son las unidades de modelado, diseño e implementación de una aplicación.
- Los componentes soportan algún régimen de introspección, de modo que su funcionalidad y propiedades puedan ser descubiertas, enlazadas y utilizadas en tiempo de ejecución.

Las interfaces (que permiten la interacción entre componentes) están separadas de las implementaciones, y las interfaces y sus interacciones son el centro de incumbencia en el diseño arquitectónico.

3.2.2.3. Sistemas basados en capas

Un estilo por capas se caracteriza por tener un sentido de desarrollo "de abajo hacia arriba" (*bottom-up*), de manera que las capas inferiores deberán aportar recursos a las superiores, según sus necesidades particulares. En un sentido estricto, una capa sólo utiliza servicios de la inmediatamente inferior; y en un sentido relajado, de cualquier capa inferior.

Las capas representan un nivel de abstracción, y los conectores que las unen se determinan a través de protocolos que establecen cómo éstas interactuarán. Un ejemplo es el protocolo de comunicación OSI (*Open System Interconnection*), en el cual cada capa proporciona una parte de la comunicación a un determinado nivel de abstracción, prestando un servicio a la capa superior, y comportándose como un cliente para la capa inferior; incluso en algunos casos, las capas más internas están ocultas y sólo son visibles las adyacentes [Sommerville 15].

Este estilo proporciona las siguientes ventajas:

- Permite descomponer un problema complejo en un conjunto de problemas más pequeños y más sencillos de resolver.
- Permite extender el modelo, añadiendo una nueva capa o un nuevo componente en una capa existente.
- Permite la reutilización, cada capa puede ser utilizada en otros sistemas.

Las desventajas del estilo por capas son:

- Es difícil determinar los niveles correctos de abstracción.
- Es difícil en algunos casos estructurar el sistema en capas, ya que existen partes que no pertenecen a ninguna capa o que deben estar en varias.

3.2.2.4. Modelo-Vista-Controlador

Este estilo, denominado MVC fue diseñado en Smalltalk-80 [Goldberg 84], donde inicialmente fue llamado Modelo-Vista-Controlador-Editor [Addy 15]. MVC es un estilo arquitectónico creado para reducir el costo y mejorar la calidad del software en el paradigma orientado a objetos, fomentando la modularidad, al encapsular los detalles de la implementación detrás de interfaces estables, reduciendo el esfuerzo necesario para entender y mantener el software existente. El patrón arquitectónico aporta un principio claro de separación entre la interfaz de usuario y la semántica de la aplicación subyacente [Goldberg 84].

MVC se ha aplicado exitosamente en diversas áreas, un ejemplo de ello es el desarrollo Web, en donde el MVC clásico se ha modificado, obteniéndose algunas variantes, tales como MVP (*Model-View-Presenter*) [Addy 15] y MVVM (*Model-View-ViewModel*) [Smith 13]. De esta manera, se ha convertido en un destacado patrón de diseño que describe soluciones que han sido probadas, dando excelentes resultados.

La estructura básica de MVC (véase la Figura 16) especifica un desarrollo modular de una aplicación mediante tres componentes [Goldberg 84, Addy 15]:

- **Modelo.** Representa el conocimiento del dominio de la aplicación, es decir, formas únicas que caracterizan el estado y semántica de los datos que una aplicación puede requerir. Cuando un modelo cambia (se actualiza o se modifica), típicamente este cambio se notificará y será visualizado en las vistas.
- **Vista.** Es una representación visual del modelo, que actúa como un filtro de presentación, ya que puede resaltar ciertos atributos del modelo y/o suprimir otros.
- **Controlador.** Éste es el vínculo entre el usuario y la aplicación. Contiene la información y los mecanismos necesarios para asociar una acción a cada evento, esto es, se encarga de recibir la entrada del usuario y decidir qué se debe hacer; en consecuencia, se comunica con las vistas, determinando los datos que deben ser afectados en el modelo para que sean visualizados.

Un importante aspecto del paradigma MVC es el conjunto de reglas que gobiernan la manera en que se comunican los tres componentes. El controlador transforma las acciones del usuario en actualizaciones que son enviadas al modelo. Las actualizaciones provocan que el modelo cambie el estado de los datos. Los cambios en el modelo dan como resultado notificaciones que son enviadas a la vista y al controlador. Cuando se notifica que un estado cambia en el modelo, la vista empieza a modificar la interfaz de usuario. Durante esta modificación, la vista requiere al modelo para consultar el nuevo estado del dato. El controlador puede también necesitar requerir datos del modelo cuando transforma las acciones del usuario [Phillips 99].

Una consecuencia clave de las reglas de comunicación de MVC es que las actualizaciones de la interfaz de usuario son simplemente realizadas si lo requieren los cambios en los datos del modelo.

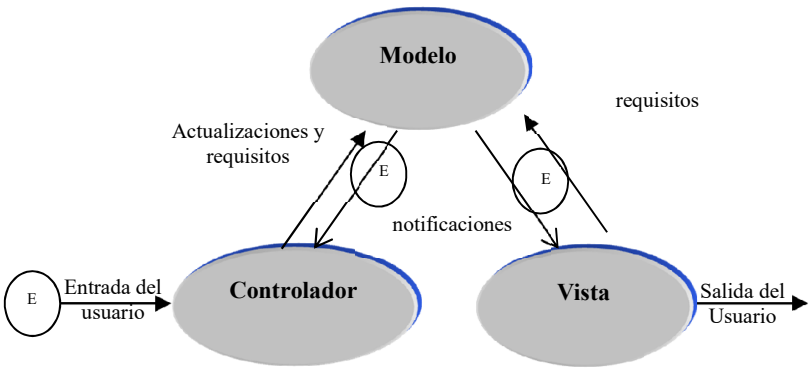


Figura 16. Modelo MVC [Goldberg 84].

3.2.3. Estilo de sistemas distribuidos

En este estilo, las aplicaciones o componentes están distribuidos entre varias computadoras. Por tanto, las aplicaciones o componentes pueden estar, según [Sommerville 15]: implementadas en diferentes lenguajes; ejecutarse en distintos tipos de procesadores; usar diferentes modelos de datos; representar de forma diferente la información y usar distintos protocolos de comunicación. Por consiguiente, se requiere software para gestionar la comunicación y el intercambio de datos entre componentes. Dicho software se denomina *middleware*.

Las principales ventajas de este estilo son [Sommerville 15]:

- Se comparten recursos.
- Es abierto, al estar estos sistemas diseñados con protocolos estándares.
- Concurrencia.
- Escalabilidad.
- Tolerancia a fallos.

Las principales desventajas que presenta este estilo son [Sommerville 15]:

- Complejidad.
- Seguridad.
- Difíciles de gestionar.
- Muy poco predecibles.

Los sub-estilos que se describirán son: Cliente-Servidor, Objetos Distribuidos, y Peer-To-Peer (p2p).

3.2.3.1. Estilo cliente-servidor

El diseño de sistemas cliente-servidor refleja la estructura lógica de la aplicación; por tanto, una aplicación se separa en [Sommerville 15]:

- **Capa de presentación.** Presentación de información al usuario e interacción con el mismo.
- **Capa de procesamiento.** Implementación de la lógica de la aplicación.
- **Capa gestión de datos.** Operaciones sobre bases de datos y archivos.

Al realizar esta separación, es posible distribuir cada capa en computadoras diferentes. De esta forma, se tiene el estilo cliente-servidor en [Sommerville 15]:

- **Dos niveles.** Se tienen varios clientes y un servidor (o varios servidores idénticos) con dos formas diferentes:
  - *Cliente fino.* El procesamiento de la información y la gestión de los datos ocurre en el servidor. El cliente sólo es responsable de ejecutar el software de presentación, dando lugar a un procesamiento pesado del lado del servidor y de la red.
  - *Cliente grueso.* El servidor es responsable sólo de la gestión de los datos. El cliente ejecuta el procesamiento de la aplicación (la lógica de la aplicación) y la presentación. Por tanto, se tiene mejor distribución del procesamiento, pero la administración del sistema es más compleja, de tal manera que, cuando el software de aplicación cambia, hay que reinstalar la aplicación en cada computadora cliente.

  Por otra parte, se debe tener en cuenta que en dos niveles hay que distribuir las tres capas (presentación, lógica y datos) en dos sistemas de computadoras, surgiendo problemas de: escalabilidad y rendimiento, con el cliente fino, o de gestión del sistema, si se usa cliente grueso.
- **Tres niveles.** La presentación, la lógica y los datos se separan como procesos lógicos diferentes distribuidos en distintas máquinas. Las ventajas de este estilo son:
  - Es más escalable que la de dos niveles.
  - Reduce el tráfico en la red, en comparación con el de cliente fino.
  - La capa de procesamiento de la aplicación se puede actualizar fácilmente, ya que está centralizada.

- **Múltiples niveles.** En este estilo se tienen aplicaciones:
  - De gran escala, con cientos o miles de clientes.
  - En donde tanto los datos como el procesamiento son volátiles.
  - Con fuentes de datos múltiples.

3.2.3.2. Estilo Objetos distribuidos

Este estilo ofrece un enfoque general que remueve la distinción entre clientes y servidores, diseñando la arquitectura como una de objetos distribuidos. Sus características principales son [Sommerville 15]:

- Se compone de objetos que proveen servicios a otros objetos y usan servicios de otros objetos.
- No hay distinción entre clientes y servidores.
- Pueden ser distribuidos entre distintas computadoras en una red y se comunican a través de un *middleware*, el cual es conocido como distribuidor de peticiones de objetos (*Object Request Broker*, ORB), por ejemplo, CORBA.
- Es más complejo de diseñar que cliente-servidor.

3.2.3.3. Estilo peer-to-peer

Este estilo consiste en procesos o entidades independientes que se comunican a través de mensajes. Cada entidad puede enviar mensajes a otras entidades, pero no controlarlas directamente [Sommerville 15]. Los mensajes pueden ser enviados a componentes o ser propagados mediante *broadcast*. En este apartado se describirán el estilo basado en eventos y la arquitectura orientada a servicios, como sub-estilos del estilo Peer-To-Peer.

3.2.3.3.1. Estilo basado en eventos

Este estilo, que tiene sus principios en los sistemas basados en actores, satisfacción de restricciones, "demonios" y redes de intercambio de paquetes, es conocido como *invocación implícita, integración reactiva o broadcast selectivo* [Sommerville 15]. La idea fundamental de la invocación implícita es que un componente publica uno o más eventos. Cuando se publica el evento, el sistema invoca automáticamente a todos los procedimientos que habían estado interesados en dicho evento, de tal manera que la publicación de un evento origina la invocación implícita de procedimientos en otros componentes.

Los componentes de la invocación implícita son los módulos, cuyas interfaces proporcionan un conjunto de procedimientos y un conjunto de eventos. Las ventajas de este estilo son [Sommerville 15]:

- Un componente puede requerir un evento, sin importar quién lo haya publicado, de tal manera que no existe comunicación directa.
- Es fácil modificar, añadir, eliminar o sustituir eventos y/o componentes, gracias a la modularidad e independencia de éstos. Además, como pueden existir varios componentes que notifiquen el mismo evento, si uno de ellos se elimina, los componentes interesados en dicho evento pueden obtenerlo de los demás.

Las principales desventajas que presenta este estilo son [Sommerville 15]:

- Se desconocen los componentes que se verán afectados por las publicaciones de los eventos y el orden en que se realizarán los procedimientos involucrados.
- Se desconoce si algún componente responderá a la publicación de un evento realizado por otro componente.
- El rendimiento global y la gestión de recursos del sistema puede llegar a situaciones críticas, cuando los sistemas de eventos dependen de un repositorio compartido para la interacción.

3.2.3.3.2. Arquitectura orientada a servicios

El término *orientado a servicios* existe desde hace mucho tiempo y se ha usado en diferentes contextos para distintos propósitos. Sin embargo, durante su existencia, ha representado un enfoque diferente para separar, principalmente, la lógica de negocios. Esto significa que la lógica requerida para resolver un gran problema puede ser mejor construida, llevada a cabo y mantenida si se divide en un conjunto de pequeñas piezas relacionadas.

SOA (*Service-Oriented Architecture*) define una arquitectura que está relacionada con las de sus predecesores, como son las arquitecturas cliente-servidor y distribuida, de las cuales preserva las características exitosas [Anzures 06a]. Aunque se construye sobre estas características, difiere en que está profundamente influenciado por distintos conceptos, como patrones de diseño y por un conjunto de nuevas tecnologías.

Para poder establecer una SOA, tanto los procesos de negocio como los procesos de tecnología deben diseñarse para que funcionen como una sola unidad, garantizando que cada proceso tenga su independencia del otro, sin que se vea afectado notablemente cuando se presenten cambios en alguno de ellos [Whittle 16].

SOA suministra los componentes que constituyen la arquitectura básica y un conjunto de principios de diseño que se utilizan para determinar y estandarizar estos componentes [Bell 08]. La tecnología de servicios Web ofrece la plataforma de implementación que permitirá colocar estas piezas juntas para construir soluciones de automatización orientada a servicios. Por tanto, SOA representa una arquitectura que promueve la orientación a servicios a través del uso de servicios Web, limitando el rol de la tecnología propietaria a la implementación y presentación de la lógica de la aplicación encapsulada por el servicio [Anzures 06b]. De esta manera, se suministra un marco de trabajo de comunicación común, estandarizado y abierto, que tiene una implicación significativa para disminuir la heterogeneidad y permitir a las organizaciones generar el mejor entorno para aplicaciones específicas. Con ello, se establece y estandariza la habilidad para encapsular la lógica de aplicación, publicándola vía dicho marco de trabajo. Por consiguiente, SOA representa una *arquitectura abierta, extensible y compuesta* que promueve la orientación a servicios y que está constituida de *servicios autónomos, débilmente acoplados, competentes en calidad de servicio, de diversos vendedores, interoperables, descubribles y potencialmente reutilizables, implementados como servicios Web* [Anzures 07a].

SOA considera los siguientes requisitos de seguridad comunes para implementar aplicaciones [Bloomberg 06]:

- *Seguridad*: Propiedad que garantiza que las tareas sean realizadas de una manera adecuada, protegiendo los contenidos de los mensajes y el acceso a servicios individuales [Eastlake 01].
- *Fiabilidad*: Asegura que las tareas sean realizadas de manera íntegra, de tal forma que la repartición de mensajes o de notificación de fallos se garantice.
- *Ejecución:* Se deben llevar a cabo las solicitudes de los servicios Web para asegurar que el encabezado del mensaje SOAP (*Simple Object Access Protocol*) y el procesamiento del contenido XML (*eXtensible Markup Language*) no inhiban la ejecución de una tarea.
- *Integridad:* Capacidad para permitir que las tareas no sean alteradas en las interacciones que participen.

3.2.3.3.2.1. Conceptos básicos de los servicios Web

Cuando se decide implementar soluciones de negocio bajo servicios Web, la palabra clave es *servicio* [Bloomberg 06, Heffner 07]. Un servicio es una unidad de trabajo que se realiza por un proveedor para obtener un resultado final, el cual es requerido, utilizado o ejecutado por un consumidor [Heffner 07]. Tanto el proveedor como el consumidor del servicio son representados por elementos de software que se ejecutan al lado de cada actor que participa en la comunicación [He 03]. Los servicios corresponden a funciones de negocio que, cuando son invocadas, ejecutan una tarea específica, por ejemplo: registrarse y/o autentificarse en un sistema, solicitar datos de cualquier tipo o reuniones, consultar información, etc.

Existen varias definiciones de servicio Web en la literatura. Para el consorcio W3C (*World Wide Web Consortium*) es "una aplicación software identificada por un URI (*Uniform Resource Identifier*), cuyas interfaces públicas y enlaces son identificados, descritos y descubiertos usando XML; que realiza interacciones directas con otras aplicaciones de software aplicando mensajes basados en XML vía protocolos estándares de Internet" [w3c 16]. El consorcio UDDI (*Universal Description, Discovery and Integration*) lo define como "aplicaciones de negocio modulares y autónomas con interfaces abiertas basadas en estándares orientados a Internet" [UDDI 15]. Las partes involucradas en el servicio Web no tienen que preocuparse del sistema operativo, entorno del lenguaje, o modelo de componente usado para crear o tener acceso al mismo, porque están basados en estándares de Internet ubicuos y abiertos, como XML, HTTP, y SMTP" [Arsanjani 09]. Estas definiciones pueden ser complementarias, ya que enfatizan alguna de las propiedades o características de los servicios Web.

3.2.3.3.2.2. Características de los servicios Web

Algunas de las características principales que incorpora la implementación de servicios Web son las siguientes:

- Los servicios encapsulan la lógica interna relacionada con la implementación del mismo, separándolos del resto de servicios o aplicaciones con los que interactúan (*abstracción*) y tienen control sobre ésta lógica (*autonomía*).
- Los servicios Web generalmente se comunican entre ellos intercambiando documentos XML a través de mensajes. El empleo de un modelo de comunicación basado en documentos provee relaciones débilmente acopladas entre servicios. La interfaz del servicio agrega una capa de abstracción al entorno que hace las conexiones flexibles y adaptables.
- Un servicio es *reutilizable*, debido a que la lógica de negocios se divide en entidades modulares, autónomas e independientes.
- Los servicios se adhieren a *contratos* de comunicación, *definidos* por la *descripción del servicio y documentos relacionados*.
- Los servicios Web son diseñados principalmente para ser invocados por otros servicios y/o aplicaciones Web, así que deben describir sus capacidades a otros servicios. Estas descripciones deben incluir sus operaciones, mensajes de entrada y salida, y el modo en que pueden ser invocados.
- Los servicios son distribuidos sobre la Web y accesibles vía protocolos estándares, como HTTP (*HyperText Transfer Protocol*) y SMTP (*Simple Mail Transfer Protocol*), haciéndolos independientes de la plataforma y del lenguaje utilizado, facilitando la integración de entornos heterogéneos.
- Se suministra un interfaz estándar que sirve de enlace entre la Web y la lógica de la aplicación que implementa el servicio, que puede ser invocada desde cualquier tipo de aplicación cliente o servidor.
- Se proporciona un registro, donde el servicio Web es registrado y puede ser localizado, permitiendo a los usuarios encontrar los servicios acordes a sus necesidades.
- La invocación del servicio se hace mediante mecanismos basados en estándares abiertos de la tecnología.
- Un servicio es diseñado para describir cómo puede ser encontrado y evaluado vía mecanismos de descubrimiento.

3.2.3.3.2.3. Arquitectura de los servicios Web

El planteamiento de un servicio Web está fundamentado en las interacciones producidas entre tres elementos principales [Papazoglou 08, Erl 15], como se muestra en la Figura 17:

- *Proveedor del servicio:* Desde la perspectiva de negocio, corresponde al propietario del servicio, mientras que desde la perspectiva de la arquitectura, se relaciona con la plataforma tecnológica que aloja el servicio.
- *Consumidor del servicio:* Desde la perspectiva de negocio, es el sistema de información que requiere satisfacer ciertas funciones. Desde la perspectiva de la arquitectura, es la aplicación o componente que invoca o inicia una interacción con el servicio.
- *Registro del servicio:* Corresponde a un catálogo de servicios donde se buscan las descripciones y donde los proveedores publican las mismas. Los consumidores de servicios se conectan al directorio o registro de servicios, obteniendo información de enlace en la descripción de los mismos.

Figura 17. Elementos básicos de SOA [Erl 04].

Las interacciones comprenden las operaciones de *publicación, búsqueda y enlace* [Erl 15]. En un escenario típico, se lleva a cabo la *implementación del servicio,* esto es, un proveedor de servicios crea y registra un componente o módulo de software que es accesible a través de la red. Para lo cual, se realiza y pública una descripción del servicio en un *registro de servicios* (*directorio*). Finalmente, el consumidor busca (de forma local o remota) el servicio que desea utilizar mediante la descripción del mismo; una vez que el servicio es encontrado se establece un enlace y comunicación con el proveedor para invocarlo o interactuar con la implementación de éste.

Desde una perspectiva tecnológica, los componentes que permiten la implementación de un servicio (véase la Figura 18) se describen a continuación [Alonso 04, Erl 04, Casanave 09, Erl 15]:

- **SOA** [Erl 04]. Es un paradigma arquitectónico que permite especificar las interacciones entre los componentes de un sistema. Un componente ofrece un servicio y otros componentes pueden invocar el servicio localizado en un registro de servicios, conforme a un contrato de servicio.
- **XML (*eXtensible Markup Language*)** [Bray 06]. Es un estándar de sintaxis, con un modelo de datos basado en grafos, para representar e intercambiar datos semi-estructurados. XML se deriva del estándar ISO SGML *(Standard Generalized Markup Language)*. Este lenguaje es conocido como lenguaje de meta-marcado, porque permite la definición de lenguajes específicos de marcado.
- **SOAP *(Simple Object Access Protocol)*** [Kohlhoff 03]. Es un marco para intercambiar mensajes con datos en formato XML entre servicios Web en un entorno distribuido. SOAP puede ser usado con una variedad de protocolos de transporte, como HTTP, SMTP y FTP (*File Transfer Protocol*). Un mensaje SOAP tiene una estructura muy simple, siendo un documento XML (llamado *sobre*) en el que se distinguen dos elementos: El primer elemento, denominado *encabezado*, incluye la seguridad, mientras que el segundo, el *cuerpo*, circunscribe datos intercambiados realmente. SOAP define un marco de trabajo que permite definir un mensaje y la manera para procesarlo, un conjunto de reglas para expresar instancias de tipos de datos

definidos en la aplicación y un convenio para representar llamadas a procedimientos remotos o RPC (*Remote Procedure Call*) y las respuestas a dichas llamadas.

- **WSDL *(Web Services Description Language)*** [Christensen 01]. Es un lenguaje basado en XML que describe las características operacionales de los servicios Web. Las descripciones WSDL están compuestas de definiciones de interfaz e implementación. La interfaz es una definición de servicio abstracta y reutilizable, que puede ser referida por múltiples implementaciones. La implementación describe cómo la interfaz es puesta en práctica por un proveedor de servicio dado.
- **UDDI *(Universal Description, Discovery, and Integration)*** [Clemen 05]. Define una interfaz programática para publicar (*publicación* de API) y descubrir (*búsqueda* de API) servicios Web. El componente principal de UDDI es el *registro*, un repositorio XML donde se anuncian los servicios para que otros servicios o aplicaciones puedan encontrarlos. Conceptualmente, la información proporcionada en un registro UDDI consta de páginas blancas (información de contacto), páginas amarillas (clasificación industrial) y páginas verdes (información técnica sobre servicios).

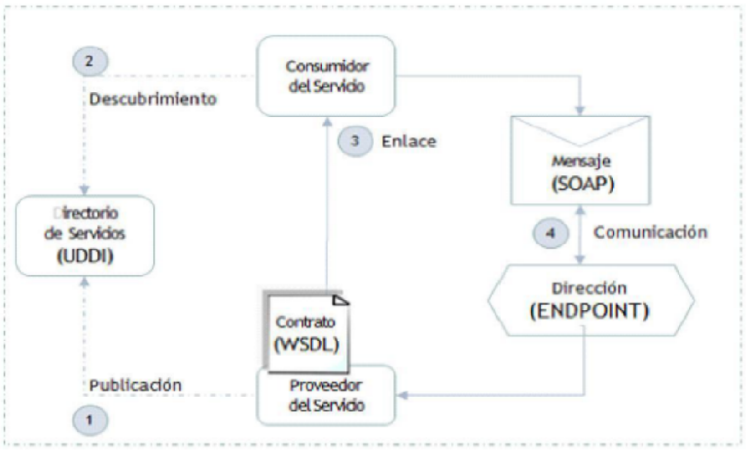


Figura 18. Elementos de implementación de SOA [Erl 16].

3.2.4. Estilo de arquitecturas de Referencia

Las arquitecturas de referencia surgieron como un mecanismo importante en la definición de dominios específicos mediante módulos y sus relaciones [SchmidtD 13]. En general, una arquitectura de referencia se refiere a un tipo especial de arquitectura de software que captura la esencia de una colección de arquitecturas de sistemas en un dominio dado [SchmidtD 13]. También pueden ser consideradas como un repositorio de conocimientos de tal dominio. Entre los beneficios de las arquitecturas de referencia, destaca la posibilidad de reutilización de experiencias a través de la comprensión de un dominio específico.

Las arquitecturas de referencia han sido ampliamente investigadas en el contexto de las arquitecturas de software, proporcionando una estructura para la caracterización de las funcionalidades del sistema de software de un dominio de aplicación dado [Herzog 15], siendo un elemento importante que se reutilizará tanto en el desarrollo de nuevos sistemas como en la evolución de los sistemas existentes. Su uso ha sido explorado en diversos campos, como sistemas de comercio electrónico, sistemas embebidos, computación ubicua y sistemas robóticos, entre otros [Fioravanti 17].

Por otra parte, ProSA (*Process based on Software Architecture for software development*) [Nakagawa 06] es un proceso iterativo que implica diseño, especialización e instanciación arquitectónica, dirigido a entornos de desarrollo de software incrementales y evolutivos. Su objetivo es sistematizar el establecimiento de arquitecturas de referencia.

ProSA es un proceso que se centra en arquitecturas de referencia para el desarrollo de sistemas de software. Esencialmente, este proceso define los pasos requeridos para la construcción y evaluación de

arquitecturas de referencia, así como los pasos hacia la especialización y la instanciación arquitectónica, dirigidas principalmente a la reutilización e incremento de la productividad en el desarrollo de software.

ProSA se compone de tres procesos, como se muestra en la Figura 19 [Nakagawa 06]:

- **ProSA-RA:** ***Reference Architecture*****.** Consta de pasos que permiten el establecimiento, representación y evaluación de arquitecturas de referencia.
- **ProSA-S:** ***Architectural Specialization*****.** Apoya la especialización de la arquitectura de referencia, que representa el refinamiento de una arquitectura de referencia general de un dominio para establecer una arquitectura de referencia específica.
- **ProSA-I:** ***Architectural Instantiation*****.** Establece los pasos necesarios para la instanciación de la arquitectura de referencia, que se refiere a la creación de instancias arquitectónicas a partir de arquitecturas de referencia. La instancia arquitectónica es la arquitectura de un sistema de software particular. Después del establecimiento de la instancia de la arquitectura, se inicia el diseño del sistema y las actividades de implementación.

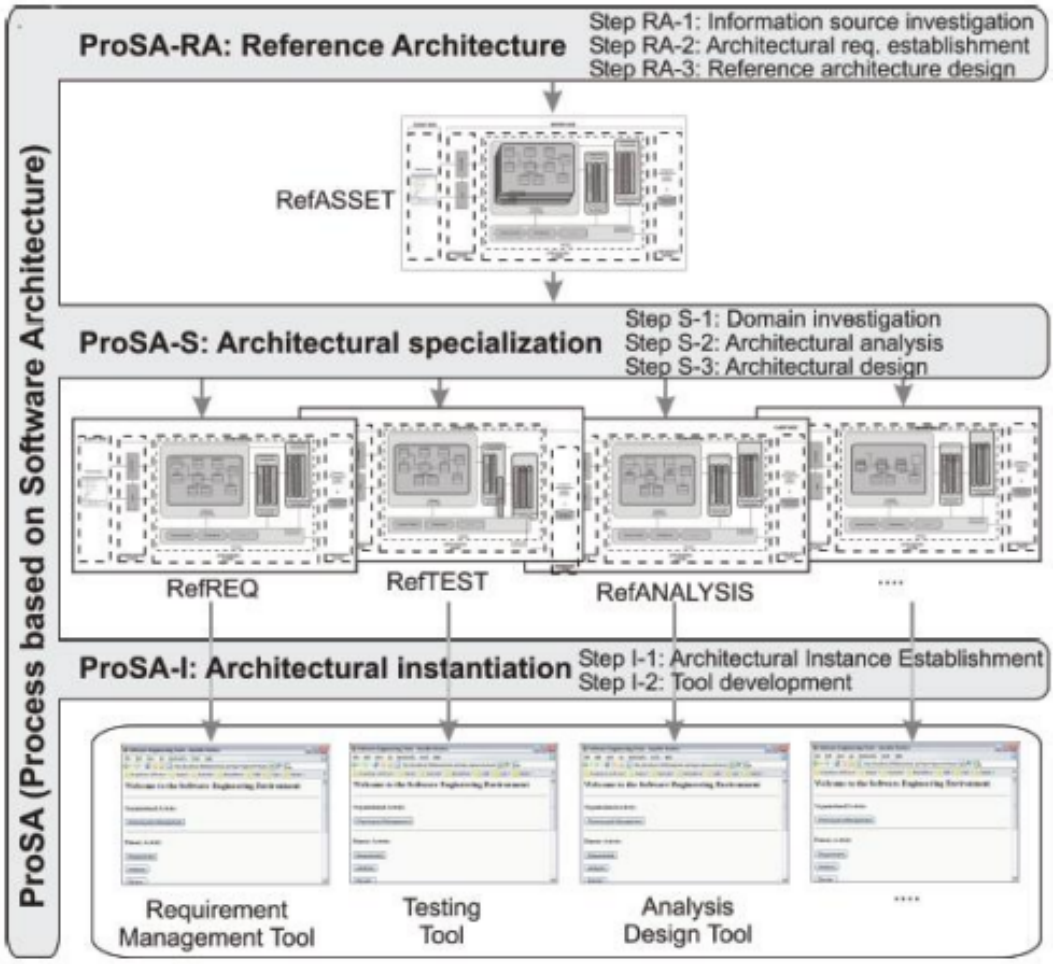


Figura 19. Esquema general PROSA [Nakagawa 06]

3.3. Arquitecturas para el desarrollo de *groupware*

Después de haber explicado en la sección 3.2 algunos de los estilos arquitectónicos que han sido utilizados como base para el desarrollo de *groupware*, en esta se sección se presentan propuestas para este tipo de desarrollo referentes a estilos basados en componentes y capas, así como de MVC.

3.3.1. Estilo de llamada y retorno en *groupware*

En esta sección se presentan propuestas referentes a estilos basados en componentes y capas, así como de MVC.

3.3.1.1. Estilo basado en componentes en *groupware*

Un ejemplo de este estilo es AORTA (*Action-oriented decOupled aRchitecture for coordinaTion and Awareness*) [Orozco 04]. AORTA es un estilo de software para un conjunto de componentes potencialmente integrables en los marcos de trabajo de CSCL/W (*Computer Supported Colaborative Learning/Work*), que sólo permite trabajo síncrono, proporcionando coordinación a nivel de objetos y conciencia de grupo en el espacio de trabajo compartido. AORTA está caracterizada por:

- El uso de ***acciones*** como su abstracción clave. Las acciones son la unidad básica de intercambio de información entre las aplicaciones y AORTA, de modo que se dice que AORTA está orientado a la acción.
- El suministro de un ***conjunto de políticas de coordinación y conciencia***, que permiten soportar escenarios colaborativos complejos y dinámicos. Así, los servicios de coordinación y conciencia del espacio de trabajo compartido proporcionados por AORTA se guían por políticas. La definición de nuevas políticas implica la adaptación y extensión del comportamiento de esta plataforma, sin modificar su arquitectura. Las aplicaciones solicitan la ejecución de acciones a AORTA, entonces AORTA las procesa y decide (coordina) si una acción se debe ejecutar (convirtiéndose así en una interacción indirecta, ya que AORTA es quién finalmente decide que acciones se llevaràn a cabo y no las aplicaciones que las solicitan) o no. AORTA después notifica (conciencia de grupo) el resultado de esta decisión a la totalidad o a parte de las otras aplicaciones.
- La utilización de los ***patrones de diseño de software*** para desacoplar las políticas de coordinación y conciencia del desarrollo de otros aspectos de las aplicaciones CSCL/W.

El uso de políticas orientadas a la acción en AORTA permiten determinar el comportamiento de coordinación y conciencia para *groupware* distribuido y síncrono, con el fin de lograr una mejor adaptación a la dinámica de escenarios colaborativos. Los desarrolladores de aplicaciones CSCL/W sólo deben seleccionar las políticas existentes más adecuadas, o incluso especificar otras nuevas. Las políticas de coordinación y toma de conciencia de AORTA se basan en la información que caracteriza a las interacciones colaborativas y pueden soportar escenarios de colaboración dinámicos, donde una política particular podría no ser válida para toda la duración de la colaboración.

Las políticas de AORTA se activan por acciones que potencialmente pueden convertirse en interacciones colaborativas indirectas, y que se caracterizan por: un *rol* (que desempeña el participante que genera la acción), un *objeto compartido* (sobre el cual se realiza la acción), una *operación* (sobre ese objeto), y un *indicador de tiempo* (que indica cuándo se lleva a cabo la acción). Una política de coordinación deberá evaluar si una acción requerida por un participante se permite o no, de acuerdo con el estado actual de la coordinación (por ejemplo, si este participante tiene el turno o no). La política de conciencia decide si el resultado de una decisión de coordinación debe ser comunicada a los demás participantes. Por lo tanto, las propiedades de una acción determinan qué información debe comunicar una aplicación a AORTA.

AORTA es una arquitectura por capas replicada, diseñada para ofrecer sus servicios a *groupware* que sigue variantes replicadas o híbridas del patrón arquitectónico MVC. La integración entre las aplicaciones y AORTA se logra por medio del *Controlador*, que debe ser total o parcialmente replicado en todas las aplicaciones de los participantes en la colaboración. AORTA toma decisiones sobre tareas de coordinación y conciencia, que se resuelven a nivel local en cada uno de los participantes de la aplicación (utilizando la información previamente recibida de otras aplicaciones), lo que evitará la degradación del rendimiento en la colaboración síncrona. La arquitectura replicada de AORTA se compone de cuatro bloques funcionales, que se encuentran en tres capas de software (véase la Figura 20):

- **Capa de aplicación.** La capa de aplicación constituye el punto de contacto entre las aplicaciones y el bloque funcional de AORTA, denominado AEE (*Action Execution Engine*). que es el encargado de ejecutar las acciones, recibir las solicitudes de la aplicación para la realización de acciones específicas y responder con su ejecución o con una excepción (si la acción no puede ser llevada a cabo). La relación entre las aplicaciones y AEE está definida por medio del patrón de diseño de software *Command*, que encapsula la solicitud para la ejecución de una acción dentro de un objeto. La ejecución de una determinada acción depende del cumplimiento de un conjunto de reglas (contenidas en la política) que se asocian a la solicitud. El uso del patrón *Command* permite el desacoplamiento de la ejecución de la acción requerida de la lógica que determina si puede ser ejecutada o no. La ventaja de esta disociación es triple, dado que: (1) la lógica de negocio puede ser separada de la propia aplicación, lo que permite que se convierta en transparente para el desarrollador de la aplicación; (2) permite que la ejecución de la acción pueda depender de los servicios colaborativos, lo que con lleva a un trabajo en grupo por parte de las aplicaciones involucradas en dicha acción; y (3) facilita el mantenimiento y la reutilización de los servicios colaborativos. Para usar los servicios de AORTA, un desarrollador de aplicaciones CSCL/W sólo tiene que representar acciones a nivel de la aplicación, de acuerdo a las prescripciones de AORTA, y solicitar su ejecución al AEE.

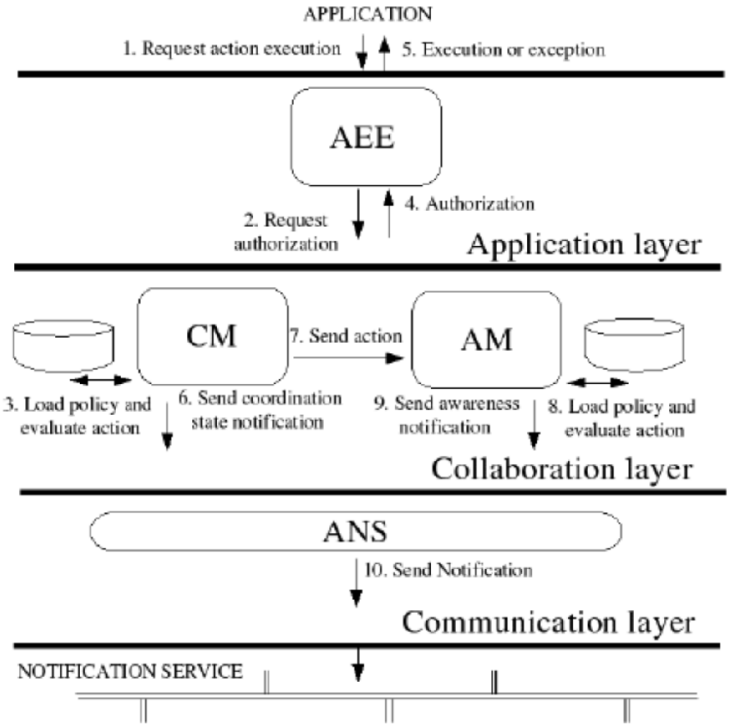


Figura 20. Arquitectura general de AORTA [Orozco 04].

- **Capa de colaboración.** Esta capa ofrece servicios de coordinación y trabajo compartido a través de los bloques funcionales del CM *(Coordination Manager*) y AM (*Awareness Manager*). CM es el encargado de la toma de decisiones relacionadas con la coordinación, evaluando si las solicitudes para la ejecución de acciones pueden llevarse a cabo o no. AM es responsable de la

conciencia de grupo en el espacio de trabajo compartido, por lo que recibe decisiones sobre la ejecución de acciones por parte del CM y resuelve si han de ser notificados o no los demás participantes de las aplicaciones. AEE pide autorización al CM para la ejecución de una acción. CM recibe una acción y evalúa su contenido, esta evaluación se basa en el contenido de la política de coordinación que ha sido previamente cargada desde un repositorio de políticas. La decisión sobre la política que debe cargarse es dictada por una interfaz administrativa. Las políticas de coordinación pueden dictar reglas para gestionar turnos, conflictos, acceso a recursos y a la información relativa a la conciencia del espacio de trabajo compartido para adaptar su estado de manera dinámica. Estas reglas se basan en la información proporcionada por la acción (operación, usuario, objeto, fecha y hora) y en el estado de coordinación actual asociado a esa política. El estado de coordinación está determinado por las decisiones de coordinación previamente hechas. El estado de coordinación es replicado en todos los CM que participan, permitiendo así que el CM local tome las decisiones de coordinación pertinentes en el futuro. AM evalúa el contenido de la acción de acuerdo con una política de conciencia cargada previamente desde un repositorio de políticas. Las políticas de conciencia determinan qué acciones deben ser notificadas y qué otros AM deben recibir las notificaciones. Una vez que la política es cargada, el AM realiza las correspondientes notificaciones por medio del servicio de notificación (ANS) que se explicará a continuación.

- **Capa de comunicación.** Contiene sólo un bloque funcional, denominado Servicio de Notificación de Acciones (*Action Notification Service* - ANS). ANS es un servicio de notificación de acciones, responsable de la propagación de la ocurrencia de una acción a todas las réplicas de AORTA. AM y CM acceden a ANS para solicitar el envío de notificaciones. En el caso de AM, las notificaciones relacionadas con la conciencia se envían a otras aplicaciones interesadas en ellas. En el caso de CM, ANS se utiliza para el intercambio de notificaciones relativas a los cambios en el estado de coordinación. En otras palabras, los CM de las aplicaciones colaborarativas son sincronizados a través de ANS. ANS desacopla AORTA de la utilización de la tecnología MOM (*Message-Oriented Middleware*), que es una capa que permite a diferente componentes hetereogeneos comunicarse, a través de mensajes, a pesar de su diferencias.

3.3.1.2. Estilo basado en capas en *groupware*

En esta sección se presentan tres propuestas clásicas de arquitecturas para *groupware,* como son: Modelo Arch/Slinky, Arquitectura de Dewan y Clover.

3.3.1.2.1. Modelo Arch/Slinky

El modelo de Seeheim [Pfaff 85] plantea la comunicación entre el usuario y la aplicación estructurada en tres niveles: Presentación (parte estática y visible de la interfaz que se comunica con el usuario); Diálogo (parte dinámica que maneja los eventos o mensajes que se producen como consecuencia de las acciones del usuario sobre la interfaz, y establece la comunicación entre el nivel de presentación y el nivel de aplicación) e Interfaz de aplicación (parte de la aplicación que el usuario controla a través de la interfaz y que es visible para éste último). Este modelo fue refinado, dando lugar al modelo Arch [Arch 92].

Arch promueve la descomposición funcional similar a la de Seeheim, pero con mejoras: una clara identificación del nivel de abstracción de cada componente, una definición explícita de la estructura de intercambio de datos entre los componentes, adaptadores entre los principales componentes para mejorar la modificabilidad y la portabilidad, y el meta-modelo *Slinky* para la equilibrar la asignación de funciones en todo el sistema.

Slinky proporciona la capacidad de cambiar funcionalidades entre los componentes, que puede usarse para acondicionar criterios conflictivos o para apoyar la evolución de las herramientas de implementación.

Al incorporar *Slinky* a Arch se genera *Arch/Slinky*, que define una descomposición funcional de un sistema interactivo a través de componentes independientes y de la interfaz de usuario. Este modelo está conformado por cinco componentes (mostrados en la Figura 21):

- **Centro Funcional:** Realiza la manipulación de datos, así como las funciones orientadas al dominio de la aplicación.
- **Adaptador del Centro Funcional.** Este componente, que media entre el Centro Funcional y el Controlador de Diálogo, agrega los datos de dominio específico en las estructuras de alto nivel, realiza una verificación semántica sobre los datos y activa las tareas de diálogo iniciadas en el dominio.
- **Controlador de Diálogo.** Media entre las partes de dominio específico y de presentación de una interfaz de usuario, realizando un mapeo de datos, si es necesario. Asegura la consistencia y controla la secuencia de tareas.
- **Interacción Lógica.** Implementa el comportamiento de la aplicación de acuerdo con las salidas y entradas de la misma, además proporciona un conjunto de *toolkits* para el Controlador de Diálogo.
- **Interacción Física.** Permite la interacción entre el usuario y la computadora. Este componente denota la plataforma subyacente, software y hardware (dispositivos de interacción), correspondiente a los servicios de un *toolkit* de interfaz de usuario.

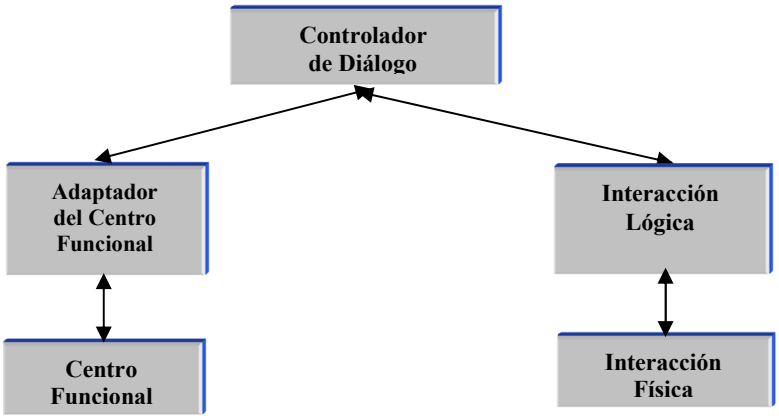


Figura 21. Modelo Arch [Pfaff 92].

3.3.1.2.2. Arquitectura Genérica de Dewan

La arquitectura genérica de Dewan (véase la Figura 22) [Dewan 95a] puede considerarse como una generalización de la taxonomía de Patterson [Patterson 94] y del modelo Arch [Arch 92]. Dewan lleva a cabo esta generalización para incluir varios patrones de comunicación entre las instancias replicadas de una aplicación.

Dewan organiza el *groupware* como una pila de capas (el número de estas es variable) compartidas y replicadas, que se comunican entre sí intercambiando eventos. Esta estructura puede estar *"unzipped"* (distribuida) en el mismo nivel, permitiendo una comunicación directa entre las capas replicadas [Dewan 95b].

La estructura global está compuesta de un árbol y varias ramas. El árbol contiene capas compartidas (de L+1 a N), mientras que una rama está compuesta de capas replicadas (de 0 a L) [Dewan 99]. Una rama contiene capas privadas, que corresponden a cada usuario de la aplicación. Los objetos manejados por las capas de una rama son todos objetos privados (de un usuario), mientras que las capas compartidas y los objetos compartidos son públicos. Las capas se comunican usando eventos de interacción y de colaboración. Los primeros denotan *eventos de entrada*, si se encuentran en la parte superior de la capa, y *eventos de salida*, si están en la parte inferior de la capa, mientras que los eventos enviados entre las capas de ramas diferentes son *eventos de colaboración*.

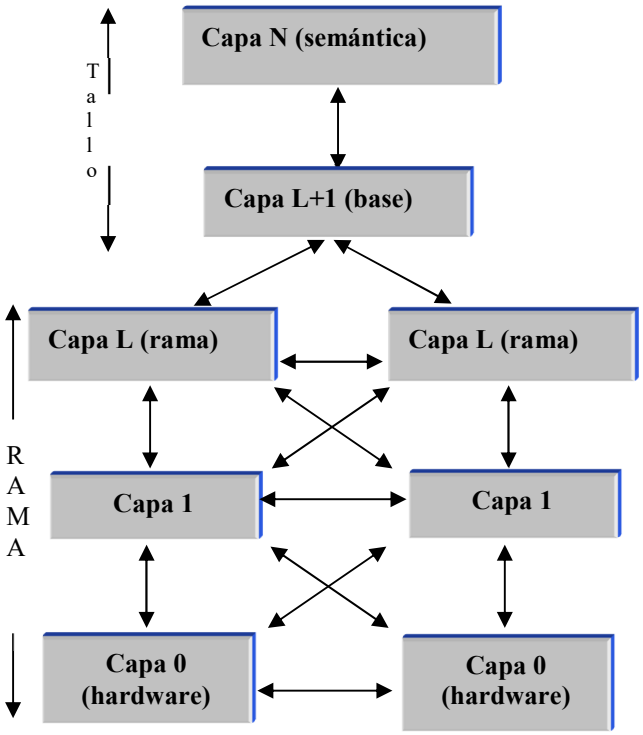


Figura 22. Arquitectura genérica de Dewan [Dewan 95a].

3.3.1.2.3. Clover

Clover [Laurillau 02] es un modelo arquitectónico conceptual para *groupware*. El modelo de diseño para la arquitectura Clover soporta tres servicios:

- **Producción.** Se refiere a los objetos producidos por una actividad de grupo o a los objetos compartidos por usuarios múltiples.
- **Comunicación.** Se refiere a comunicación persona-persona, como correo electrónico, chat, etc.
- **Coordinación.** Se relaciona con dependencias de actividades que incluyen relaciones temporales entre las actividades de los usuarios. También se refiere a las relaciones entre personas y actividades.

El modelo de diseño Clover ayuda a definir especificaciones funcionales y estructuras de datos durante el diseño de *groupware,* pero afecta a su usabilidad, porque la interacción entre múltiples usuarios se vuelve muy compleja.

La arquitectura Clover permite soportar *groupware*. Además, la descomposición de capas en subcomponentes soporta los atributos no funcionales de adaptabilidad, reusabilidad y extensibilidad, que aseguran el mantenimiento de un diseño centrado en el usuario a lo largo del ciclo de vida del producto.

Clover presenta un meta-modelo arquitectónico (véase la Figura 23) derivado de tres modelos arquitectónicos: Arch [Arch 92], arquitectura de Dewan [Dewan 95a, Dewan 95b] y PAC* [Calvary 97].

Este meta-modelo consiste de tallos, ramas, y capas múltiples que representan unidades de software. Cada unidad de modelado e interacción está compuesta de tres elementos: producción, comunicación y coordinación. Estos tres elementos se encapsulan en una entidad lógica, llamada *wrapper*. El *wrapper* oculta a Clover los problemas de otras capas.

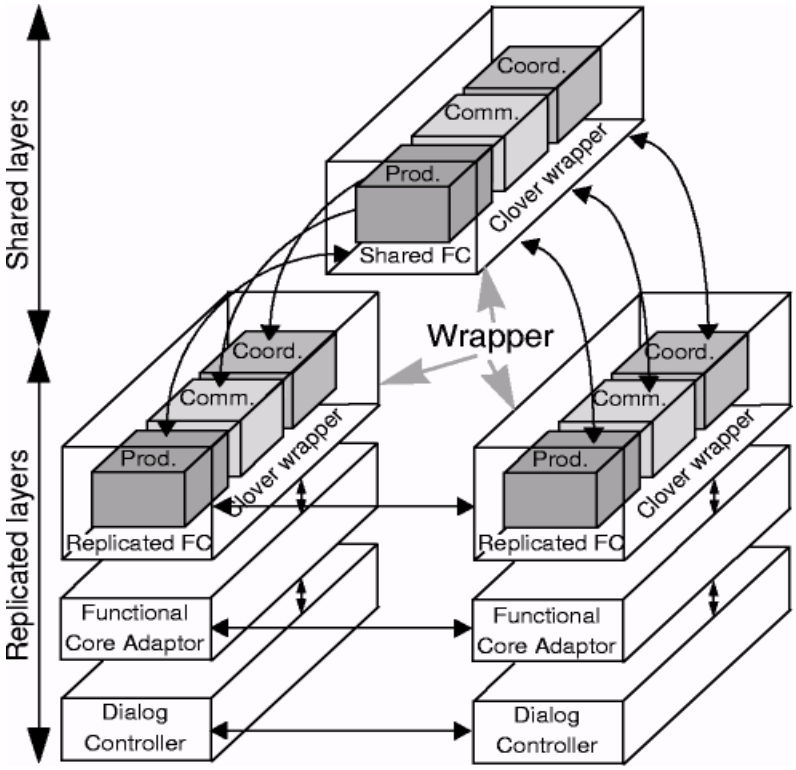


Figura 23. Arquitectura Clover generalizada [Laurillau 00].

3.3.1.3. Estilo MVC en *groupware*

MVC es uno de los patrones de diseño más utilizados en CSCW, principalmente para la gestión de la interacción entre los objetos en diferentes procesos distribuidos, ya que permite separar la presentación de los datos (vistas) del modelo de datos (modelo) y controlar ambos componentes a través del controlador. De esta manera, proporciona independencia a los objetos y procesos, facilitando su manipulación y regulación. MVC se ha utilizado en *toolkits*, tales como: *GroupKit,* COAST, Habanero, JSDT y ANTS, explicados en la sección 2.2.1, así como en las arquitecturas correspondientes a: Taxonomía de Patterson, PAC y Clock, que serán descritas a continuación.

3.3.1.3.1. Taxonomía de Patterson

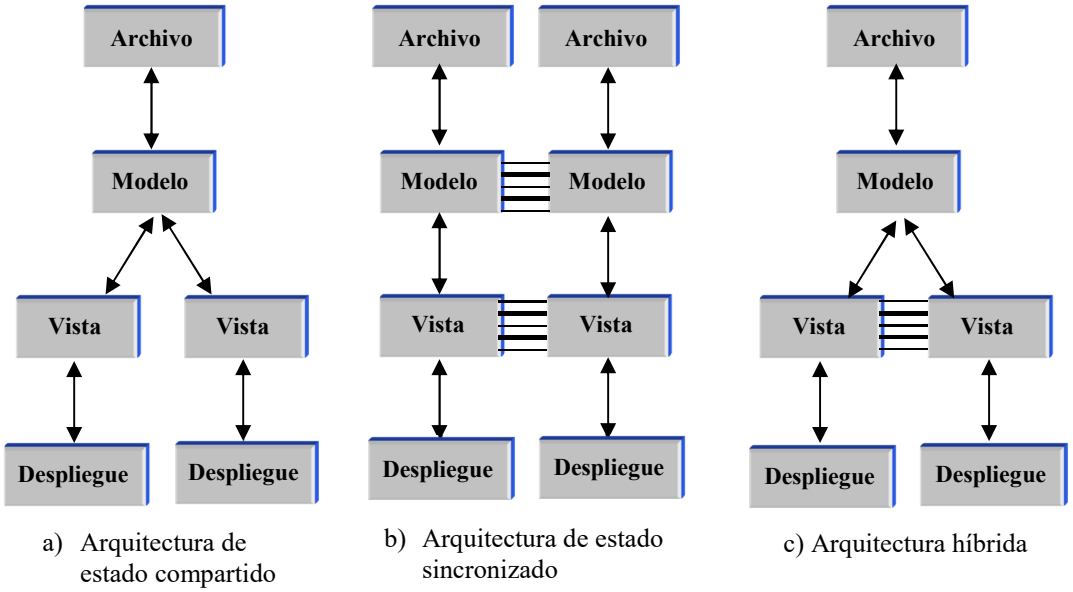
La taxonomía de Patterson fue el primer modelo desarrollado específicamente para *groupware*, y se centró en cómo mantener los objetos compartidos con los que interactúan los usuarios [Patterson 94]. La taxonomía divide el estado de la aplicación en cuatro niveles:

- **Estado de despliegue.** Se lleva a cabo en el hardware que maneja el usuario.
- **Estado de visualización.** Es una representación visual lógica de los datos subyacentes.
- **Estado del modelo.** Los datos subyacentes.
- **Estado de archivo.** La representación persistente del modelo.

Sin embargo, en una aplicación concreta, algunas capas se pueden mezclar con otras u omitir completamente.

Este modelo permite especificar cuáles de estos componentes se localizan en un servidor central y cuáles se ejecutan como clientes en cada máquina de los usuarios. Debido a que los niveles se pueden distribuir de diferente forma (*unzipping)*, se proponen tres tipos de arquitecturas para *groupware*:

- **Basada en el estado compartido real.** Todos los niveles que se encuentran sobre el primero que es compartido, se asumen también como compartidos (como se aprecia en la Figura 24a, a partir de *Modelo*, los niveles *Vista* y *Despliegue* son compartidos). Esto conduce a arquitecturas en las que se observa como si hubieran sido distribuidos los niveles.
- **Basada en estado replicado con sincronización.** Los elementos sincronizados deben contener exactamente la misma información y son replicados para mejorar el desempeño u otras razones pragmáticas (como se puede ver en la Figura 24b, en los niveles *Modelo* y *Vista* hay líneas que indican la replicación y otras —las más gruesas— se refieren a la sincronización).
- **Híbrida.** Incluye aspectos de ambos enfoques. Por ejemplo, una arquitectura híbrida en la que el *Modelo* es compartido y las *Vistas* son sincronizadas (ver Figura 24c). Esto proporciona un mecanismo simple para asegurar la consistencia del estado del modelo y permite a la *Vista*, al ser compartida, modificarse de acuerdo a la interacción con los usuarios.



a) Arquitectura de estado compartido

b) Arquitectura de estado sincronizado

c) Arquitectura híbrida

Figura 24. Ejemplos de la Taxonomía de Paterson [Patterson 94].

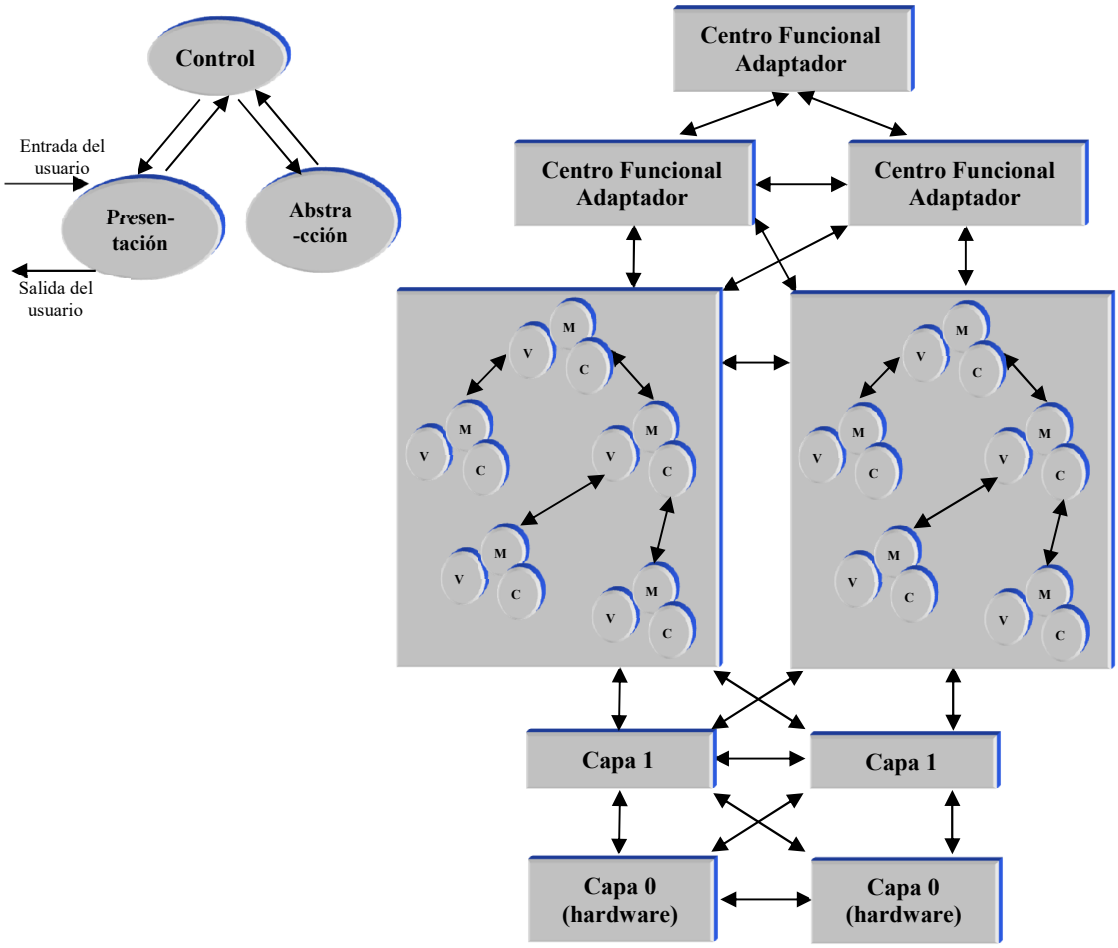
3.3.1.3.2. PAC (*Presentation-Abstraction-Control*)

PAC (*Presentation-Abstraction-Control*) [Coutaz 97] descompone un sistema en una jerarquía de agentes PAC. Cada agente (véase la Figura 25a) incluye tres facetas: *presentación*, que representa la interfaz de usuario; *abstracción,* que mantiene los datos subyacentes; y *control,* que media toda la comunicación entre los agentes de presentación y abstracción, proporcionando código que controla el diálogo entre la presentación y la abstracción, con lo que simplifica la creación y mantenimiento de políticas de diálogo. Cada agente es autónomo, y puede ejecutarse en un proceso o hilo independiente. La faceta de control PAC no sólo adapta las interfaces a sus presentaciones y abstracciones, sino que también inicia y sigue la ruta de las comunicaciones en la jerarquía. Esto conserva las presentaciones y abstracciones simples y localiza todo el código de comunicación relacionado en las facetas de control.

La estructura de una aplicación PAC es una jerarquía de agentes, donde el agente de la raíz representa la aplicación en su conjunto, y se descompone en dos o más subcomponentes que podrían representar diferentes ventanas visibles de la aplicación, las cuales, a su vez, se descomponen hasta que el nivel de objetos interactivos individuales se alcance.

Posteriormente, PAC se usó como la base para PAC-Amodeus [Nigay 91], una arquitectura de aplicación basada en el modelo de referencia de Arch, en el cual el Componente de Diálogo es implementado en el estilo de PAC. Finalmente, PAC* [Calvary 97, Coutaz 97] para *groupware* se extiende de PAC-Amodeus a través de "unzipping" (la distribución de diferente manera) de la estructura Arch en el estilo sugerido por la arquitectura genérica de Dewan (véase la Figura. 25b, en la cual la letra V representa

la Presentación, la letra C, Control y la M, Abstracción), pero también permitiendo múltiples bifurcaciones y uniones. PAC* se ha usado con éxito en el desarrollo de aplicaciones multiusuario complejas con entrada multimodal [Coutaz 97].



a) Un simple agente PAC **b) Arquitectura PAC***

Figura 25. Ejemplos de la Arquitectura PAC [Coutaz 97].

3.3.1.3.3. Arquitectura Clock

Clock es un modelo arquitectónico que proporciona un marco de trabajo a nivel de componentes para soportar el desarrollo de software interactivo con el lenguaje de programación declarativo también denominado Clock [Graham 95]. Esta arquitectura está directamente soportada por un entorno de programación visual, ClockWorks [Graham 96]. Clock está basado en el estilo arquitectónico MVC [Graham 95], como se puede apreciar en la Figura 26.

El modelo Clock se compone de uno o más tipos de datos abstractos *(Abstract Data Types - ADT)* cuyas interfaces incluyen requisitos (acceso de referencia) y actualizaciones (transformaciones). Los controladores constan de funciones declarativas activadas por las entradas del usuario, que pueden ser requeridas por o enviadas al modelo. La vista consiste en funciones declarativas, automáticamente activadas por cambios en los valores requeridos del modelo, el cual calcula la salida del programa.

Una aplicación Clock está constituida por componentes jerárquicamente ordenados, de modo que cada componente incluye típicamente al menos una vista y un controlador. La jerarquía representa un contenedor visual de la vista y, como las vistas que contiene son expresadas (parcialmente) como funciones de otras sub-vistas contenidas, las vistas son restringidas por sub-vistas. Conceptualmente, esto significa que una vista podrá ser automáticamente actualizada si cualquier de sus sub-vistas cambia. En la práctica, Clock incluye optimización para reducir cálculos innecesarios en las vistas.

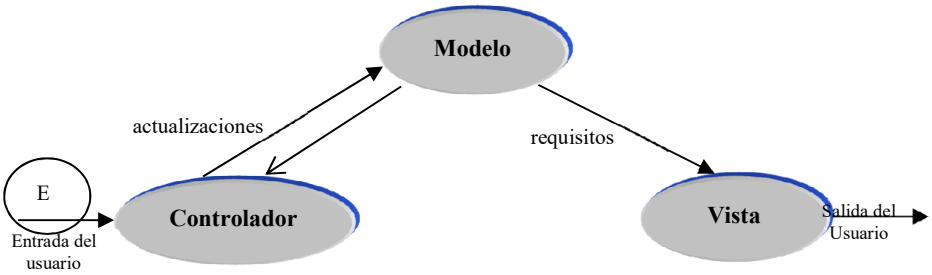


Figura 26. Componente CLOCK [Graham 95]

La jerarquía de componentes extiende el patrón MVC, permitiendo a los controladores y vistas responder a los requisitos o actualizaciones, originando controladores de bajo nivel. Un requisito o actualización es accionado por el modelo local o, si esto no es posible, por el siguiente componente en la jerarquía que proporcione el requisito o actualización en su interfaz. Dentro de un componente, el controlador está debajo del modelo. Los controladores mantienen su transformación de forma jerárquica y permiten la resolución de cualquier desigualdad sintáctica o semántica entre los componentes, ya que responden a los requisitos y actualizaciones del modelo y/o la vista. Esto proporciona un mecanismo simple y natural para compartir datos, dado que si un ADT particular es compartido por varios componentes, normalmente se localiza en el componente más bajo de la jerarquía que es visible a todos ellos. El editor de ClockWorks proporciona un mecanismo simple para mover los ADT de un componente a otro.

Una aplicación *groupware* se programa en Clock proporcionando una función de la vista a algún nivel que permita una sub-vista (subárbol) para ser creado para cada usuario (véase la Figura 27).

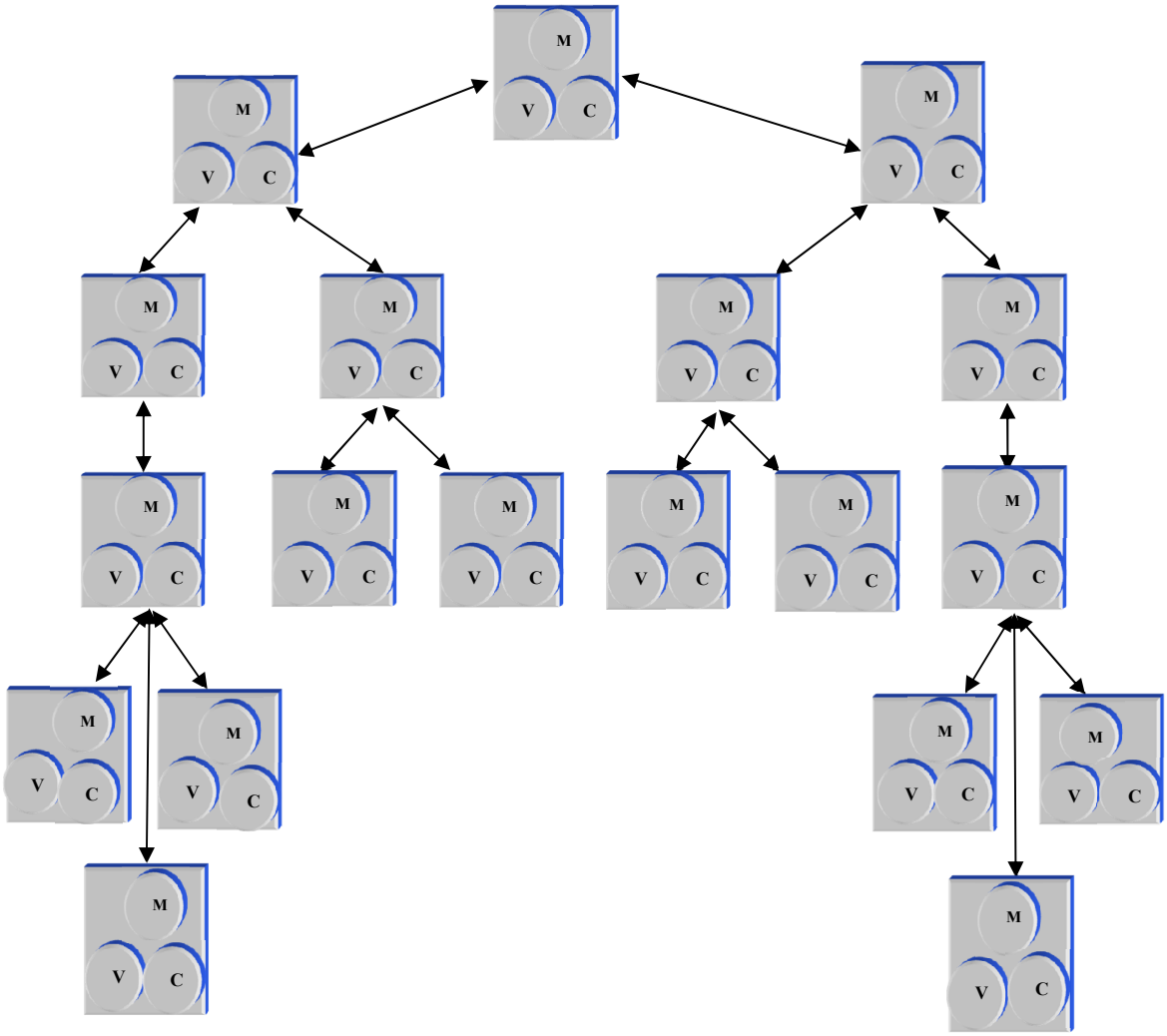


Figura 27. CLOCK Multiusuario [Graham 95].

3.3.2. Estilo p2p

Esta sección se centra en la arquitectura orientada a servicios que sustenta la tecnología de servicios Web, la cual es ideal para implementar *groupware*, ya que se basa en la construcción de nuevas aplicaciones a través de la combinación de servicios disponibles en la red. Estos servicios participan en procesos cooperativos distribuidos, intercambiando mensajes y coordinando sus ejecuciones, facilitando así la interoperabilidad e integración entre aplicaciones. Los sistemas cooperativos resultantes son potencialmente auto-configurables, adaptables y robustos, porque permiten la incorporación dinámica de servicios alternativos y evitan puntos de fallo simples.

En el dominio de CSCW, el desarrollo de aplicaciones basadas en servicios Web se ha centrado en sustentar el trabajo en grupo o en resolver problemas colaborativos. Ejemplos de tales aplicaciones son:

- En [Fox 03] se suministra una infraestructura colaborativa de acceso universal que permite a usuarios de comunidades diferentes colaborar; para ello, sus autores implementaron el subsistema de descomposición de mensajes *NaradaBrokering* [Fox 02a], que transmite los mensajes en entornos heterogéneos, y un adaptador de comunicación de red basado en HHMP (*HendHeld Message Protocol*) [Fox 02b], que soporta dispositivos móviles, en los cuales los procesos se implementan en el servidor de contenido y los datos ya procesados se envían a los usuarios como Servicios Web.
- En [Shaohua 03] y [Shaohua 04] se especifica un modelo de procesos que puede ser dinámicamente reconfigurado y que permite la colaboración de éstos.
- En [Woerner 02a] y [Woerner 02b] se especifica el desarrollo de una estructura de servicio Web con métodos de encriptación y firma sobre el modelo de roles y fases para proporcionar un modelo de producción completo.
- En [Ouyang 04] se proporciona un sistema CAD (*Computer-Aided Design*) colaborativo basado en características[1], que son utilizadas como los elementos centrales del diseño.
- En [Pan 04] se presenta un *framework* para sistemas CAD colaborativos, que permite a los diseñadores conjuntamente realizar una tarea de diseño común, reduciendo las dificultades en cada etapa de diseño.
- En [Greiner 02] se provee un sistema basado en reglas para controlar automáticamente y hacer cumplir las restricciones de procesos cooperativos (especificados por un *workflow*) basados en servicios Web.
- En [Bulcao 04] se muestra un *kernel de contexto* de servicio Web, que permite el intercambio de información de contexto [Dey 01], [Greenberg 01] entre aplicaciones e infraestructuras en entornos heterogéneos.

Para concluir esta sección, se revisan dos plataformas basadas en servicios Web que permiten el desarrollo de *groupware*, como son AGORA [Martínez-Prieto 07a, Martínez-Prieto 07b] y SAGA (*Web Services Architecture for Groupware Applications*) [Fonseca 06]. AGORA es una arquitectura de software por capas destinada a la creación de aplicaciones cooperativas, que está inspirada en SOA [Papazoglou 06] [Heffner 07] [Karakostas 08] [Papazoglou 08], por lo que garantiza que las aplicaciones construidas, desde un punto de vista tecnológico, sean altamente extensibles y adaptables. SAGA es un *framework* que permite el desarrollo de aplicaciones cooperativas a través de la composición de varias funcionalidades básicas, disponibles como servicios Web, los cuales proporcionan un conjunto de operaciones adecuadas para satisfacer las necesidades de toda clase de aplicaciones cooperativas.

[1] Característica es la representación de los aspectos de la forma de un producto relacionados a su diseño y manufactura, que posee atributos y objetivos de ingeniería.

3.3.3. Arquitectura de referencia en *groupware*

En los últimos años, las arquitecturas de referencia se han utilizado principalmente en el ámbito educativo, con el fin de simplificar la creación de plataformas centradas en la interacción estudiante/profesor. Por esta razón, en esta sección se presentan los dos siguientes ejemplos de arquitecturas de referencia: Educar y Ref-mLearning.

3.3.3.1. EDUCAR

EDUCAR es una arquitectura de referencia orientada a aspectos para el desarrollo de entornos de aprendizaje [Barbosa 13]. El principal objetivo de EDUCAR es proporcionar orientación para el diseño arquitectónico de nuevas versiones de entornos de aprendizaje, así como promover una mejor reutilización, evolución y mantenimiento de los existentes. EDUCAR se ha construido siguiendo las actividades prescritas por ProSA-RA en la sección 3.2.4.

EDUCAR se basó en estilos arquitectónicos bien conocidos y consolidados de sistemas interactivos y sistemas Web, como el estilo arquitectónico MVC y la arquitectura de tres niveles. Para representar adecuadamente a EDUCAR, se construyeron las vistas arquitectónicas (vista del módulo, vista de tiempo de ejecución y vista de implementación) usando UML. En ese trabajo, sólo se refieren a la vista del módulo (representada en la Figura 28), que está compuesto de tres capas:

- **Persistencia.** Corresponde al conjunto de datos que debe almacenarse en el entorno de aprendizaje.
- **Presentación.** Se refiere a los módulos del lado del servidor, que son responsables de la interfaz de usuario presentada en el lado del cliente. Esta capa está compuesta por: (i) elemento Controlador, que procesa eventos (típicamente acciones de usuario) e invoca las funcionalidades implementadas por el elemento Modelo; (ii) elemento Vista, que contiene la interfaz de usuario.
- **Aplicación.** Contiene el elemento Modelo, que agrega las funcionalidades relacionadas con el núcleo de entornos de aprendizaje.

Con el fin de promover la separación de los aspectos de los entornos de aprendizaje construidos a partir de EDUCAR, cada módulo en la capa de aplicación está diseñada de manera independiente, permitiendo que cada uno pueda ser diseñado e implementado como un herramienta (o subsistema).

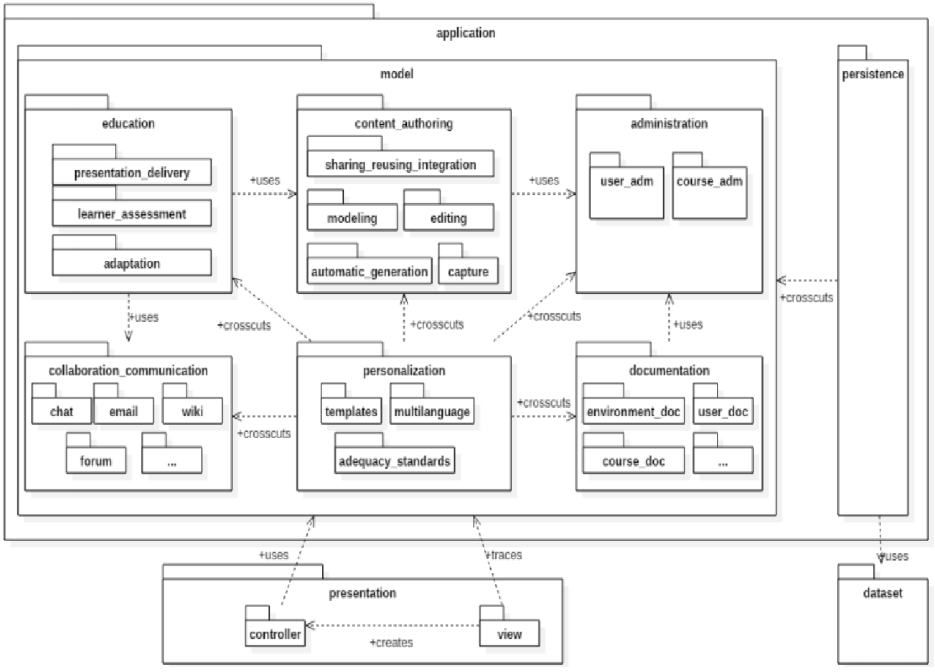


Figura 28. EDUCAR: Vista del módulo [Barbosa 13].

3.3.3.2. Ref-mLearning

Ref-mLearning es una arquitectura de referencia específica, obtenida durante la etapa de ProSA-S (véase la sección 3.2.4) aplicada a EDUCAR. Ref-mLearning se centra en el contexto de aprendizaje móvil e incorpora aspectos de la arquitectura orientada a servicios (SOA) [Erl 15], garantizando pautas para la reutilización y la interoperabilidad de entornos educativos. Ref-mLearning establece cuatro vistas arquitectónicas:

1. Vista general.
2. Vista del módulo.
3. Vista en tiempo de ejecución.
4. Vista de Despliegue.

La Figura 29 presenta la vista general, que suministra una perspectiva global de Ref-mLearning y que contiene cuatro capas:

- **Capa de Persistencia:** Proporciona un manejador de base de datos (DBMS, *Data Base Management System*) y la bases de datos (DB, *Data Base*), para facilitar el procesamiento de los datos que permiten el buen funcionamiento del entorno educativo desarrollado con Ref-mLearning.
- **Capa de Aplicación:** Corresponde al lado del servidor y contiene tanto los requisitos funcionales (los módulos del entorno educativo) como requisitos no funcionales (los servicios que suministran la calidad de este entorno). El entorno educativo es suministrado mediante la *adaptación del módulo de contexto (Adaptation to the Context Module),* que permite ajustar el aprendizaje académico al contexto del estudiante. Tal módulo presenta un *módulo de documentación (Documentation Module)* y de *comunicación (Comunication Module).* El primero contiene toda la documentación necesaria para el entorno educativo. El segundo permite comunicar al módulo de adaptación con el *módulo de enseñanza* (*Teaching Module*). Este último módulo suministra el material requerido para enseñar un tema educativo en específico, para lo cual dicho módulo utiliza un *módulo de autoreo* (Authoring Module). Este es un tutorial automatizado que guía y proporciona material para este aprendizaje, el cual puede ser personalizado, gracias al *módulo de personalización (Personalization Module).* Por otra parte, el usuario puede realizar la *adaptación del módulo de contexto* sólo si accede al mismo a través del *módulo de acceso (Access Module).* Este acceso puede ser configurado mediante el *módulo de administración (Administration Module).*
- **Capa de Presentación:** Como su nombre lo indica, es responsable de presentar, gestionar y controlar los servicios disponibles en la capa de aplicación. Por ello, esta capa contiene un *descriptor de servicios (Services Descriptor),* que se encarga de describir cada uno de los servicios registrados. Una vez que cada servicio es descrito, el *motor de servicios* (*Services Engine*) supervisa, opera y controla (*Controller)* cada servicio utilizado en los diferentes módulos de la capa de aplicación.
- **Capa de Intermediación:** Corresponde al lado del cliente, funciona como un intermediario entre la capa *de aplicación* y la *capa de presentación.* El cliente para interactuar con el entorno educativo de Ref-mLearning utiliza un *agente de servicio* (*Service Agent*), que le permite acceder al *planificador de servicios* (*Scheduler Service*), a través del cual determina cuáles servicios y cómo los utilizará en el entorno educativo proporcionado por Ref-mLearning. Para ello, cada servicio debe ser registrado en el *registro de servicio* (*Service Registry)* y este registro se almacena en el *repositorio de registro* (*Registration Repository*), lo que permitirá su posterior utilización.

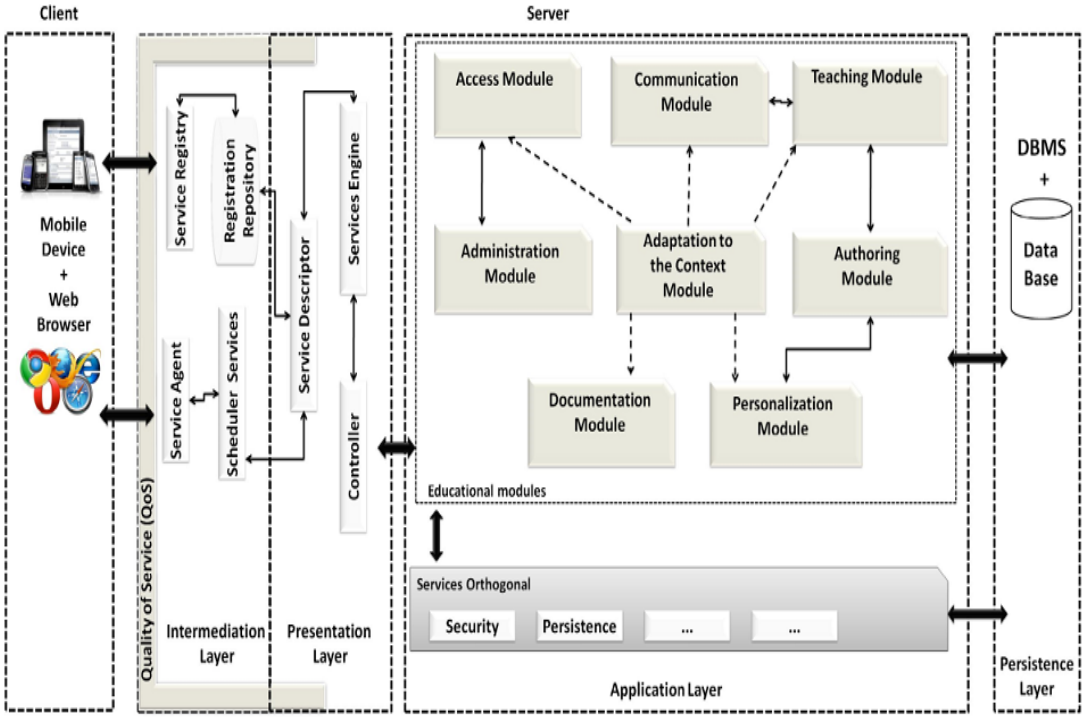


Figura 29. Vista general de Ref-mLearning [Fioravanti 17].

3.4. Conclusiones de arquitecturas de software

Se han revisado una variedad de estilos arquitectónicos, algunos son específicos para el desarrollo de *groupware* y otros sustentan dicho desarrollo de manera implícita o explícita. Lo que se puede resaltar de esta revisión es que:

- Todas las propuestas arquitectónicas son representadas a través de un conjunto de elementos o capas, cuyo número puede variar entre 3 y n.
  - En estas capas, para el caso de estilo no centrados específicamente para desarrollar *groupware,* se presentan las interfaces de usuario, los datos y el control de la interacción de las dos anteriores; mientras en los casos de estilos centrados en desarrollar *groupware* además se agregan elementos claves de *groupware* como la coordinación, colaboración, comunicación, notificación, sincronización y conciencia de grupo.
  - Por tanto, en este trabajo de tesis se propone utilizar el estilo MVC, que se puede ver como una estructura de 3 capas, en las cuales se agrupen elementos que proporcionen la coordinación, colaboración, comunicación, notificación, sincronización, memoria y conciencia de grupo, a partir de la estructura organizacional del grupo. Así como el espacio de trabajo compartido mediante interfaces de usuario compartidas para soportar el trabajo de grupo.
- Las propuestas se basan en uno o combinan varios estilos o arquitecturas como: AORTA (estilo de capas y componentes, así como la utilización de MOM para simplificar la comunicación), modelo Arch/Slinky (basado en Seeheim, Arch y *Slinky*), arquitectura genérica de Dewan (basado en la taxonomía de Patterson, el modelo Arch y estilo por capas), Clover (Arch, arquitectura de Dewan y PAC*), taxonomía de Patterson (basada en el estilo por capas), Clock (extiende la arquitectura PAC*), PAC (utiliza capas y agentes), CLock (fundamentado en MVC), AGORA y SAGA (estilo de servicios Web y capas), Ref-mLearning (estilo de referencia y servicios Web), EDUCAR (estilo de referencia, capas y MVC, así como en un UML).

- La propuesta metodológica utiliza una combinación de estilos (capas, MVC y servicios Web) y un modelo formal para complementar esta metodología. Por lo tanto, la propuesta se fundamenta en un estilo por capas y MVC, utiliza ontologías para modelar y especificar la propuesta; para finalmente sugerir que en la implementación del *groupware* se aplique servicios Web o un marco de trabajo basado en éstos.
- Los elementos que constituyen al MVC de la propuesta son los determinados en el análisis realizado en la sección 2.4. Los cuales se mapean en los conceptos del modelo ontológico que sustenta a la metodología.

En consecuencia, en este trabajo de tesis, se propone una arquitectura MVC por las siguientes razones:

1. Es un estilo muy utilizado y reconocido en el ámbito computacional.
2. Propone un enfoque modular que simplifica la organización, adaptación y extensión de una aplicación.
3. Proporciona un conjunto de directrices para el desarrollo de una aplicación.
4. Se ha usado con gran éxito en el dominio CSCW.
5. Se considera ideal para *groupware*, al poder distribuir los elementos básicos (conforme al estudio realizado en el capítulo 2) de este tipo de aplicaciones en tres componentes: Modelo, Vista y Controlador.

4. Ontologías

En este capítulo, se hace una revisión del concepto y uso de la Ontología, así como sus principales componentes. Se presentan los tipos de ontologías, sus principales aplicaciones, los lenguajes de descripción de ontologías y las principales herramientas utilizadas para su desarrollo. De tal manera, que se identifique el tipo, lenguaje y herramienta de una ontología para modelar la estructura organizacional de grupo.

4.1. El concepto de ontología

El concepto de ontología tiene su origen en el campo de conocimiento de la Filosofía. En la actualidad, es considerada una rama de la Filosofía que tiene por objetivo la explicación de la existencia de una manera sistemática, trata sobre los tipos y estructuras de objetos, propiedades, eventos, procesos y relaciones relativas a cada porción de la realidad [Gómez-Pérez 04].

A principios de los años noventa, el concepto de ontología fue utilizado en el campo de la Inteligencia Artificial, especialmente en la Ingeniería del Conocimiento, el Procesamiento de Lenguaje Natural y la Representación del Conocimiento.

Existen diversas definiciones de ontología en la literatura, que tienen connotaciones diferentes con respecto a su uso del dominio de aplicación. Para Gruber, una ontología es "la especificación explícita de una conceptualización" [Gruber 93]. En esta definición, una *conceptualización* se refiere a un modelo abstracto de cómo la gente piensa sobre algo del mundo real, determinada por objetos, conceptos y otras entidades existentes en un área, así como sus relaciones. *Especificación explícita* significa que los conceptos y las relaciones de un modelo abstracto reciben nombres y definiciones explícitas.

Borst extendió la definición de Gruber para precisar que: “Una ontología es una especificación formal de una conceptualización compartida” [Borst 97]. En esta definición Borst enfatiza que debe haber un acuerdo en la conceptualización que se especifica al añadir el calificativo de “*compartida*” para referirse al tipo de conocimiento contenido en la ontología, es decir, conocimiento consensuado. *“Formal”* se refiere a que la ontología debe ser legible por las computadoras; de otra manera, formal simboliza que en una ontología la semántica es clara y consistentemente definida a través de un formalismo, permitiendo interpretar las representaciones sin ambigüedades y siempre de la misma manera en las computadoras.

Studer, Benjamins y Fensel logran combinar las definiciones de Gruber y Borst para explicar el concepto de ontología: “Una ontología es una especificación explicita y formal de una conceptualización compartida” [Studer 98]. Los términos que comprenden tal definición, son explicados por los autores de la siguiente manera: *conceptualización* se refiere a un modelo abstracto de algún fenómeno en el mundo identificando los conceptos relevantes del mismo, *explícita* significa que el tipo de conceptos utilizados y la restricciones sobre su uso son definidos explícitamente, *formal* se refiere al hecho de que la ontología debe ser legible por la computadora, y *compartida* refleja la noción de que una ontología captura conocimiento consensuado, que no es privado para algún individuo, sino aceptado por un grupo.

El propósito de una ontología es describir formalmente e inequívocamente las nociones relevantes de un dominio. Esta formalización de la terminología constituye la base para un entendimiento compartido y generalmente aceptado de este dominio. Además, las ontologías facilitan el acceso basado en el contenido, la comunicación, y la integración a través de diferentes sistemas de información porque asignan un significado preciso a los datos almacenados.

Gruber propuso utilizar técnicas de Inteligencia Artificial para especificar ontologías [Gruber 95], como *marcos (frames,* que proporcionan más poder expresivo pero menos capacidad de inferencia), *lógicas de primer orden* (que proporcionan poderosas primitivas de modelado) y *lógica descriptiva* (que es más

robusta para el razonamiento). Gruber identificó los siguientes elementos que permiten simbolizar el conocimiento de algún dominio:

- *Clases:* Representan los conceptos (abstractos o concretos) del dominio a formalizar. Las clases se pueden organizan en taxonomías aplicando herencia. Las clases pueden contener individuos (o instancias) con sus correspondientes atributos.
- *Relaciones:* Definen las interrelaciones entre dos o más clases (propiedades del objeto) o un concepto para un tipo de datos (propiedades del mismo).
- *Axiomas:* Los axiomas formales representan expresiones (enunciados lógicos) que son siempre ciertos. Se utilizan para imponer restricciones a los valores de las clases o instancias, representar conocimiento y verificar la consistencia de la ontología.
- *Instancias:* Representan objetos, elementos o individuos de un concepto en una ontología.

En función de estos componentes utilizados para representar el conocimiento del dominio, se distinguen dos tipos de ontologías: ontologías ligeras *(lightweight ontologies)* y ontologías pesadas *(heaviweight ontologies)* [Gómez-Pérez 04]. Las ontologías ligeras contienen todos los componentes excepto los axiomas. Las ontologías pesadas incluyen todos los componentes, y cuantos más axiomas contengan, más pesadas serán.

4.2. El uso de la ontología

El uso de ontología se refiere al propósito de la ontología para humanos o aplicaciones[2]. El propósito de una ontología en el área de ciencias de la computación es describir de un modo formal la noción principal de un dominio [Gómez-Pérez 04].

4.2.1. Clasificación de Uschold y Grüninger

Uschold y Grüninger identificaron tres categorías principales de uso de las ontologías [Uschold 96]:

- **Comunicación** entre la gente y las organizaciones, como medio de unificación entre los diferentes campos de investigación. Los aspectos que facilitan la comunicación entre la gente dentro de una organización son:
  - Modelos normativos: creando la semántica de un sistema y modelo para extenderlo y transformarlo en diferentes contextos.
  - Red de relaciones.
  - Consistencia y falta de ambigüedad.
  - Integración de diferentes perspectivas de usuarios.
- **Interoperabilidad** entre los sistemas, el uso de la ontología como una interlingüa para unificar diferentes lenguajes y herramientas de software. La interoperabilidad puede ser:
  - Interna: bajo el control de una unidad organizacional.
  - Externa: aislada del resto.
  - Integrada entre dominios.
  - Integrada entre herramientas.
- **Ingeniería de sistemas**, el uso de la ontología en el proceso de construcción y mantenimiento de sistemas de software, basados en el conocimiento como:
  - **Reusabilidad:** La ontología cuando es representada en un lenguaje formal puede ser un componente reusable y/o compartido en un sistema de software. (o convertirse por traducción automática)
  - **Confiabilidad:** Una representación formal facilita una verificación de consistencia automática.

[2] **"aplicación"**, un sistema o proceso, (usualmente un programa de computadora) que hace uso o se beneficia de la ontología.

- **Especificación:** La ontología puede asistir el proceso de identificación de requerimientos y definir una especificación para un sistema.

4.2.2. Clasificación de Jasper y Uschold

Jasper y Uschold identificaron cuatro categorías principales de escenarios de aplicación de la ontología, tomados de la industria y entornos académicos [Jasper 99]:

- **Autoreo neutral**: una ontología en un dominio determinado es implementada en un lenguaje único. Diferentes aplicaciones usarán la ontología como fuente de conocimiento, traducida al lenguaje que necesite la aplicación. Los beneficios de esta aproximación son el reuso del conocimiento y el almacenamiento de una gran cantidad de términos.
- O**ntología como especificación**: la ontología en el proceso de identificación de requerimientos y definición de la especificación para un sistema de software.
- **Acceso común a la información:** En algunos casos, se necesita algún tipo de información para una o más personas y aplicaciones con un vocabulario no familiar para todos, o con formatos inaccesibles. La ontología proporciona un vocabulario común y consensuado para múltiples aplicaciones, permitiendo el acceso a una fuente de información homogénea y comprensible. La diferencia con el escenario anterior es que, en este caso, no se traducen a diferentes lenguajes o formatos, sino que la ontología se mantiene como estándar común que deben usar las personas y aplicaciones. Los beneficios que se logran son la Interoperabilidad y un uso más efectivo de las fuentes de conocimiento.
- **Búsqueda basada en ontologías:** Las aplicaciones que realizan búsquedas basadas en conceptos (por ejemplo documentos en algún repositorio) usan una ontología como índice para localizar términos. Se obtiene un acceso más rápido, consensuado y controlado a los recursos de información almacenados.

4.2.3. Clasificación de Mizoguchi

Mizoguchi habla de niveles de uso en áreas educacionales; sugiriendo una lista de cómo usar la ontología [Mizoguchi 95].

- ***Nivel 1***: Usada como un vocabulario común para la comunicación entre agentes distribuidos.
- ***Nivel 2:*** Usada como un esquema conceptual de una base de datos relacional. Se usa la información estructural de conceptos y relaciones entre ellos. La *conceptualización* en una base de datos no es más que el esquema conceptual. La recuperación de datos de una base de datos se hace fácilmente cuando hay un acuerdo sobre su esquema conceptual.
- ***Nivel 3:*** Usada como una información vertebral para un usuario de una cierta base de conocimiento.
- ***Nivel 4:*** Usada para responder en cuestiones de competencia.
- ***Nivel 5:*** Estandarización
    i) Estandarización de terminología (en el mismo nivel del Nivel 1)
    ii) Estandarización de significado de conceptos
    iii) Estandarización de componentes de objetos target (ontología de dominio).
    iv) Estandarización de componentes de tareas (ontología de tarea)
- ***Nivel 6:*** Usada para la transformación de bases de datos considerando las diferencias del significado del esquema conceptual. Esto requiere no sólo la transformación estructural, sino también la transformación semántica.
- ***Nivel 7:*** Usada para predicar conocimiento de una base de conocimiento usando información de representación del dominio.
- ***Nivel 8:*** Usada para reorganizar una base de conocimiento basada en información de representación del dominio.

Así, podemos observar que la variedad de uso de ontología es profunda y amplia. Aquellos niveles más altos que el nivel 3 son novedosos y sugiere el estilo futuro de manipulación de conocimiento por ordenadores, que demuestra la utilidad de la ingeniería de ontología [Gómez-Pérez 04].

4.3. Tipos de ontología

Existe una variedad de tipos de ontología, en este trabajo se mencionan los más importantes, centrándonos en las relacionadas con la representación del conocimiento.

- ***Ontologías de Representación del Conocimiento***. Dado que el conocimiento es una parte de todas las actividades humanas, es necesario almacenarlo, organizarlo y ponerlo a disposición. Esto lleva a requerir una base de conocimiento para representar el dominio del problema, así como ser capaz de razonar y obtener conclusiones a través de un mecanismo de inferencia para los contenidos de tal base [Horrocks 99]. La ontología se presenta como la representación de recursos y conocimientos de una organización a través de un modelo abstracto. Este modelo de representación proporciona un vocabulario común de un dominio y define el significado de los términos y las relaciones entre ellos. Esto proporciona un conjunto de conceptos o clases, relaciones, funciones, axiomas e instancias para describir un dominio de una manera formal y explícita. Además, se puede adaptar cambiando, agregando o eliminando algunos de los elementos que lo constituyen. Por lo tanto, la ontología es ideal para establecer una base de conocimiento que permita especificar y modelar la estructura organizativa del grupo. Por tanto, La ontología es una solución ideal para representar el dominio del conocimiento usando símbolos de lógica descriptiva, que permiten especificarlo de una manera sencilla y legible tanto para humanos como para máquinas. Así como realizar un razonamiento mucho más profundo a través de la computadora. Facilita una base de conocimiento para proporcionar conocimiento semántico, comprensión común, comunicación y conocimiento sobre el dominio de interés y un razonamiento de conocimiento, llevando a cabo un proceso de inferencia para llegar a conclusiones sobre esta base por medio de un razonador. En resumen, las ontologías de representación de, conocimiento describen las primitivas de representación usadas para formalizar conocimiento. Se pretende que sean neutrales con respecto a las entidades del mundo, esto es, que proporcionen un marco representativo sin hacer afirmaciones acerca del mundo. Ejemplos más representativos de estas ontologías es *Frame Ontology* [Gruber 93] y *OKBC ontology* disponibles en *Ontolingua Server*.
- ***Ontologías Generales*** [Horrocks 99] o comunes [Mizoguchi 95]. Son usadas para representar conocimiento de sentido común reusable a través de dominios. Estas ontologías incluyen vocabulario relacionado con eventos, tiempo, espacio, casualidad y funciones. *Mereology Ontolgy* [Borst 97] es un ejemplo de una ontología general.
- ***Ontologías Top-Level o Upper Level***. Describen conceptos muy generales como espacio, tiempo y evento, que son independientes de un problema o dominio particular. Pretenden establecer una estructura básica, bajo la que todos los términos en cualquier ontología existente deberían poder relacionarse. El problema es que hay varias ontologías *top-level* que difieren en el criterio seguido para clasificar los conceptos más generales de la taxonomía, es decir, no existe una ontología única de este tipo.
- ***Ontologías de Dominio*** [Mizoguchi 95, Horrocks 99]. Son reusables en un dominio específico (médicas, ingeniériles, empresariales, químicas, de gestión del conocimiento, de comercio electrónico, etc.). Estas ontologías proporcionan vocabularios acerca de conceptos dentro de un dominio y sus relaciones.
- ***Ontologías de Tareas*** [Mizoguchi 95, Guarino 98]. Describen el vocabulario relacionado con una tarea o actividad genérica mediante la especialización de las ontologías *top-level* (es decir "hipótesis" pertenece a la ontología de la realización de diagnósticos).
- ***Ontologías de Dominio-Tarea.*** Son aquellas ontologías ya clasificadas como ontologías de tareas que se distinguen por ser reusables en un determinado dominio, pero no a través de dominios.
- ***Ontologías Método.*** Definiciones determinadas de los conceptos relevantes y relaciones aplicadas a un proceso de razonamiento específico para una tarea particular.

- ***Ontologías de Aplicación*** [Horrocks 99]. Son dependientes de la aplicación. Contienen todas las definiciones que son necesarias para modelar el conocimiento requerido para una aplicación particular en un dominio dado. Típicamente son una mezcla de conceptos provenientes de ontologías de dominio y de ontologías genéricas. Las ontologías de aplicación no se construyen con el propósito de lograr usabilidad en diferentes dominios.
- ***Ontologías Representacionales*** [Fensel 01]. No están restringidas a un dominio particular, proporcionan entidades representacionales sin establecer lo que debe ser representado (p.e. Frame Ontology define conceptos como frame, slot, slot constraint).
- ***Ontologías de Metadatos*** [Fensel 01]. Como la *Dublín Core*, proporcionan vocabulario para describir el contenido de fuentes de información on-line.

4.4. Lenguajes de ontología

Las ontologías requieren de un lenguaje lógico y formal para ser expresadas. En el área de Inteligencia Artificial se han desarrollado muchos lenguajes con este fin, algunos basados en Lógica de predicados de primer orden, como *KIF* y *Cycl* que proporcionan primitivas de modelado y la posibilidad de rehacer fórmulas que les permita convertirse en términos de otras fórmulas. Otros lenguajes basados en *frames*, con más poder expresivo pero menos capacidad de inferencia como *Ontolingua* y *F-Logic*; así como algunos fundamentados en Lógica Descriptiva que son más robustos en el poder de razonamiento como *Loom, Classic* y *OWL*.

4.4.1. Lenguajes basados en Lógica de primer orden

Se presentan los principales lenguajes desarrollados en proyectos.

- ***KIF*** (***K****nowledge* ***I****nterchange* ***F****ormat*) [Genesereth 92, KIF 00], desarrollado por el grupo *Interlingua* de la Universidad de *Standford*, lenguaje de intercambio de conocimiento entre programas. Usa una semántica declarativa, formalmente definida con suficiente expresividad, posee una estructura que permite traducciones semiautomáticas dentro y fuera de lenguajes típicos de representación. Cuenta con cuatro tipos especiales de expresiones: términos, sentencias, reglas, y definiciones. *Términos* para denotar objetos en el dominio descrito, incluye variables para la cuantificación de objetos, y constantes para denotar objetos. Las *sentencias* expresan hechos sobre el dominio. Las *reglas* expresan pasos de inferencia. Las *definiciones* definen constantes.
- ***CycL*** [Lenat 90, CyC 00], desarrollado en el proyecto CYC (*enCYClopedia,* que ensambla una ontología y una base de datos de conocimiento general para realizar razonamientos del tipo humano), es un lenguaje formal cuya sintaxis se basa en el cálculo de predicados de primer orden y Lisp. Su vocabulario consiste en términos. El conjunto de términos se puede dividir en constantes, términos no atómicos (NATs), variables, y algunos otros tipos de objetos.

4.4.2. Lenguajes basados en *frames*

En el contexto de representación de conocimiento, un *frame* es una estructura que representa un objeto (concepto), compuesto de *slots* (atributos) para los que se especifican *fillers* (valores).

El razonamiento en sistemas basado en *frames* implica tanto conocimiento intensional como extensional contenido en la base de conocimiento del *frame*. En el nivel intensional, el proceso de deducción conduce a la construcción de una taxonomía de *frame*, usando tanto aserciones de inclusión explícitas en la base de conocimiento, como la información estructurada asociada a los *frames* en virtud de sus slots. La taxonomía induce a una modificación de propiedades en el nivel extensional, por valores asignando a *slots* y por propagar los efectos causados por la activación de los procedimientos asociados a los *slots*.

Entre los lenguajes/sistemas basados en *frames*, se mencionan los siguientes:

- ***Ontolingua*** [Farquhar 97] fue desarrollado por *KSL* (*Knowledge Systems Lab*) de la Universidad de *Stanford,* disponible en *Ontolingua Server,* basado en *KIF* (*Knowledge Interchange Format*), *FO (Frame Ontology*) y lógica de primer orden. Como *KIF* es un lenguaje de intercambio de conocimiento, *Ontolingua* emplea FO para describir ontologías utilizando paradigmas de *frames*, y con él empezaron a surgir términos como clase, instancia y subclase. FO no permitía axiomas, pero al surgir a partir de *KIF*, si permite que se incluyan axiomas de *KIF* dentro de sus definiciones.
- ***OCML (Operational Conceptual Modeling Language***) [Motta 98] formó parte del proyecto *VITAL* [Shadbolt 93]. Es un lenguaje para el modelado de conocimiento basado en *frames* con una sintaxis que permite declarar relaciones, funciones, reglas, clases e instancias. A este lenguaje se le añadió un módulo de mecanismos de lógica y una interfaz para poder interactuar con conocimiento. Una de las ventajas que tiene este lenguaje es que es compatible con estándares como Ontolingua [Gruber 93b] y OCML [OCML 95, Motta 98] que usan una sintaxis basada en una mezcla de lógica de primer orden y LISP.
- ***F-Logic*** (*Frame Logic*) [Kifer 95, Angele 04] lenguaje basado en *frames* y lógica de primer orden. Con una estructura parecida a los lenguajes orientados a objetos como la herencia, objetos complejos, tipos polimórficos, etc. Se considera la misma relación (paradigma) que existe entre *F-Logic* y la programación orientada a objetos como la que hay entre el cálculo de predicados y la programación relacional. Con *F-Logic* se han creado bases de datos (deductivas y orientadas a objetos) así como ontologías y se puede combinar con otros programas de lógica para interactuar con la información almacenada en la ontología.
- ***OKBC Protocol*** *(Open Knowledge Base Connectivity Protocol)* [Chaudri 98, OKBC 95], es un protocolo basado en *GFP (Generic Frame Protocol*) es decir, basado en *frames*. Es utilizado como complemento de lenguajes de representación de conocimiento para acceder a bases de conocimiento. *OKBC* contiene un conjunto de primitivas para describir *frames*, clases constantes, atributos e instancias, y para describir operaciones más complejas ya que define procedimientos.

4.4.3. Lenguajes basados en lógica descriptiva

La Lógica Descriptiva, LD, (*Description Logics*) es una familia de formalismos de representación de conocimiento, que representan el conocimiento de un dominio de aplicación en una base de conocimiento en términos de conceptos, individuos (instancias de conceptos) y roles (relaciones binarias). Una base de conocimiento se compone de una terminología TBox y un conjunto de aserciones ABox. TBox es un conjunto de axiomas que definen una jerarquía de conceptos. Además, alguna LD también permite una jerarquía de rol. ABox es una colección de aserciones sobre individuos. Hay dos tipos de aserciones: (1) aserciones de concepto, que especifican que un individuo es una instancia de un concepto, y (2) aserciones de rol, que relacionan una pareja de individuos con un rol [Baader 03].

- ***LOOM*** [MacGregor 91] lenguaje basado en Lógica Descriptiva (familia de lenguajes *KL-ONE* [Brachman 85]*)*, se compone de dos sublenguajes: lenguajes descriptivos que permiten describir

modelos de dominio a través de objetos y relaciones, y lenguajes asertivos que permiten afirmar hechos acerca de individuos.

- ***OIL*** *(Ontology Inference Layer)* [Horroks 00, Fensel 01] fue desarrollado por *European Union IST* (*Information Society Technologies*) como parte del proyecto *OntoKnowledge*. Lenguaje cuyo propósito es alcanzar interoperabilidad semántica entre recursos Web, sintaxis basada en *OKBC, XOL* y *RDF(S)*. Una ontología *OIL* es una estructura de varios componentes (algunos pueden ser estructuras, siendo algunos opcionales y pudiendo repetirse alguno). Al describir una ontología en OIL se deben distinguir tres capas diferentes: (1) el nivel de objeto, donde se describen instancias concretas de la ontología; (2) el primer metanivel, donde se proporcionan las definiciones ontológicas actuales; y (3) el segundo metanivel, consiste de la descripción de características de la ontología como autor, nombre, etc. Aparte de los encabezamientos encapsulados en su contenedor, una ontología OIL consiste en un conjunto de definiciones tales como:
  - **Importar**: Lista de referencias a otros módulos OIL.
  - **Definiciones de Clase**: Una definición de clase (*class-def)* asocia un nombre de clase con una descripción de clase.
  - **Definiciones de *Slots***: Una definición de *slot* (*slot-def)* asocia un nombre de *slot* con una descripción de *slot*. Una descripción de *slot* especifica las limitaciones globales al aplicar al *slot*.
  - **Reglas de Base**: Lista de reglas que se aplican a la ontología.
- ***DAML+OIL*** *(DAML (DARPA Agent Markup Language)*: *DARPA (Defence Advanced Research Proyects Agency)* [Horrocks 01] fue desarrollado por la Unión Europea y los Estados Unidos en el contexto de *DARPA Agent Markup Language (DAML).* Su objetivo es definir un lenguaje que pueda ser usado para definir otros lenguajes que permitan la comunicación entre agentes, p.e. DAML+S (*Services*). Construido sobre RDF(S). DAML+OIL divide el universo en dos dominios: El dominio de tipos de datos que consiste en los valores que pertenecen a los tipos de datos del esquema XML. El dominio de objetos que consiste en objetos que son considerados miembros de clases descritas en DAML+OIL (o RDF). DAML+OIL trata con la creación de clases que describen partes del dominio de objetos. Tales clases se llaman clases de objetos y son elementos de *daml:Class*, una subclase de *rdfs:Class.* DAML+OIL también permite el uso de tipos de datos de esquemas XML para describir parte del dominio de tipos de datos. Estos tipos de datos se usan en DAML+OIL simplemente incluyendo sus URL en una ontología DAML+OIL. Son elementos implícitos de daml:Datatype. Los tipos de datos no son objetos DAML+OIL individuales. A continuación se describen los elementos principales del lenguaje
  - **Elementos de clase**: Un elemento de clase daml: Class contiene parte de la definición de una clase de objetos.
  - **Enumeraciones***:* Una enumeración es un elemento daml:one of, que contiene una lista de objetos que son las instancias. Esto nos permite definir una clase para enumerar exhaustivamente sus elementos. La clase definida por el elemento oneOf contiene exactamente los elementos enumerados.
  - **Propiedades**: Un elemento rdf:Property se refiere al nombre de una propiedad (una URL). Las propiedades que se usan en restricciones de propiedades deberían ser propiedades que relacionan objetos, y que son instancias de ObjectProperty, o propiedades de tipos de datos, que relacionan objetos a valores de tipos de datos, y que son instancias de DatatypeProperty.
  - **Restricciones de propiedad**: Una restricción de propiedad es un tipo especial de expresión de clase. Se define implícitamente una clase anónima, llamada clase de objetos que satisface la restricción.
  - **Instancias**: Instancias de clases y propiedades se escriben en el esquema sintáctico RDF.
- ***OWL*** *(Web Ontology Language)* [Horrocks 03] es un lenguaje del consorcio W3C para definir e instanciar ontologías Web, basado en *XML (Extensible Markup Language)* [Bray 06] y *RDF (Resource Description Framework)* [RDF 10]. OWL está diseñado para usarse en aplicaciones que necesitan procesar el significado o semántica de la información. OWL puede usarse para

representar explícitamente el significado de términos en vocabularios y las relaciones entre tales términos; a través de este lenguaje es posible inferir nuevo conocimiento de una conceptualización usando un software específico llamado *reasoner*. OWL proporciona tres sublenguajes:

- **OWL DL:** Ofrece la máxima expresividad manteniendo la completitud y decidibilidad (los cálculos acabarán en tiempo finito) computacional. OWL DL incluye todos los constructores del lenguaje OWL, pero estos solo pueden usarse bajo ciertas condiciones (p. ej. una clase no puede ser además un individuo o propiedad, una propiedad no puede ser además un individuo o una clase).
- **OWL Full:** Proporciona la máxima expresividad manteniendo la libertad sintáctica de RDF (puede aumentar el significado de cualquier vocabulario definido en RDF), pero sin garantía sobre la complejidad computacional (es improbable que se pueda ofrecer) p. ej., una clase puede ser tratada de forma simultánea como una colección de individuos o como un individuo. Una diferencia clave con OWL Lite es que hay un proceso deductivo dentro del cual una teoría puede ser indefinida.

- **OWL 2.0.** Es una extensión de OWL, con la finalidad de abordar sus limitaciones de expresividad y requisitos de desempeño utilizando lógicas descriptivas con propiedades computacionales deseables [OWL 09]. OWL 2.0 se basa en la lógica descriptiva SROIQ(D) [Horrocks 09], que es una extensión de SHOIN, con un conjunto de constructos expresivos que lo hacen más útil en la práctica. Además de aumentar el poder expresivo de OWL, OWL 2 también define varios perfiles: basado en DL Lite, una lógica descriptiva para la cual los problemas de razonamiento estándar se pueden reducir a la respuesta de la consulta de SQL (*Structured Query Language*); un perfil basado en EL++ (lógica descriptiva ligera diseñada para capturar el poder expresivo que es utilizado por las ontologías a gran escala de aplicaciones prácticas que tienen problemas de clasificación y comprobación de la instancia) para la cual se pueden realizar problemas de razonamiento estándar en tiempo polinomial; y un perfil basado en DLP (*Description Logic Programs*) [Grosof 03], una lógica para la que la consulta de respuesta se implemente utilizando las técnicas basadas en reglas que se han demostrado que escalan bien en la práctica. OWL 2 describe el dominio en términos de individuos, clases (conceptos), propiedades (roles) y tipos y valores de datos (dominios concretos). Y una ontología de OWL 2 consiste en un conjunto de axiomas y hechos que describen el dominio. Finalmente, OWL 2 puede ser visto como un fragmento de la lógica del primer orden (FOL, *First Order Logic*), con una semántica formal basada en la teoría del modelo del primer orden.

Por otra parte, construir ontologías es una tarea laboriosa y propensa a errores, por tanto, se requiere utilizar herramientas que automaticen parte de este trabajo, ocultando características y formalismos de los lenguajes de especificación de ontologías, a través de interfaces gráficas que faciliten la representación y el razonamiento del conocimiento. Ejemplos de este tipo de herramientas son: Ontolingua server [Farquhar 97], WebOnto [Dominguez 98], OilEd [Horrocks 99, Bechhofer 01], OntoSaurus [Swartout 97], Protégé [Musen 00], SWOOP [Swoop 04], TopBraid Compositor [Horridge 11], WebODE [Gómez-Pérez 01], OntoEdit [Sure 02], Neon *Toolkit* [Neon 14].

Esta tesis se centra en Protégé, que es una herramienta de ingeniería de ontología de código abierto y un marco basado en el conocimiento. Protégé se utiliza ampliamente para el desarrollo de ontologías, debido a su escalabilidad y extensibilidad gracias a que dispone de una gran cantidad de *plugins*. Además, de qué simplifica la inferencia de conocimiento a través de razonadores, lenguajes de consulta y reglas.

4.5. Conclusiones

La ontología es un modelo conceptual semántico del dominio de la aplicación, que proporciona una representación estructurada de un dominio de conocimiento en una manera formal, ayudando a remover la ambigüedad, redundancia, detentando errores y permitiendo el razonamiento automatizado. Esto último puede ser llevado a cabo a partir de los axiomas establecidos en cada concepto o las reglas impuestas a la ontología. Esta representación estructurada es ideal para especificar aspectos estáticos como los que se presentan en *groupware* suministrando la semántica requerida. Así como también, para especificar los aspectos dinámicos de este tipo de aplicaciones, el dinamismo puede ser suministrado al nivel de elementos o al nivel estructural; el primero se logra modificando o agregando nuevas instancias, y el segundo al modificar, añadir o eliminar nuevos conceptos o relaciones. Por tanto, en este trabajo de tesis se propone una ontología pesada y clasificada para especificación del manejo de sesión basado en políticas, que se basará en el lenguaje OWL 2.0 y su tipo es tanto de Representación del Conocimiento como de Dominio.

5. Workflow

En este capítulo, se presentan los conceptos fundamentales sobre *workflow*, además de incluir una clasificación y revisión de los principales sistemas de gestión de éste existentes. Finalmente, se describen la ontología *workflow* y se esbozan las conclusiones de lo expuesto en el capítulo.

5.1. Conceptos fundamentales de *workflow*

Un *workflow* se define como la automatización computarizada de un proceso de negocio, en parte o en su totalidad [Fischer 04]. En el contexto del *workflow*, un proceso puede ser considerado como el conjunto de actividades realizadas por diferentes entidades y cuyo orden de ejecución viene determinado por diferentes constructores, que determinan los distintos flujos de ejecución posible (por ejemplo, secuencia, elección, paralelismo, unión y sincronización), así como por la definición de varios objetos que participan en el flujo del proceso, tales como documentos, roles, información intercambiada y las herramientas necesarias para la realización de cada paso. Cada paso se describe generalmente por algunas características como: información de entrada, información de salida y transformaciones hechas por una persona o una máquina desempeñando un rol específico [Marinescu 02].

Por tanto, un *workflow* es la definición estructurada de un proceso utilizado para la gestión automática de las actividades realizadas. Una actividad elemental es una pieza atómica de trabajo [Allen 01]. La formalización de un proceso (esquema de *workflow*) involucra la definición de actividades, así como la especificación de su orden de ejecución (es decir, el flujo de control o enrutamiento) y de los actores responsables. Otras características que deben tomarse en cuenta también, por ejemplo, son el flujo de datos [Allen 01] o las diversas maneras en que los recursos son representados y utilizados en los *workflows* [Fischer 04]. Se pueden distinguir tres tipos de actividades:

- *Actividades colaborativas:* Un conjunto de actores trabajan sobre un mismo recurso para obtener un resultado común. Se considera que el trabajo de cada uno de ellos tiene entidad en sí mismo.
- *Actividades cooperativas:* Un conjunto de actores trabajan sobre su conjunto de actividades, estableciendo los mecanismos de cooperación entre ellos.
- *Actividades de coordinación:* Un conjunto de actores trabajan sobre su conjunto de actividades, estableciendo los mecanismos de coordinación entre ellos.

Por regla general, un *workflow* para un proceso específico refleja la estructura organizacional de ese proceso, el tamaño y la apertura de la comunidad de actores, el nivel de rigor requerido en el control de calidad, la complejidad de la representación, y otros factores. Con respecto a la estructura organizacional en un *workflow*, los actores tienen asignados roles. Cada rol tiene un conjunto de privilegios específicos asociados. Los privilegios determinan qué operaciones se le permiten realizar a un actor determinado en el contexto del *workflow*.

Un *workflow* se centra principalmente en: reflejar, mecanizar y automatizar los métodos de la organización y el proceso de negocios existente en el sistema de información; establecer los mecanismos de control y seguimiento de los procedimientos organizativos; independizar el método y flujo de trabajo de las personas que lo ejecutan; facilitar la movilidad del personal; soportar procesos de reingeniería de negocio; y agilizar el proceso de intercambio de información, así como la toma de decisiones de una organización, empresa o institución [Fischer 04].

Por tanto, el *workflow* ofrece a una empresa la posibilidad de automatizar sus procesos, reducir costos, y mejorar servicios, debido a que existe:

- *un control del proceso de negocio,* que permite monitorear cada paso de este proceso, conociendo su evolución, así como los problemas que se presentan para tomar decisiones críticas

o sólo de modificación de tiempos, con la finalidad de que el proceso siga su curso y se termine en el tiempo establecido;

- *una asignación de las actividades a los roles*, eliminando la tediosa tarea de asignar los trabajos caso por caso; y asegurando que los recursos estarán disponibles cuando se requieran.

5.2. La clasificación de *workflows*

Cada proceso que define un *workflow* formalmente tiene un conjunto de roles diferentes para los actores y un conjunto de actividades diferentes que los actores pueden realizar con estos roles. No sólo los nombres para los roles son diferentes, sino también la manera en la cual los privilegios específicos se distribuyen entre los diferentes roles. La diversidad de procesos que puede haber en una organización lleva a pensar en la existencia de diferentes tipos de *workflows*. En la literatura se pueden encontrar diferentes clasificaciones de *workflows*, en este documento se consideran dos: una con respecto a la diversidad de procesos de negocio y otra referente a la entidad que toman como la pieza clave del proceso, que puede ser actividad o estado.

Para el primer caso, debido a la ***diversidad de procesos de negocio*** que existen dentro de las empresas, se consideran los siguientes tipos:

- *Workflow de Producción*: También conocido como *Workflow de Transacciones*, debido a que la transacción en una base de datos es considerada la clave de todo proceso. Este tipo de *workflow* automatiza procesos de negocio que tienden a ser repetitivos, bien estructurados y con gran manejo de datos. Este tipo es el que tiene el segmento más grande en el mercado.
- *Workflow de Colaboración*. Los *workflow* de este tipo estructuran o semi-estructuran procesos de negocios donde participan personas, con el objetivo de lograr una meta común. Típicamente involucran documentos, y en ellos se sigue la ruta especificada paso a paso, donde se indican las acciones que se toman sobre ellos. Los documentos son la clave, y por tanto, es esencial para la solución de *workflow* mantener la integridad de dichos documentos.
- *Workflow Administrativo*. Es aquél que involucra procesos de administración en una empresa, tales como órdenes de compra, reportes de ventas, etc. Estos *workflows* se emplean cuando existe una gran cantidad de procesos administrativos dentro de la empresa y es necesaria la distribución de soluciones a diferentes actores. Una solución de *Workflow* Administrativo difiere para cada organización, y en ella los cambios son frecuentes; por esto, es muy importante la posibilidad de poder hacer cambios de diseño en este tipo de *workflows*.

Para el segundo caso, es decir, depediendo de lo que se considere como ***pieza clave del proceso***, se distinguen dos tipos de *workflows*:

- En primer lugar, hay *workflows* que se definen como un *conjunto de actividades*, con una actividad invocando la siguiente. Tal *workflow* consiste de un conjunto parcialmente ordenado de actividades. Desde esta perspectiva, un *workflow* es un modelo de una actividad, que está formado por un conjunto de operaciones o pasos.
- El segundo tipo son los *workflows* definidos como *máquinas de estado*, con estados y transiciones. Las transiciones de un estado a otro se producen en respuesta a eventos. Desde este matiz, un *workflow* es un modelo de estados, que está formado por un conjunto de estados y eventos que activan la transición de un estado a otro.

5.3. Sistemas de gestión de *workflows*

Los sistemas de gestión de *workflows* (WfMS, *Workflow Management Systems)* son sistemas que distribuyen el trabajo a diversos actores en una empresa sobre la base de modelos de proceso [Dumas 13], simplificando la implementación de cambios en los procesos de negocio, ya que estos cambios pueden efectuarse fácilmente. Además, un WfMS apoya la idea de trabajar en una manera centrada en el proceso.

Originalmente, los WfMSs se referían principalmente al trabajo de enrutamiento entre actores humanos; después, éstos fueron extendidos con módulos para monitorear y analizar la ejecución de procesos de negocio. A medida que los WfMS se volvieron más sofisticados y mejor integrados con otros sistemas empresariales, se les conoció como Sistemas de Gestión de Procesos de Negocio (BPMS, *Business Process Modelling Systems*) [Dumas 13, Becker 00].

Existen diversos formalismos para la especificación/modelado de BPMS, tales como [Howard 03]:

- BPEL (*Business Process Execution Language*) [Louridas 08, Juric 06] es un lenguaje de ejecución basado en la especificación XML para la descripción formal de los procesos de negocio basados en servicios Web. BPEL utiliza un lenguaje basado en XML y proporciona la capacidad de interconectar con sistemas externos. Los procesos en BPEL importan y exportan información usando exclusivamente interfaces de servicios Web. BPEL no proporciona un protocolo estricto y no hay abstracciones explícitas para las personas, funciones, elementos de trabajo o bandejas de entrada. En cambio, es un modelo centrado en el proceso que se orienta en las interacciones e intercambios de mensajes que se producen en un proceso.
- BPMN (*Business Process Modelling Notación*) [May 03, BPMI 05] es una notación gráfica basada en diagramas de flujo e intuitiva para la definición de los procesos de negocio. Se originó a partir de la BPMI (*Business Process Management Initiative*), en 2005 se fusionó con OMG (*Object Management Group*) [OMG 01] y en 2007 la versión 1.1 se convirtió en un estándar.
  - XPDL (*Process Definition Language*) [XPDL 02] es un lenguaje de marcado creado para garantizar la interoperabilidad entre las diferentes herramientas de gestión de *workflow* para gestionar los procesos del mismo. Fue diseñado para permitir el intercambio de definiciones de procesos, abordando tanto las anotaciones gráficas como las semánticas del *workflow* correspondiente. Nacido como un soporte para la serialización de los constructos BPMN, también incorpora información relativa a la representación gráfica (por ejemplo, la posición de los bloques en el *workflow*). XPDL fue desarrollado por la WfMC (*Workflow Management Coalition*) [WMC 13], un consorcio formado por usuarios, desarrolladores, consultores, analistas y grupos universitarios de investigación, cuyo objetivo es identificar las características comunes de los sistemas de gestión de *workflow* y definir los estándares para la interoperabilidad de dichos sistemas. La WfMC ha desarrollado el modelo de referencia de *workflow*, para definir un sistema de *workflow* e identificar las interfaces más importantes para la interacción entre dichos sistemas.
  - YAWL (*Yet Another Workflow Language*) [Hofstede 10] es un sistema de BPM/*workflow* basado en un lenguaje de modelado conciso y poderoso para manejar transformaciones de datos complejos, permitiendo integración completa con recursos de la organización y externos mediante servicios Web. YAWL es un sucesor de redes de flujo de trabajo (que son una extensión de las redes de Petri para modelar los procesos de negocio, con una sintaxis sencilla y que gira en torno a dos elementos: lugares —corresponde a los eventos BPMN— y transiciones —corresponde a las actividades BPMN), que agrega elementos específicos para capturar el comportamiento de OR, actividades de múltiples instancias, subprocesos y regiones de cancelación. YAWL conserva la simplicidad y la intuitividad de las redes *workflow*, aunque proporciona un lenguaje mucho más expresivo.
  - Wf-XML (*Workflow-eXtend Markup Language*) [Swenson 04] proporciona operaciones de servicio Web para invocar y controlar un proceso que puede necesitar mucho tiempo para completarse, con el fin de facilitar la comunicación entre una herramienta de edición del proceso y una herramienta de ejecución del proceso, que puede ser proporcionada por un proveedor diferente. Wf-XML se centra principalmente en proporcionar una representación de un proceso de negocio.

- jBPM (java *Business Process Management)* [jbpm 10] es un conjunto de herramientas de gestión de procesos de negocio flexible basado en java, que modela los objetivos de un negocio describiendo los pasos y el orden en que se deben ejecutar para lograr un objetivo. Se utiliza un diagrama de flujo, donde un proceso se compone de tareas que están conectadas con flujos de secuencia.

5.4. Ontología *Workflow*

En los últimos años, el interés por el desarrollo de *workflows* ha experimentado un crecimiento considerable en la comunidad científica [Fischer 04], debido a que el flujo de trabajo de cualquier organización se puede construir sobre una ontología. El *workflow* define de qué manera las tareas u operaciones deben ser utilizadas y combinadas. La ontología puede simbolizar el *workflow*, debido a su expresividad. Por tanto, la ontología es una solución ideal para representar los *workflows*, ya que el dinamismo de éstos puede ser representado con la ontología.

La gestión del *workflow* generalmente se logra a través de tres elementos: rutas (representan la secuencia de tareas), reglas (definen condiciones entre roles y rutas), y roles (describen la función de un usuario responsable de una tarea). Una gestión adecuada de un *workflow* requiere [Sloman 94]:

- **Expresividad:** Proporcionar elementos para representar relaciones entre roles y usuarios basados en el modelo organizacional.
- **Verificación del modelo:** Asegurar la especificación exacta del *workflow*, junto con la comprobación de reglas inconsistentes, redundantes e incompletas.
- **Gestión de cambios:** Desarrollar mecanismos y reglas de propagación de cambios para asegurar la precisión del modelo.

La ontología es un modelo adecuado para la gestión de un *workflow*, puesto que suministra: expresividad (al especificar de manera inambigua, explícita y formal el dominio del *workflow* por medio de conceptos, relaciones, instancias y axiomas), verificación (al poderse utilizar razonadores tanto para comprobar que la ontología sea correcta como para inferir información de sus elementos) y manejo de cambios (al permitir modificar, añadir y/o eliminar conceptos, relaciones, y/o instancias de la ontología).

Ejemplos de ontologías *workflow* son los propuestos en los siguientes trabajos: un *workflow* colaborativo para la extracción terminológica y modelado colaborativo de ontologías formales [Gacitua 09] usando dos herramientas: Protégé y OntoLancs; un entorno que permite el desarrollo de ontologías cooperativas y distribuidas basadas en el manejo de dependencias entre módulos de ontologías [Kozaki 07]; una ontología basada en *workflows* para el desarrollo colaborativo de ontologías en Protégé [Sebastian 08]; la combinación de *workflows* con ontologías para diseñar de manera formal protocolos de laboratorios [Maccagnan 10] y una ontología *workflow* para la preservación del material digital de una organización o sistema de archivos [Mikelakis 12]. Todos estos trabajos se centran en construir ontologías *workflow* para representar el trabajo colaborativo en diferentes ámbitos.

5.5. Conclusiones de *workflow*

En esta sección, se puede concluir que la gestión de *workflows*/procesos de negocio es de vital importancia para especificar cada una de las actividades, personas y recursos involucrados para llevarlos a cabo, gestionarlos y supervisarlos en cada paso. También, se puede deducir que una notación formal sería ideal para hacer esta especificación, en especial, las ontologías. Por tanto, en este trabajo de tesis, se presentará una ontología *workflow* para representar y especificar de manera explícita y formal el proceso de negocio relacionado con la estructura organizacional del grupo (que está fundamentada en la política de manejo de sesión). Esta ontología *workflow* simplificará el análisis, diseño y desarrollo de la estructura organizacional, parte central de la interacción desempeñada en *groupware*.

6. Un enfoque metodológico semántico

En este capítulo, se presenta la propuesta principal de este trabajo de tesis, un enfoque metodológico semántico basado en un modelo arquitectónico para el desarrollo de *groupware*. Para ello, se describe el enfoque, los modelos que lo sustentan tanto para soportar como modelar las actividades de comunicación, colaboración y coordinación, así como los pasos que guían el análisis, diseño e implementación de *groupware*.

6.1. Introducción al enfoque metodológico

Los trabajos en el dominio CSCW (revisados en los capítulos 2 y 3), afines con el proceso desarrollo de *groupware*, han hecho avances por disminuir el esfuerzo relacionado con este proceso, tratando de reducir la curva de aprendizaje concerniente al mismo. Sin embargo, no proporcionan una especificación formal para sustentarlo ni para guiar al desarrollador durante las etapas de este proceso.

Como se ha mencionado anteriormente; sobresalen cuatro enfoques [Sosa 06, Molina 09, Rodríguez 12]: Ad hoc, desarrollo de *groupware* basado en *toolkit*s, desarrollo de *groupware* basado en Componentes y desarrollo de *groupware* basado en Modelos. En los últimos años, el desarrollo de *groupware* basado en modelos ha tomado gran relevancia, dando buenos resultados, no obstante, algunos autores han identificado limitaciones en el mismo [Weiseth 06, Penichet 07b, Giraldo 08, Verginadis 10, Grudin 12, Rodríguez 12, Anzures 16b]:

A. Falta de modelos teóricos y computacionales que permitan especificar las actividades grupales.
B. Dificultades para abordar el modelado integral de los aspectos interactivos.
C. Carencia de artefactos de especificación conceptual para modelar tareas colaborativas.
D. Necesidad de nuevas herramientas colaborativas sustentadas en investigaciones que, por una parte, estudien el contexto actual y los efectos significativos en la sociedad de implementaciones exitosas, y por otra, creen o validen estructuras ontológicas de procesos de colaboración para especificar la interacción del grupo.
E. Falta de un modelado formal de estructuras organizacionales acordes a las necesidades y estilos dinámicos del grupo, que soporten la creación de aplicaciones colaborativas.

Por tanto, en este trabajo de tesis se propone un enfoque metodológico semántico basado en un modelo arquitectónico, para soportar el carácter multidisciplinario implícito en el proceso de construcción de *groupware;* así como la naturaleza dinámica del trabajo en grupo, que está determinada por las actividades de comunicación, colaboración y coordinación; proporcionando un conjunto de pautas que asisten al desarrollador en todo este proceso.

De esta forma, el enfoque metodológico semántico propuesto toma en cuenta los aspectos esenciales que permiten el desarrollo de *groupware* (véasen capítulo 2 y 3)*,* su organización y relación en diferentes niveles de abstracción; el modelado explícito y formal inherente al grupo (véase capítulo 4), que dota de la flexibilidad necesaria para ajustarse a los diversos estilos de un grupo o de varios grupos; la especificación explícita y formal de la precedencia de tareas, el acceso y la interacción de los usuarios (véase capítulo 5*)*, así como el uso de servicios Web (véase sección 3.2) para proporcionar reusabilidad y adaptabilidad a la aplicación resultante.

El análisis sobre *groupware* se centró en identificar los componentes esenciales y necesarios que permiten la comunicación, colaboración y coordinación de un grupo de usuarios, que trabajan conjuntamente para alcanzar una meta común. Estos componentes son: Estructura Organizacional del Grupo, Política, Usuario, Rol, Estatus, Derecho/Obligación, Fase, Precedencia de Fase, Evento, Tarea, Precedencia de Tarea, Actividad, Recurso, Tarea Secuencial, Tarea Paralela, Tarea Parcialmente Concurrente, Tarea Totalmente Concurrente, Sesión, Notificación, Concurrencia, Interface de Usuario, Vista de Información, Vista de Participantes y Vista de Contexto.

La revisión correspondiente a las arquitecturas de software se concentró en determinar el modelo arquitectónico más apropiado para sustentar el desarrollo de *groupware*, de modo que los componentes identificados fueran mapeados en este modelo. Primero se decidió utilizar un estilo por capas pero, posteriormente, se optó por un estilo MVC, que se adecuó empleando dichos componentes para soportar la construcción de *groupware;* debido a que MVC simplifica el proceso de desarrollo de una aplicación, estableciendo una organización en tres componentes.

El estudio sobre ontologías se realizó con la finalidad de modelar la semántica y de describir formalmente los componentes mapeados en el patrón arquitectónico MVC. Se resolvió especificar ontológicamente la estructura organizacional del grupo (que determina la división de labor, indicando cómo será realizada y presentada la interacción en las vistas de la aplicación) regida por una política, para proveer el dinamismo requerido por el trabajo en grupo, de tal manera, que sea posible reemplazar o modificar la estructura o la política que lo gobierna cambiando simplemente la política, las instancias, conceptos y/o relaciones. Por ello, también se analizó el desarrollo de sistemas basados en políticas, que permiten separar las reglas que gobiernan el comportamiento de un sistema de la funcionalidad proporcionada por el mismo. De esta forma, sólo se requiere modificar la política para que la estructura organizacional se adapte continuamente a las restricciones y condiciones impuestas interna o externamente.

La investigación referente a *workflows* se encaminó en su capacidad para gestionar un conjunto de actividades considerando su orden de ejecución, así como en los formalismos existentes para su especificación. En este trabajo de tesis se especifica el *workflow* para desarrollar un *groupware,* partiendo de la estructura organizacional de grupo y del MVC, la precedencia de tareas, el control de acceso y la interacción de los usuarios. Dicha especificación se hará a través de ontologías, generando ontologías *workflow*, para proporcionar la expresividad y semántica requerida por el dinamismo inherente del trabajo en grupo, así como también para facilitar el seguimiento del desarrollador en cada etapa relacionada con la construcción de *groupware*.

Se realizó una revisión sobre la tecnología de servicios Web porque teníamos la intención de usar el servicio como la entidad central del modelo arquitectónico, sin embargo, después de dicha revisión, entendimos que el punto central del enfoque metodológico lo constituye el conjunto de modelos y el procedimiento empleado en el desarrollo de *groupware.* Por tanto, se determinó usar los servicios en la etapa de implementación, debido a las ventajas asociadas respecto a la modularidad, reusabilidad y adaptabilidad de una aplicación.

Nuestra propuesta de Enfoque Metodológico pretende ser un soporte conceptual que regule la manera en que se aplican procedimientos y técnicas para desarrollar *groupware*. El término Enfoque se define como: “dirigir la atención o el interés hacia un asunto o problema desde unos supuestos previos, para tratar de resolverlo acertadamente” (definición tomada de la Real Academia Española). Mientras que Metodológico se corresponde con la definición “perteneciente o relativo a la metodología” (de acuerdo a la Real Academia Española). Metodología se define como “"un conjunto completo e integrado de técnicas o métodos que crean una teoría general de sistemas, especificando cómo debería llevarse a cabo un trabajo intensivo” [IEEE 90]. Un método es un “proceso o procedimiento utilizado en la ingeniería de un producto o la realización de un servicio” [IEEE 90]. Una técnica es “un procedimiento técnico y administrativo utilizado para lograr un objetivo determinado” [IEEE, 1990].

La propuesta está conformada de modelos—indicando un conjunto de procesos que se utilizan para establecer la estructura conceptual, que fundamenta el proceso de desarrollo de *groupware*— y el procedimiento —que determina los pasos que se deben ejecutar para lograr tal construcción.

En las siguientes secciones, se presentan los modelos conceptuales que fundamentan la propuesta del Enfoque Metodológico Semántico basado en un modelo arquitectónico para el desarrollo de Groupware. En primer lugar se explica el modelo arquitectónico, en segundo lugar el modelo ontológico de la estructura organizacional de grupo, en tercero el modelo de acceso, en cuarto el modelo de precedencia de tareas y finalmente, el modelo de interacción.

.

6.2. El estilo arquitectónico MVC

Todos los trabajos revisados (en el capítulo 2, 3, 4 y 5) han llevado a obtener diversas conclusiones. Una de estas es que el desarrollo de *groupware* requiere establecer un modelo conceptual, que determine los componentes fundamentales, sus relaciones y evolución en el entorno. Otra conclusión, es que el patrón arquitectónico MVC es el modelo conceptual para fundamentar este desarrollo.

Todos los trabajos revisados (en el capítulo 2, 3, 4 y 5) han llevado a obtener diversas conclusiones. Una de estas es que el desarrollo de *groupware* requiere establecer un modelo conceptual, que determine los componentes fundamentales, sus relaciones y evolución en el entorno. Otra conclusión, es que el patrón arquitectónico MVC es el modelo conceptual para fundamentar este desarrollo debido a que:

- Permite separar el desarrollo de trabajo en grupo en tres componentes (Modelo, Vista y Controlador), donde cada uno desempeña su propio funcionamiento por separado. Esta separación de la aplicación facilita la reutilización del Modelo para varias interfaces de usuario (Vistas); permitiendo organizar el desarrollo de una aplicación de una manera estructurada.
- Es ideal para la creación de *groupware*, ya que al estar organizado en tres componentes se simplifica el desarrollo, detección y corrección de errores; reduciendo el tiempo, el esfuerzo y el costo de este proceso de desarrollo.
- Surgió para el análisis y diseño orientado a objetos, lo que lo hace viable y aplicable para el análisis de requisitos y diseño de cualquier paradigma de software.
- Permite la reutilización de los componentes del patrón, ya que estos se han diseñado de forma independiente (es decir, las responsabilidades de cada una de ellos están separados perfectamente).
- Provee una plantilla que sirve como guía para analizar, diseñar e implementar un proyecto de software; estableciendo un conjunto de recomendaciones para facilitar este proceso de desarrollo.

Lo más importante del MVC propuesto en este trabajo de tesis —que asiste al desarrollo de *groupware*— son los elementos que constituyen cada una de los tres componentes; porque éstos caracterizan tal desarrollo, considerando, principalmente, la interacción generada por el trabajo en grupo y determinada por la comunicación, coordinación y colaboración entre los usuarios del mismo. Estos elementos se obtuvieron del análisis realizado, principalmente en el segundo y tercer capítulo, pero también se consideran aspectos del cuarto y quinto capítulo. Después de realizar la identificación de los elementos, estos se mapearon a cada componente de la arquitectura: Modelo, Vista y Controlador (véase la Figura 30).

6.2.1. El componente modelo

El Modelo correspondiente al estilo arquitectónico MVC contiene los datos necesarios para el desarrollo adecuado de *groupware*. Por lo tanto, en este trabajo de tesis, el Modelo especificará los elementos de la estructura organizacional del grupo con el fin de coordinar y adecuar el trabajo en grupo a sus necesidades cambiantes. La estructura organizacional del grupo se fundamenta en la ontología para modelar la política de gestión de la sesión [Anzures 07b, Anzures 08a]. Para su desarrollo se consideraron tres elementos principales: la estructura organizacional, la gestión de sistemas basada en política y los aspectos básicos de los modelos ontológicos.

6.2.1.1. La estructura organizacional del grupo

El aspecto transcendental de las estructuras organizacionales es la separación de responsabilidades y poderes [Aalst 04], a través de los roles que los usuarios de una organización pueden tener, considerando que estos roles pueden ir cambiando a lo largo del trabajo en grupo o de su permanencia en la organización. Aalst considera tres mecanismos de coordinación o estilos organizacionales:

- Organización jerárquica: Es una estructura tipo árbol conocida como diagrama organizacional, las hojas representan grupos o departamentos y las ramas muestran la jerarquía, una persona arriba de otra significa que posee mayor autoridad y puede ordenar a las personas que están debajo de ella.
- Organización matricial: Considera dos dimensiones: funcional y jerárquica. La primera se refiere a las tareas que puede realizar una persona de acuerdo a la jerarquía (que es la segunda dimensión) que tiene en la organización.
- Organización de malla: Usuarios autónomos colaboran suministrando productos o servicios dando la sensación de que existe una organización.

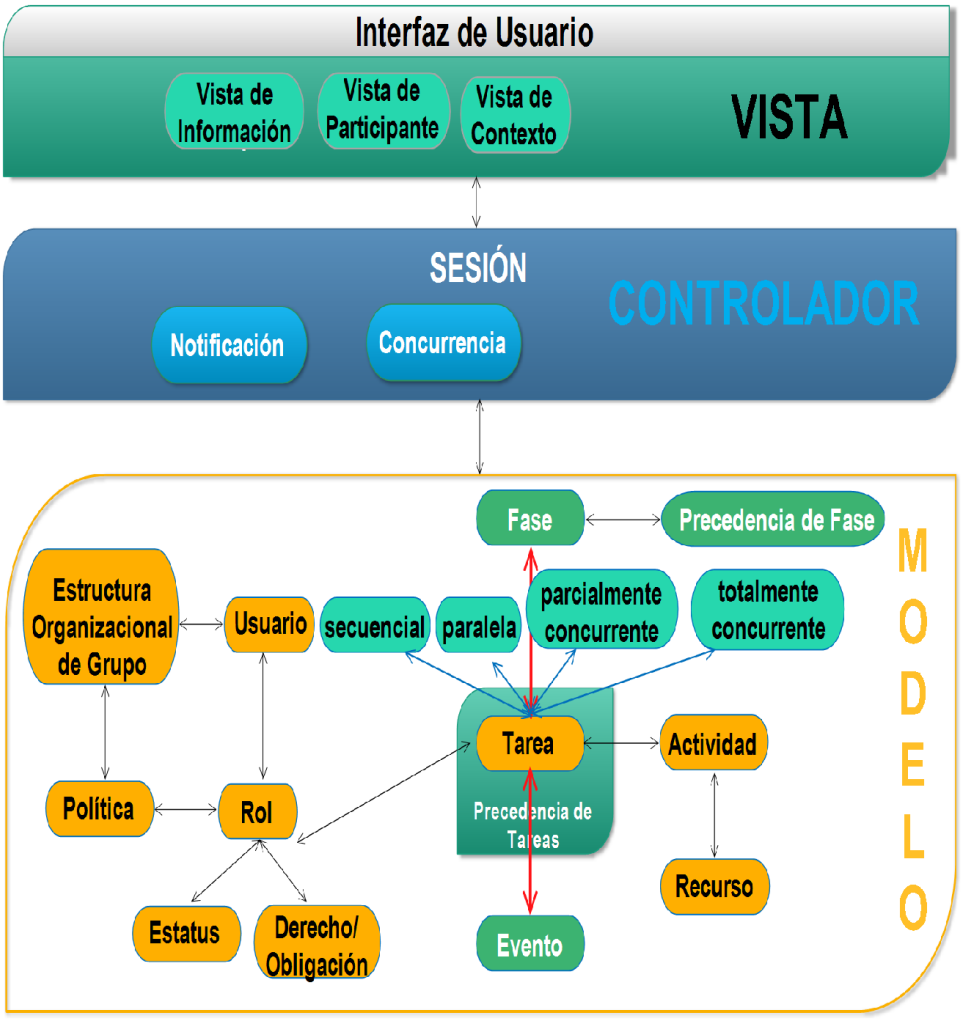


Figura 30. MVC personalizado para desarrollar *groupware*.

La estructura organizacional del grupo se clasifica como de malla, define la división de labor basada en políticas, tomando en cuenta los roles que pueden desempeñar los usuarios; de esta forma, los permisos son asignados a los roles y no a los usuarios, así la política no tienen que ser cambiada cuando un usuario modifica su rol dentro de la organización.

En *groupware* comúnmente se suministra un mecanismo para controlar y gestionar las sesiones, denominado gestor de sesión, que permite definir sesiones por medio de una interfaz de usuario, a través de la cual los usuarios establecen la conexión, se unen a, salen de, detienen y terminan sesiones. Este mecanismo también específica, generalmente, sólo una estructura organizacional del grupo; sin embargo, es importante soportar y definir diferentes maneras en cómo está organizado el trabajo (estilos de trabajo). Por consiguiente, si la política que el sistema impone no es aceptada por el grupo o adecuada para el trabajo en grupo, debería ser cambiada por un estilo apropiado al grupo y a sus necesidades. Esto requiere, por una parte, proporcionar mecanismos que permitan separar la política de la implementación del sistema para cambiar dinámicamente el comportamiento del mismo sin llevar acabo modificaciones sustanciales a su estructura o entorno de ejecución; por otra, se necesitan suministrar modelos conceptuales para especificar de manera independiente la estructura organizacional del grupo, con la finalidad de adaptarla a las necesidades y estilos de diversos grupos. Por consiguiente, en este trabajo de tesis se propone modelar la estructura organizacional regida por una política mediante una ontología.

6.2.1.2. Gestión de sistemas basada en políticas

La gestión de sistemas basada en políticas separa la política de la implementación del sistema. Las políticas proporcionan la flexibilidad y autonomía necesarias para regular sistemas complejos, determinando su comportamiento de acuerdo con un conjunto de reglas, permitiendo especificar dinámicamente el comportamiento de un sistema, así como modificarlo a fin de cambiar sus componentes o el propio sistema sin necesidad de llevar a cabo cambios sustanciales en su estructura o entorno de ejecución [Damianou 01, Bradshaw 03]. Por tanto, la gestión de sistemas basada en políticas es ideal para aplicaciones dinámicas y complejas, como por ejemplo *groupware*, ya que sólo se requiere cambiar la política para que éstas puedan adaptarse continuamente a las restricciones de la estructura organizacional y a las condiciones impuestas externamente.

El enfoque de la gestión de sistemas basada en políticas ha incluido áreas tradicionales, como seguridad, gestión de recursos, intercambio y difusión de información [Edwards 96]. Esto se debe a que proporciona muchos beneficios: comportamiento autónomo, reusabilidad, eficacia, extensibilidad, sensibilidad al contexto, verificabilidad, soporte a componentes simples o sofisticados, protección de un mal diseño, de errores, o de componentes dañinos, y razonamiento acerca del comportamiento de los componentes [Bradshaw 03, Suri 03]. Este enfoque se ha utilizado cada vez más para manejar sistemas automatizados y controlar el comportamiento de sistemas complejos, en áreas como: modelos de organización, interacción humano-computadora, filtrado y transformación de información, computo ágil, sistemas móviles y pervasivos, formalización de acuerdos transversales de organización y modelos de confianza, entre otras.

El dominio *groupware* se ha centrado principalmente en las políticas de coordinación y de seguridad. Las políticas de coordinación permiten la interacción entre los usuarios y de los usuarios con los recursos compartidos, evitando conflictos que conduzcan a la inconsistencia de la información. Los ejemplos de este tipo de política son: control de acceso [Shen 92], control de concurrencia [Ellis 89, Greenberg 94] y *floor control* [Dommel 97]. Las políticas de seguridad son necesarias para definir las acciones que deben tomarse cuando ocurre alguna violación, como la que se produce cuando se originan fallos de conexión de un usuario o aquella que se produce al descubrirse un ataque sobre el sistema.

En *groupware* se necesita definir la manera en que se organizarán las sesiones para llevar a cabo el trabajo en grupo; por tanto, se debe establecer la política de la estructura organizacional del trabajo en grupo, también conocida como política de gestión de la sesión. En la propuesta se han utilizado una gran variedad de políticas (mencionadas en el capítulo 2), como son: puerta abierta, lluvia de ideas, moderada, puntos de reunión y específicas.

Este trabajo de tesis, se centra principalmente en las áreas del modelado organizacional y en la interacción humano-computadora. Se define la gestión de la estructura organizacional de grupo basada en una política, porque permite:

- Establecer el control de acceso a la sesión, determinando quién autoriza el registro de los usuarios, cómo se llevará a cabo la interacción entre los usuarios y cómo se definen los turnos de participación de los mismos.
- Adaptar el comportamiento de la estructura organizacional del grupo sin cambiar el código y sin que sea necesario requerir el consentimiento o la cooperación de los componentes que son controlados.

Sin embargo, la gestión de sistemas basada en políticas, necesita una especificación apropiada de la política. En la literatura, se han propuesto múltiples enfoques de la especificación de la política en los diferentes dominios de aplicación, que van desde el uso de lenguajes formales para la descripción de la política a la utilización de una notación basada en reglas, así como la especificación de las políticas como entradas de una tabla que consta de múltiples atributos. Estos enfoques de la especificación han sido restrictivos en muchos aspectos, de tal manera que la elección de un enfoque debe ser impulsada por las características del dominio de aplicación y por los principales requisitos que toda especificación de la política debe satisfacer [Suri 03]:

- **Simplicidad.** Para facilitar la definición de políticas a los administradores de tareas con diferentes grados de experiencia;
- **Expresividad.** Para gestionar la amplia gama de requisitos de las políticas que surjan en el sistema que se está manejando;
- **Análisis.** Para permitir el razonamiento sobre las políticas;
- **Escalabilidad.** Para garantizar la adecuada ejecución; y
- **Exigibilidad.** Para garantizar un mapeo de la especificación de la política a la implementación de las políticas para varias plataformas.

La gestión de sistemas basada en políticas exige un enfoque semántico que simplifique el análisis de las políticas, reduzca las incoherencias y conflictos de las políticas, facilitando su reutilización a través de diversos sistemas. La ontología es ideal para especificar la gestión de sistemas basada en políticas, ya que según [Suri 03, Gruber 95], ésta permite:

- Simplificar la tarea de gobernar el comportamiento de entornos complejos;
- Utilizar conceptos para describir el dominio y las entidades que son controladas, simplificando su descripción, facilitando un profundo análisis y cuidadoso razonamiento sobre ellas; y
- Simplificar el acceso a la información de la política, con la posibilidad de calcular dinámicamente las relaciones entre las políticas y el entorno, entidades u otras políticas basadas en relaciones de la ontología.

6.2.1.3. Modelo ontológico de la estructura organizacional

La estructura organizacional del trabajo en grupo se específica mediante una ontología, debido a que proporciona:

- Una estructura taxonómica del dominio (estructura organizacional del grupo), lo que facilita la representación del carácter jerárquico de la organización.
- Una especificación explícita de los elementos del dominio por medio de conceptos, relaciones, axiomas e instancias. Los dos primeros componentes de nuestra ontología facilitan el modelado de la política de gestión de la sesión. Los axiomas especifican las condiciones (o restricciones) que deben tenerse en cuenta para llevar a cabo los cambios apropiados de conformidad con la dinámica de grupo y las instancias especifican los individuos de cada estructura organizacional. Los axiomas y las relaciones entre los conceptos proporcionan el carácter dinámico a la asignación de roles aportando flexibilidad para el dominio *groupware*, dando la idea de una organización matricial.
- Conocimiento del estado actual de la organización del trabajo en grupo, como por ejemplo: qué actor está realizando una tarea, con qué rol la está llevando a cabo y qué recurso está utilizando.

La especificación del modelo ontológico de la política de la estructura organizacional del trabajo en grupo [Anzures 07b, Anzures 08a, Anzures 08b] (véase la Figura 31), se utilizó para establecer el control

de la sesión, determinando quién autoriza el registro de un usuario, cómo se realiza la interacción entre los usuarios y cómo se definen los turnos para la participación del usuario en la sesión, así como también para soportar los cambios, determinando cómo se modifica el rol de usuario, los permisos de rol y/o la misma política. Se considera que, en un momento determinado, la estructura organizacional del grupo es gobernada por una política específica, que establece cómo estará organizado el trabajo en grupo. La política establece el control de acceso a la sesión y a los recursos compartidos, al definir una estructura organizacional por medio de los roles que cada usuario puede desempeñar. Los roles establecen el conjunto de derechos y obligaciones del trabajo en grupo, así como también las tareas que pueden ser realizadas por los usuarios con los recursos compartidos.

Posteriormente para suministrar la coordinación a nivel de actividad y a nivel de objeto, se extiende (véase la Figura 32) la estructura organizacional del grupo mostrada en la Figura 31. De tal manera, que se simplifique el control de acceso y la interacción de los usuarios, así como la visualización de la información resultante de dicha interacción.

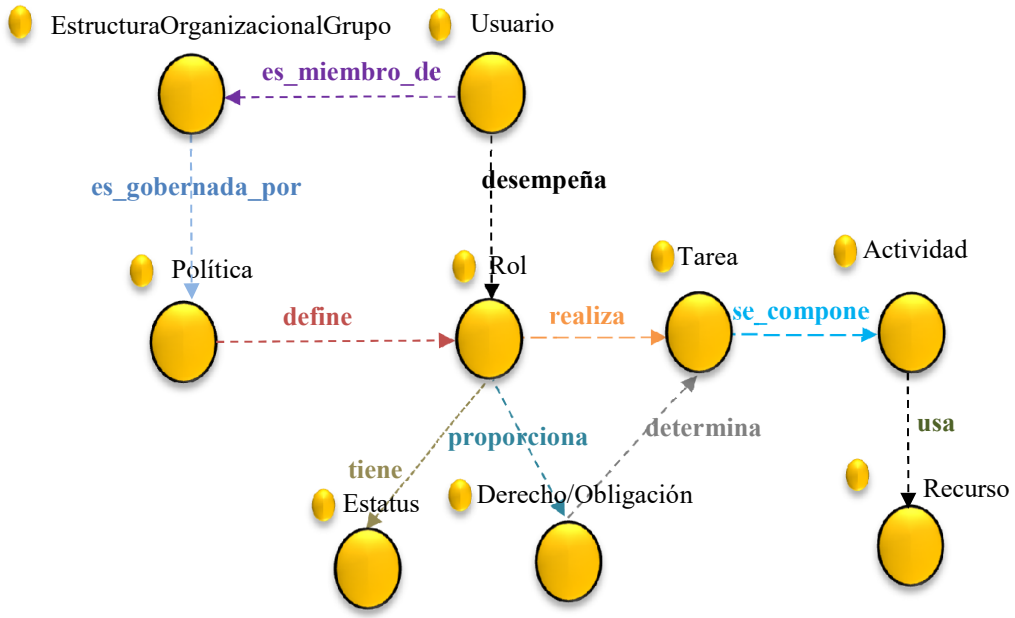


Figura 31. Modelo ontológico de la estructura organizacional de grupo.

Para la coordinación a nivel de actividad se añaden los conceptos de: Evento, que activa cada tarea; precedencia de cada tarea, que permite conocer cuando inicia una o cuando termina, así es sencillo establecer y monitorizar la tarea; y Fase, indica un momento de colaboración, que está constituido de una o más tareas relacionadas con alcanzar un objetivo y el orden entre éstas, de esta forma, contribuye a simplificar la precedencia de tareas, el control de acceso y la interacción de los usuarios; un usuario desempeñando un rol accede a una fase sí y sólo sí debe realizar una o varias tareas.

Con respecto a la coordinación a nivel de recurso (objeto), se agregan cuatro tipos de tareas: Secuencial, se realiza una tarea después de la otra dentro de una Fase, así que las tareas acceden a los mismos o distintos recursos en diferente tiempo; Paralela, las tareas se llevan a cabo al mismo tiempo, pero acceden a distintos recursos; Parcialmente Concurrente, las tareas se ejecutan al mismo tiempo accediendo a un único recurso, pero sólo una de ellas lo modifica; y Totalmente Concurrente, las tareas se producen al mismo tiempo accediendo a un único recurso, pudiéndolo modificar más de una de éstas. El tipo de tarea indica qué interacción deberá efectuar cada rol, así como el mecanismo de sincronización a utilizar (véase en la sección 6.2.3) y la vista o vistas que mostrará la interfaz de usuario de la aplicación (véase en la sección 6.2.2). Además, se añade el concepto Componente para representar un elemento de la interfaz de usuario derivado del recurso compartido utilizado en las actividades.

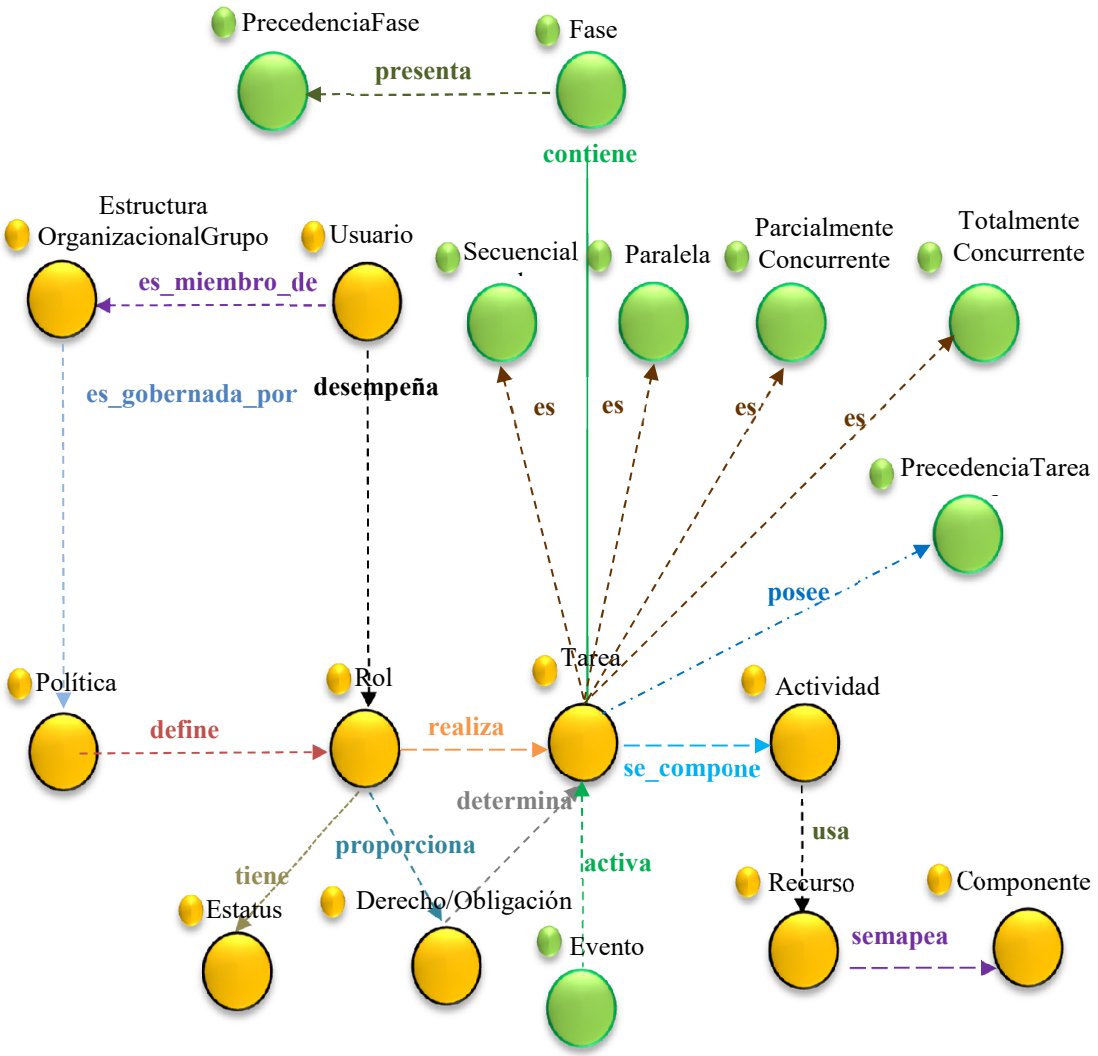


Figura 32. Modelo ontológico de la estructura organizacional extendido.

6.2.1.3.1. Conceptos de la ontología de la estructura

La mayoría de los conceptos que se utilizan en la ontología de la estructura organizacional regida por políticas, han sido usados en varios contextos del dominio HCI y CSCW, con diferentes connotaciones e incluso nombres. Sin embargo, este trabajo de tesis no se centrará en estas diferencias, ni tampoco en exponer las deficiencias (si es que las hubo) en el uso de estos conceptos. Esta tesis presenta los conceptos adecuados al objetivo central, que es simplificar el desarrollo de *groupware* a través de guías claras y fáciles para reducir el tiempo y esfuerzo en este desarrollo. Por tanto, el modelo ontológico que se propone en este trabajo de tesis, contiene los siguientes conceptos (véase la Figura 32):

- **Estructura Organizacional del Grupo (EOG).** Indica el estilo organizacional que tendrá el trabajo en grupo, en este trabajo la estructura se fundamenta en una combinación de la organización matricial y de malla. La Estructura Organizacional del Grupo *es gobernada por* una Política y tiene como *miembros* a Usuarios.
- **Política**. Determina la configuración de la estructura organizacional del trabajo en grupo, que puede ser modificada sin necesidad de llevar a cabo cambios sustanciales en su estructura o su entorno de ejecución. La Política *gobierna* la Estructura Organizacional del Grupo y *define* los Roles que los Usuarios pueden desempeñar.
- **Usuario**. Es un participante en el trabajo en grupo, que puede ser una persona, un conjunto de éstas o un elemento del sistema (agente o servicio Web), que realiza una función dentro de la

Estructura Organizacional del Grupo. Existen varios términos usados para denotar al participante, como son: actor, agente, usuario, persona quién desempeña el trabajo o el rol. En esta tesis, se usa **Usuario** porque es un participante que **usa** una aplicación o un recurso (a través del Rol que desempeña) para llevar a cabo el trabajo colaborativo. De esta manera, un Usuario *es miembro de* una Estructura Organizacional del Grupo y *desempeña* uno o más Roles.

- **Rol**. Es la función (ocupación o puesto dentro de una organización) que un usuario desempeña para llevar a cabo diferentes tareas dentro de la Estructura Organizacional del Grupo y alcanzar un objetivo común. De esta forma, los permisos son asignados a los roles y no a los usuarios, proporcionando mayor flexibilidad a la estructura organizacional. Por consiguiente, el Rol *es definido por* la Política, *desempeñado por* Usuarios, *proporciona* un conjunto de Derechos/Obligaciones que le permiten *realizar* ciertas Tareas y *tiene* un Estatus dentro de la organización.
- **Estatus**. Define la línea de autoridad del Rol, de acuerdo a la posición de éste dentro de la organización. Esto permite establecer de manera clara una estructura jerárquica. El Estatus determina, en un instante dado, quién es el Usuario que controla o dirige la Sesión y quién debe tomar su lugar en caso de que el Usuario con el Estatus más alto abandone la sesión. En caso de que se trate de una política con Usuarios con el mismo Estatus (igual jerarquía), los conflictos se resuelven a través de la secuenciación de solicitudes para acceder al recurso compartido, siguiendo el criterio "el primero que lo solicite será el primero que pueda utilizarlo".
- **Derecho/Obligación** (D/O). Indica los permisos y responsabilidades del Usuario de acuerdo al Rol que desempeña. Este D/O se define en base al Rol y no al Usuario que tiene asignado el mismo; por tanto, la Política no debe modificarse cuando los Usuarios cambian su Rol dentro de la organización. Este Derecho/Obligación restringe las acciones del Usuario para interactuar con los demás Usuarios y desempeñar las Tareas.
- **Tarea**. Indica un conjunto de Actividades que se llevan a cabo para alcanzar un objetivo específico. Una Tarea *es activada* por un Evento, *realizada* por un Rol y *determinada* por el Derecho/Obligación. Una Tarea *se compone* de Actividades (al menos una). Una o más Tareas *constituyen* una Fase, que indica un momento colaborativo. Una Tarea puede ser Secuencial (una tarea se realiza después de otra), Paralela (varias tareas se realizan al mismo tiempo pero no existe relación entre ellas), Parcialmente Concurrente (las tareas se ejecutan al mismo tiempo, pero no modifican el mismo recurso) y Totalmente Concurrente (las tareas se ejecutan al mismo tiempo, accediendo al mismo recurso). Este trabajo de tesis se centra en la interacción generada por el trabajo en grupo, por tanto, no divide las tareas en atómicas, simples o compuestas, sino en secuencial, paralela, parcialmente concurrente y totalmente concurrente lo cual dará la pauta para simplificar la gestión de la interacción. En este trabajo de tesis, cada Tarea indica la interacción entre los usuarios o de éstos con la aplicación colaborativa; determinando de esta forma la configuración de la navegación y las interfaces de usuario de dicha aplicación.
- **Precedencia de Tarea**. Se refiere al orden en el cual se va a ejecutar una Tarea en una Fase determinada; en primer lugar, segundo, etc.
- **Evento**. Activa una Tarea, indicado el inicio de la misma. El Evento es vital para gestionar y monitorizar la precedencia de Tareas, porque muestra cuando debe comenzar una Tarea y luego otra.
- **Fase**. Indica un momento de colaboración, que es representado por un conjunto de Tareas relacionadas con el objetivo de esa colaboración. Definimos *groupware* de Fase Simple, cuando existe un sólo momento de colaboración, y de Fase Compuesta, cuando existen más de uno de estos momentos. En la actualidad, la mayoría de *groupware* es de Fase Compuesta porque son aplicaciones complejas que deben subdividirse para alcanzar el objetivo común. Una Fase *se constituye de* una o más Tareas y se presenta en una Sesión.
- **Precedencia de Fase**. En caso de que exista más de una Fase y se refiere al orden en el cual se va a llevar a cabo una Fase en una sesión determinada; en primer lugar, segundo, etc.
- **Tarea Secuencial**. Hace referencia a una Tarea que antecede o precede a otra, es decir hay una tarea inicial y una final dentro de una Fase; estableciendo así un orden de ejecución de las Tareas y facilitando su especificación con un *workflow*. Este tipo de Tarea requiere un servicio de

Notificación, para indicar a la aplicación colaborativa que la tarea está en ejecución o que ha terminado, de tal manera, que permite iniciar a la siguiente Tarea; así como también para actualizar la Vista de Información del Usuario que la realiza a través del Rol que desempeña.

- **Tarea Paralela.** Son aquellas Tareas que se ejecutan al mismo tiempo, pero utilizan diferentes recursos y no existe ninguna relación entre ellas. Esta Tarea hace uso de un servicio de Notificación con la finalidad de indicar al entorno de ejecución que actualice la Vista de Información y de Participante (mostrando los cambios producidos a los demás usuarios).
- **Tarea Parcialmente Concurrente.** Hace referencia a Tareas que se llevan a cabo al mismo tiempo, modificando el mismo recurso, pero diferente parte o sección del mismo. Esta Tarea activa el servicio de Notificación para modificar la Vista de Información, Vista de Participante y Vista de Contexto (para registrar todos los cambios hechos a los recursos compartidos por diferentes usuarios).
- **Tarea Totalmente Concurrente.** Hace referencia a una Tarea que se realiza al mismo tiempo que otras Tareas y que modifican el mismo recurso. Esta Tarea activa el servicio de Notificación para modificar la Vista de Información, de Participante y de Contexto; además usa el servicio de Concurrencia para sincronizar y evitar inconsistencias al utilizar un recurso al mismo tiempo.
- **Actividad**. Consiste de acciones o cadena de acciones, que a su vez contienen operaciones, que se realizan para lograr un objetivo. Las operaciones están relacionadas con la interacción del usuario a través de la interfaz de usuario, por ello, se consideran: crear, capturar, seleccionar, obtener, actualizar, modificar, consultar, eliminar, enviar, aceptar, etc. Sin embargo, las acciones y operaciones no son modeladas en la ontología, porque se consideran parte implícita de las actividades y concernientes explícitamente con la implementación. Una o más Actividades componen a una Tarea y *usan* uno o más Recursos.
- **Recurso.** Representa los recursos compartidos utilizados en las actividades realizadas por el rol o roles para alcanzar los objetivos comunes fijados dentro de la estructura organizacional del grupo. Además, en este trabajo de tesis, también se consideran como recursos los diferentes elementos que permiten realizar las actividades en la interfaz de usuario. También, un recurso es mapeado como un componente de la interfaz de usuario.
- **Componente.** Se refiere a los elementos que constituyen la interfaz de usuario, en el momento de establecer el diseño del *groupware.*

6.2.1.3.2. Relaciones de la ontología de la estructura

Los conceptos definidos en la anterior subsección están asociados por las siguientes relaciones, como muestra la Figura 32:

- *esgobernadapor(EstructuraOrganizacionalGrupo, Política).* Especifica que la estructura organizacional del grupo es gobernada por una política en un momento determinado.
- *esmiembro(Usuario, EstructuraOrganizacionalGrupo).* Define que el usuario es miembro de la estructura organizacional del grupo.
- *define(Politica, Rol).* Específica que la política define el rol o roles que pueden desempeñar los usuarios.
- *desempeña(Usuario, Rol).* Indica que el usuario puede desempeñar uno o más roles en una organización o comunidad.
- *tiene(Rol, Estatus).* Determina que el rol tiene un estatus dentro de la estructura organizacional del grupo.
- *proporciona(Rol, Derecho/Obligación).* Especifica que el rol proporciona un conjunto de derechos y obligaciones al usuario.
- *determina(Derecho/Obligación, Tarea).* Indica que el Derecho/Obligación determina la tarea que un usuario puede llevar a cabo.
- *realiza(Rol, Tarea).* Esta relación indica que un Rol realiza una o varias Tareas.
- *activa(Evento, Tarea).* Indica que un evento activa el inicio de una Tarea.
- *se_compone(Tarea, Actividad).* Señala que una Tarea se compone de una o varias Actividades.

- *posee(Tarea, PrecedenciaTarea).* Establece que una Tarea tiene una Precedencia de Tarea.
- *es(Tarea, Secuencial).* Define que una Tarea es Secuencial.
- *es(Tarea, Paralela).* Define que una Tarea es Paralela.
- *es(Tarea, ParcilamenteConcurrente).* Define que una Tarea es Parcialmente Concurrente.
- *es(Tarea, TotalmenteConcurrente).* Define que una Tarea es Totalmente Concurrente.
- *contiene(Fase, Tarea).* Indica que una Fase contiene una o varias Tareas.
- *presenta(Fase, PrecedenciaFase).* Establece que una Fase presenta una Precedencia de Fase.
- *mapea(Recurso, Componente).* Se refiere a que un recurso se mapea como un componente de la interfaz de usuario del groupware.

6.2.1.3.3. Axiomas de la ontología de la estructura

En esta sección, vamos a presentar las restricciones de la ontología de la política de manejo de sesión (mostrada en la Figura 32):

- La EOG *es gobernada* sólo por una Política.
- Al menos dos Usuarios *son miembros* de una EOG.
- Una Política *define* por lo menos un Rol.
- Un Usuario (persona, individuo, grupo o sistema) *desempeña* uno o varios Roles en diferente tiempo.
- Un Rol *tiene* un Estatus.
- Un Rol proporciona un Derecho/Obligación.
- Un Rol *realiza* al menos una Tarea.
- Un Evento *activa* una Tarea.
- Una Tarea *posee* una Precedencia de Tarea.
- Una Tarea *se compone* de al menos una Actividad.
- Una Actividad *usa* al menos un recurso.
- Una Tarea *es* Tarea Secuencial (TS), Tarea Paralela (TP), Tarea Parcialmente Concurrente (TPC) o Tarea Totalmente Concurrente (TTC).
- Una Fase *posee* una Precedencia de Fase.
- Una Fase *contiene* al menos una Tarea.

6.2.2. El componente vista

Las vistas conforman la interfaz de usuario, que muestra los recursos, la interacción entre los usuarios, así como la interacción entre éstos y la aplicación. La sesión, espacio de trabajo compartido, es proporcionada a través de las vistas, permitiendo la interacción de los usuarios con la aplicación. El controlador gestiona la información del modelo para presentar la vista apropiada, de esta forma, se pueden generar varias vistas mediante un mismo modelo. En el caso de aplicaciones colaborativas, se consideran tres vistas (véase la Figura 33): la Vista de Información (VI), la Vista de Participantes (VP) y la Vista de Contexto (VC). La interfaz de usuario puede mostrar tres, dos o una vista, de acuerdo con la tarea ejecutada, ya que una tarea puede determinar el tipo de coordinación (nivel de actividad o nivel de objeto) a ejecutar.

Figura 33. Componente Vista del MVC propuesto.

- **La Vista de Información.** Permite ver toda la información que ayuda al usuario a interactuar con la aplicación colaborativa; por esta razón, muestra todas las tareas realizadas por un rol sobre los recursos compartidos y donde otros roles no tienen participación. Esta vista muestra las modificaciones relacionadas con las tareas realizadas en el entorno colaborativo.
- **La Vista de Participantes.** Permite a cada usuario ser consciente de lo que hacen los demás usuarios y así saber lo que ocurre en el espacio de trabajo compartido ya que el grupo se encuentra distribuido (generalmente) en diferentes ubicaciones geográficas, trabajando a distancia a través de la computadora. En consecuencia, esta vista proporciona la *conciencia del grupo* por medio de las vistas que muestran las notificaciones de los distintos cambios producidos por otros usuarios del grupo, que están participando en la aplicación. Por ello, el nombre de "Vista de Participantes". La aplicación colaborativa utiliza *widgets* para mostrar a otros usuarios todo lo que está sucediendo en la aplicación. Cada actividad colaborativa activa una notificación, que envía mensajes para actualizar los *widgets* y/o elementos que contiene cada vista, de acuerdo a la modificación del recurso o trabajo de grupo, promoviendo una mayor cooperación e interacción entre los usuarios. Para ello, se utilizan mecanismos de concurrencia, que ayuda a gestionar los permisos asignados a los usuarios, con el fin de hacer uso de los recursos compartidos; asegurando su uso mutuamente exclusivo
- **La Vista de Contexto.** Representa el espacio de trabajo común donde se muestra toda la información de los recursos compartidos. Esta vista se denomina memoria o historial de grupo, que muestra las actividades de cooperación del grupo, para proporcionar la comprensión y el entendimiento sobre el proceso de colaboración, con la finalidad de llevar a cabo un control preciso. Además, permite almacenar, recopilar y distribuir información para soportar la representación de conocimiento del grupo, de manera estructural y dinámica. Se genera la memoria de grupo para almacenar la información de los recursos compartidos utilizados y las actividades realizadas por cada rol implicado en la aplicación colaborativa. Esta vista muestra los cambios implementados por la tarea parcialmente concurrente y tarea totalmente concurrente.

6.2.3. El componente controlador

El controlador gestiona y actualiza las vistas de forma adecuada de acuerdo con la modificación de los recursos compartidos del modelo, que resulta de la interacción de los usuarios. En las aplicaciones colaborativas, la interacción se realiza durante la sesión, y las interfaces de usuario (que componen el mismo) muestran los cambios del modelo.

El controlador (véase la Figura 34) lleva a cabo el mecanismo de notificación, actualizando la vista de información para el caso de una tarea secuencial, así como la vista de participante y la vista de contexto, cuando se produce una tarea paralela o concurrente (parcial o total). En este último caso, también se activa el mecanismo de concurrencia. Este mecanismo evita los conflictos cuando un recurso está siendo utilizado por varios participantes.

Figura 34. Componente Controlador del MVC propuesto.

La **sesión** establece un espacio de trabajo compartido, en el que un grupo de personas comparte un interés común, para ello, suministra un mecanismo de gestión de sesiones para controlarlas y manejarlas a través de la interfaz de usuario mediante la cual los usuarios establecen una conexión; es decir, para que los usuarios se unan, salgan, inviten y excluyan a alguien de una sesión. Generalmente, este mecanismo también especifica cómo se organizará el grupo. En la actualidad, la gestión de la sesión está desacoplada

de la estructura organizacional del grupo, ya que la sesión se implementa con los métodos o funciones proporcionados por el *framework* o lenguaje de programación utilizados, mientras la estructura organizacional es realizada por el desarrollador.

Se produce una **notificación** cuando se modifican los recursos compartidos para mantener actualizadas las vistas. La notificación debe proporcionar información sobre los eventos en una sesión relacionados con un usuario, un subconjunto de usuarios o un grupo completo. Esto es esencial para coordinar las actividades de los usuarios y proporciona *la conciencia del grupo*, es decir, un usuario sabe lo que otros están haciendo en un espacio de trabajo compartido a través de eventos o widgets presentados en la interfaz de usuario. Además, cada evento que se produce se puede almacenar y mostrar en la vista de contexto, generando la memoria del grupo.

La **concurrencia** determina de qué manera contribuyen los participantes en una sesión, proporcionando permisos temporales de acceso y manipulación generados dinámicamente para usuarios colaboradores, con el fin de reducir las condiciones de competencia y garantizar el uso de recursos mutuamente excluyentes. Los permisos se conceden a los usuarios en función de los roles que desempeñan; de tal manera que estos permisos especifican qué usuarios pueden enviar, recibir o manipular datos compartidos en una fase determinada.

La notificación y concurrencia, controlan la interacción que se produce durante la sesión y adaptan las vistas correspondientes de acuerdo con el tipo de tarea ejecutada. Estos mecanismos deben ser implementados en cada función o método que represente una tarea. La adaptación de las vistas de acuerdo con la interacción realizada se lleva a cabo a través de la notificación y se controla por la concurrencia.

6.3. El modelo ontológico *workflow* del enfoque metodológico

Una vez definidos y organizados mediante el estilo arquitectónico MVC los elementos que permiten desarrollar un *groupware,* se establece la ontología que indica los pasos que se deben seguir para lograr este desarrollo.

La estructura organizacional del grupo en el dominio CSCW determina cómo se realiza la comunicación, la colaboración y la coordinación entre los miembros del grupo; facilitando y simplificando cada uno de estos procesos. De esta manera, la gestión de esta estructura es crucial para este dominio. Sin embargo, especificar y modelar dicha gestión es bastante complejo, especialmente cuando se requiere una estructura dinámica. Este dinamismo se refiere a cambiar la estructura de acuerdo a las tareas y necesidades del grupo en un momento dado, así como a adaptarlo dinámicamente para hacer frente a una organización cambiante y a las condiciones propias de la aplicación.

Por lo tanto, es necesario un esquema de representación del conocimiento que soporte un modelo adaptable y formal. El esquema debe proporcionar un conjunto de procedimientos que permita que el conocimiento sea almacenado, organizado y gestionado de forma natural. De esta forma se requiere una base de conocimiento para especificar la estructura organizacional del grupo.

La base del conocimiento puede ayudar a entender, administrar y controlar cada proceso realizado por las organizaciones o comunidades. También nos sirve para obtener conclusiones a través de un mecanismo de inferencia para los contenidos de la base de conocimiento [Anzures 15a]. El esquema de representación debe ser denotado por un modelo de algún dominio de interés en el cual los símbolos sirvan como sustitutos de artefactos del mundo real. Estos símbolos deben ser almacenados como declaraciones de dominio de interés; por tanto, las ontologías son esquemas de representación del conocimiento ideales [Grimm 07].

La ontología es ideal para establecer una base de conocimiento que permita especificar y modelar la estructura organizacional del grupo, a través de un modelo formal y explícito. Así, esta estructura puede ser adaptada; cambiando, agregando o eliminando algunos de los elementos que la constituyen.

El proceso de gestión de esta estructura puede caracterizarse por un modelo *workflow*, que especifica como las entidades que lo componen deben ser utilizadas y combinadas [Fischer 04, Marinescu 02].

En este trabajo, se propone una ontología *workflow* basada en el conocimiento para gestionar la estructura organizacional del grupo. De esta manera, el conocimiento es especificado de manera explícita

y formal. Utilizando este esquema de representación del conocimiento, la aplicación puede adaptarse a los cambios frecuentes en la estructura organizacional.

La ontología *workflow* para el desarrollo de *groupware* (véase la Figura 35) es una extensión de la ontología de la política de manejo de sesión (refiérase a la sección 6.2.1). En dicha extensión se agregan los conceptos para el componente Vista y el componente Controlador, tomándose como base dicha ontología de la política, que corresponde al componente Modelo.

La ontología *workflow* propuesta es**:**

- **Completa.** Especifica un conjunto de conceptos comunes relacionados con la construcción de *groupware*, y los términos para referirse a éstos.
- **No ambigüa.** Proporciona definiciones simples y concisas para cada concepto.
- **Intuitiva.** Suministra conceptos conocidos en el dominio de *groupware*, que pueden ser comprendidos intuitivamente por los desarrolladores de este dominio.
- **Genérica.** Contiene los conceptos esenciales para el desarrollo de *groupware*.
- **Extensible.** Permite añadir nuevos conceptos sin afectar los existentes, muestra de ello, es que esta ontología *workflow* es una extensión de la ontología propuesta para la estructura organizacional del grupo [Anzures 14b, Anzures 07b].

Por otra parte, nuestra ontología *workflow* suministra:

- **Expresividad.** Para representar las entidades de una organización y establecer los pasos a seguir de cómo éstas deben ser utilizadas y combinadas.
- **Verificación.** La exactitud de la especificación del *workflow* es verificada mediante el uso de un razonador.
- **Gestión de cambios.** Es posible ajustar los elementos de la ontología según sea necesario, cambiando, agregando o eliminando conceptos, relaciones, axiomas y/o instancias.

La ontología *workflow* como modelo de la estructura organizacional del grupo específica los siguientes aspectos:

1. **Individuales y de grupo.** El carácter individual se modela al establecer el rol con sus derechos, obligaciones y estatus, así como las Tareas que puede desempeñar, las actividades a realizar y los recursos a utilizar. El sentido de grupo se refleja al indicar las Tareas que contiene una Fase y que las mismas se realicen de manera secuencial, paralela, parcialmente concurrente o totalmente concurrente.
2. **Interactivos**. La interacción se determina cuando se muestran los recursos compartidos en las Vistas de Participante y de Contexto, por medio de la Conciencia de Grupo y Memoria de Grupo, respectivamente. Además, la interacción es controlada y gestionada mediante los mecanismos de notificación y concurrencia que producen tanto la vista de participante como la vista de contexto.
3. **Modelado de tareas.** El tipo de tarea establece el carácter individual o interactivo del trabajo a realizar en la aplicación colaborativa. A partir, de este tipo se determina el mecanismo a utilizar notificación y/o concurrencia, así como la vista a presentar.

En las siguientes secciones se presentarán los conceptos, relaciones y axiomas de la ontología *workflow,* pero sólo aquellos que corresponden al componente Vista y Controlador , ya que del componente Modelo ha sido descrito en la sección 6.2.

6.3.1. Conceptos de la ontología *workflow*

Sólo se presentan los conceptos que complementan la ontología *workflow* (véase la Figura 35), y son:

- **Sesión.** Es un espacio de trabajo compartido, dónde se realiza la interacción de los usuarios a través de los roles que desempeñan.
- **Notificación.** Mecanismo para informar a los participantes o entidades que un recurso ha sido modificado en la Sesión.
- **Concurrencia.** Mecanismo para sincronizar la coordinación de los usuarios al utilizar los recursos compartidos; evitando así inconsistencias en la Sesión.

- **Interfaz de Usuario.** Presenta la interacción entre los Usuarios y de éstos con la aplicación a través de la Vista de Información, de Participantes y de Contexto.
- **Vista de Información.** Presenta los objetos colaborativos y las operaciones sobre éstos.
- **Vista de Participante.** Facilita la comunicación persona-a-persona suministrando widgets, audio, vídeo, etc., los cuales proporcionan la **conciencia de grupo**, es decir, permite a un usuario ser consciente sobre qué personas están participando y qué están haciendo.
- **Vista de Contexto.** Suministra **la memoria de grupo**, es decir, el material histórico de la sesión que es útil para realizar de manera efectiva el trabajo en grupo

6.3.2. Relaciones de la ontología *workflow*

Las relaciones que complementan la ontología *workflow* son (véase la Figura. 35):

- *participa(Rol, Sesion)***.** Determina que el Rol *participa* en una Sesión.
- *se_realizan(Fase, Sesion)***.** Señala que una o varias Fases *se realizan* en una Sesión.
- *necesita(Sesión, Notificación)***.** Especifica que una Sesión necesita un mecanismo de Notificación.
- *requiere(Sesion,* Concurrencia*)***.** Indica que una Sesión requiere un mecanismo de Concurrencia para evitar inconsistencias en la misma.
- *precisa(Tarea, Notificación).* Establece que una Tarea *precisa de* un mecanismo de Notificación.
- *solicita(Tarea, Concurrencia).* Define que una Tarea Parcialmente o Totalmente Concurrente *solicita* un mecanismo de Concurrencia.
- *modifica(Notificación, InterfazUsuario).* Determina que un mecanismo de Notificación que modifica la Interfaz de Usuario.
- *actualiza(Concurrencia,Vista).* Determina que un mecanismo de *Concurrencia* actualiza la Vista de Participantes y la Vista de Contexto.
- *se_visualiza_en(Sesión, InterfazUsuario).* Indica que una Sesión se visualiza en una Interfaz de Usuario.
  muestra(*InterfazUsuario, Vista).* Determina que una Interfaz de Usuario muestra la Vista, que puede ser de Información, de Participantes y/o de Contexto.

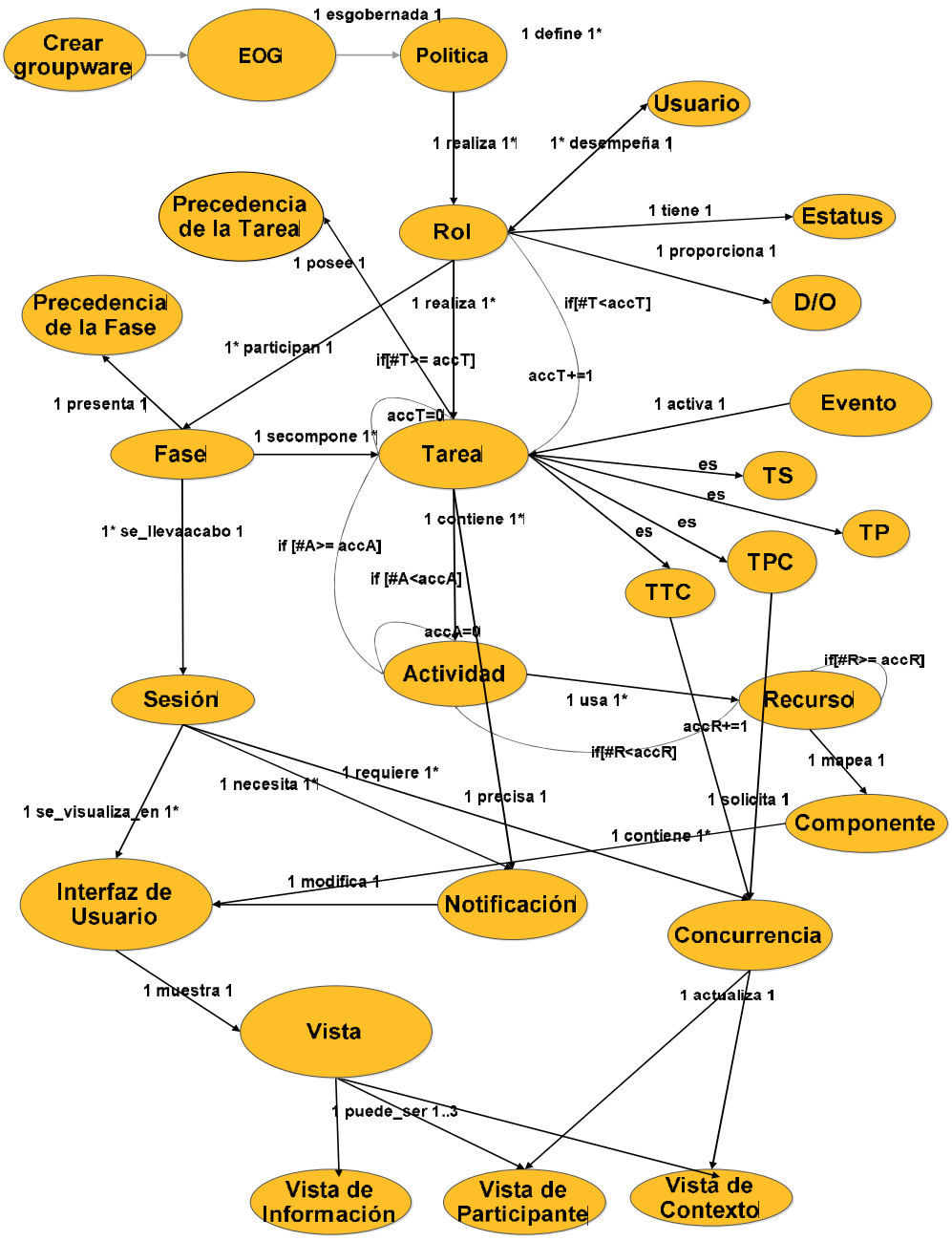


Figura 35. Ontología *workflow*.

6.3.3. Axiomas de la Ontología *workflow*

Los axiomas que complementan la ontología *workflow* son (véase la Figura. 35):

- Uno o varios Roles *participan* en una Sesión.
- Una Tarea *genera* un mecanismo de Notificación.
- Una Tarea Parcialmente o Totalmente Concurrente *produce* un mecanismo de Concurrencia.
- Una Sesión es *constituida* por una o varias Fases.
- Una Sesión se visualiza en una Interfaz de Usuario.
- Una Interfaz de Usuario muestra una o varias Vistas de Información, de Participantes y/o de Contexto.
- Un mecanismo de Notificación modifica la Interfaz de Usuario.
- Un mecanismo de *Concurrencia* actualiza la Vista de Participantes y la Vista de Contexto.

- Una Tarea se muestra en una Vista de Información.
- Una Tarea Parcialmente o Totalmente Concurrente se muestra en una Vista de Participantes y de Contexto
- Una o varias Fases *se realizan* en una Sesión.
  Una Sesión necesita uno o varios mecanismos de Concurrencia para evitar inconsistencias en la misma.

6.4. Pasos del enfoque metodológico

A partir de la descripción de la ontología *workflow*, se establecen los pasos a seguir del enfoque metodológico para desarrollar *groupware*:

1. Iniciar desarrollo
2. Especificar el nombre del *groupware*
3. Establecer la Estructura Organizacional del Grupo.
4. Establecer el nombre de la Política.
5. Definir los roles que contendrá la política y que participaran en el *groupware*.
   5.1. Indicar el Usuario quién *desempeñará* un Rol.
   5.2. Designar el Estatus que *tiene* el Rol.
   5.3. Proporcionar un Derecho/Obligación al Rol.
   5.4. Especificar las Tareas que *realizará* el Rol.
      5.4.1. Señalar el Evento que *activa* una Tarea.
      5.4.2. Decretar si la tarea es TS, TP. TPC o TTC.
      5.4.3. Determinar la Actividad que *compone* a la Tarea.
         5.4.3.1. Establecer el Recurso que se *usa* en la Actividad.
         5.4.3.2. En caso de existir otro Recurso ir al paso 5.3.3.1, si no, ir al paso 5.4.4.
      5.4.4. En caso de existir otra Actividad de la Tarea ir al paso 5.4.3, si no, ir al paso 5.5.
   5.5. En caso de existir otra Tarea para el Rol ir al paso 5.4, si no ir al paso 6.
6. En caso de existir otro Rol ir al paso 5, sino ir al paso 7.
7. Establecer las Fases de la aplicación colaborativa.
   7.1. Indicar la prioridad (orden de ejecución) de cada Fase (Precedencia de Fase).
   7.2. Asignar la Tarea a una Fase.
      7.2.1. Definir la prioridad (orden de ejecución) de la Tarea en cada Fase (Precedencia de Tarea).
      7.2.2. En caso de existir otra Tarea para la Fase ir al paso 7.2, si no, ir al paso 7.2.3.
      7.2.3. En caso de existir otra Fase ir al paso 7.1, si no, ir al paso 8.
8. Especificar el método de notificación de la Tarea, para modificar la Vista correspondiente (Vista de Información, de Participantes y/o de Contexto).
   8.1. En caso de que la Tarea sea Parcialmente Concurrente y Totalmente Concurrente definir o utilizar un mecanismo de Concurrencia (que modifica la Vista).
   8.2. En caso de existir otra Tarea para una Fase ir al paso 8, si no, ir al paso 9.
9. Crear la o las Interfaces de Usuario de una Fase.
   9.1. Generar o modificar la Interfaz de Usuario correspondiente a una Tarea.
      9.1.1. Para la Actividad de la Tarea crear o modificar la Vista correspondiente de acuerdo al Componente que contendrá la misma.
      9.1.2. En caso de existir otro Componente ir al paso 9.1.1, si no, ir al paso 9.1.3.
      9.1.3. En caso de existir otra Actividad ir al paso 9.1.1, si no, ir al paso 9.1.4.
      9.1.4. En caso de existir otra Tarea ir al paso 9.1, si no, ir al paso 9.2.
   9.2. En caso de existir otra Fase ir al paso 9, si no, ir al paso 10.
10. Terminar desarrollo de *groupware*.

Por lo tanto, para generar la especificación del *groupware,* se tendrían que añadir las instancias en la ontología *workflow* generada en la herramienta *Protégé*, en el orden indicado en el algoritmo anterior. Esto

requiere que el desarrollador maneje y éste familiarizado con dicha herramienta; de lo contrario será un trabajo muy tedioso. Por tanto, para simplificar y agilizar el establecimiento de las instancias de dicha ontología se proponen unos *Templates* de Especificación (que se explican en la sección 6.5). Estos *templates* se basan en el estilo arquitectónico MVC, en MetaOntology [Gómez-Pérez 04] y en metodologías ágiles [Stellman 14], que se proponen.

6.5. Enfoque metodológico basado en la ontología *workflow*

El enfoque metodológico está basado en el ciclo de vida de desarrollo de software [Sommerville 15], metodologías ágiles [Stellman 14], ontología *workflow* (presentada en la sección 6.2) y estilo arquitectónico MVC (véase la sección 6.1).

El enfoque metodológico permite especificar los requisitos, delinear el diseño y establecer las pautas de la implementación en tres etapas (véase la Figura 36): Especificación de Requisitos, Especificación de Diseño y Especificación de la Implementación, gracias a que:

- Coloca los elementos (o instancias de la ontología *workflow*), en tres tablas de acuerdo al estilo MVC, que permiten especificar el *groupware* a desarrollar.

Los elementos de las tres tablas ayudan a determinar los datos que deben ser almacenados o procesados para permitir el correcto funcionamiento del *groupware.* De esta forma, se establece y diseña el modelo de datos requerido para la aplicación.

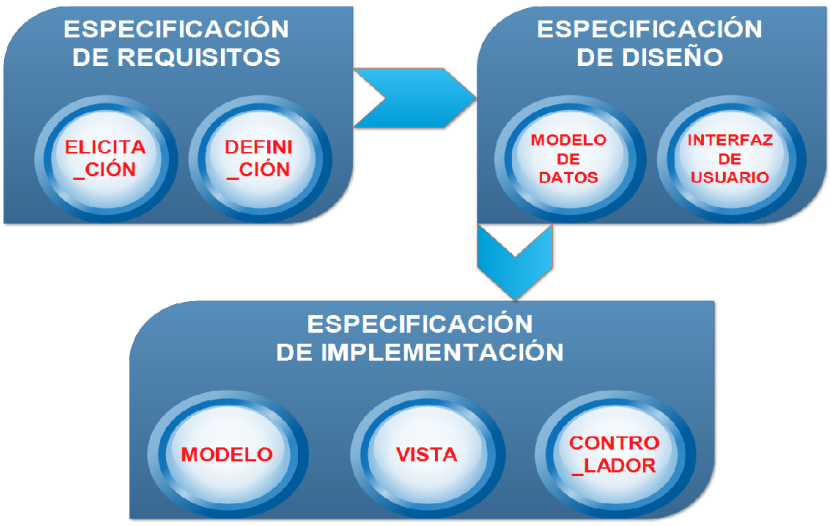


Figura 36. Etapas del enfoque metodológico.

- Establece el control de acceso a la aplicación colaborativa, ya que estas tablas se clasifican por fase, describiendo los roles (de los usuarios) que pueden participar en cada fase.
- La especificación de las tareas que tendrá cada fase permite determinar las interfaces de usuario de la aplicación.
- Los recursos especificados en la Tabla del Modelo y utilizados en las actividades determinan los elementos tendrán las interfaces de usuario de la aplicación, suministrando un esbozo de cada interfaz.
- Los tipos de tarea establecen el mecanismo a ejecutar: notificación y/o concurrencia.
- El mecanismo de notificación debe ser parte del método que permita implementar una tarea.
- El mecanismo de Concurrencia se implementa o utiliza (del *framework,* plataforma o lenguaje utilizado en el desarrollo de la aplicación) cuando se trata de una tarea parcialmente o totalmente concurrente.
- Los tipos de tarea, también, implantan el tipo de vista a presentar (información, participantes y/o contexto).

Con el fin de facilitar el desarrollo de las aplicaciones colaborativas, este enfoque suministra *templates* derivados de las ontologías presentadas en este capítulo, permitiendo establecer las instancias (elementos) de la aplicación de una manera intuitiva y práctica. Como el enfoque metodológico semántico se basa en el estilo arquitectónico MVC, entonces se generan cuatro *templates*: dos corresponden al componente Modelo (*templete de la división de labor* —estructura organizacional de grupo— y *templete del control de acceso* —información de cada Fase), uno al componente Vista y otro al Controlador. Estos *templetes* permiten;

- especificar el análisis de requisitos, al contener todos los elementos (instancias de la ontología *workflow*) que conforman el *groupware* a desarrollar;
- representar el diseño del *groupware,* a través principalmente de los *templates* Vista y Modelo, de este último destacan los conceptos de: Componente (que determinan, precisamente, los elementos de una interfaz de usuario; Fase (el diseño de interfaz se puede agrupar por Fases de acuerdo a los Roles que en ellas participan); Rol (a través de los cuales interactúa el usuario con la aplicación) y Tarea (las actividades, que son acciones u operaciones que realiza el Rol en esa aplicación).
- Realizar la implementación, considerando el *template* Modelo para establecer la base de datos (entidad-relación u orientada a objetos) requerida, los cuatro *templates* para la codificación de procedimientos o métodos y para la "maquetación" o creación de interfaces de usuario los *templates* Modelo y vista.

6.5.1. Especificación de requisitos

En esta etapa, en primer lugar se lleva a cabo la obtención (Elicitación) de requisitos, para ello, se recomienda utilizar técnicas como: etnografía, entrevistas, encuestas, diagramas de casos de uso (UML), historias de usuario, etc. La técnica más apropiada para el enfoque metodológico, propuesto en este trabajo, son las Historias de Usuario, porque permiten establecer la lista de tareas de cada usuario junto con las pruebas unitarias [Sommerville 15]; que finalmente las tareas (e información relacionada a las mismas) son lo que ayuda a llenar los cuatro *templates* mencionados anteriormente. El concepto usuario de la metodología ágil correspondería al concepto rol en nuestro enfoque.

En segundo lugar y a partir de la elicitación de requisitos se procede a la especificación de requisitos completando esos cuatro *templates*. Los requisitos se han organizado en cuatro partes: *división de labor, control de acceso, vista (o interfaces de usuario) y controlador* (métodos o procedimiento que permiten el funcionamiento de la aplicación, gestionando la interacción del componente Modelo con el componente Vista y de los Usuarios con la aplicación). A continuación se indica el llenado de cada *template:*

- La Tabla 16 corresponde al *template* de la *división de labor* que muestra los conceptos de la ontología de la estructura organizacional de grupo que representan al componente Modelo, por tanto, se coloca la información correspondiente a este componente de acuerdo a los pasos 2 al 6 especificados en el algoritmo de la sección 6.4.
- La Tabla 17 específica el *template de control de acceso*, estableciendo la Fase en que participa un rol, si esta participación es individual o grupal. Para llenar esta tabla se ejecuta el paso 7 del algoritmo de la sección 6.4.
- La Tabla 18 se refiere al *template del componente vista* y se completa de acuerdo al paso 8 del algoritmo de la sección 6.4. En esta tabla se escribe el nombre de la interfaz de usuario y el tipo de Vista a presentar (Vista de Información, Vista de Participantes y/o Vista de Contexto). La TS y TP producen la Vista de Información, mientras la TPC y TTC generan la Vista de Participantes y Vista de Contexto. Tomando en consideración que las columnas referentes a las vistas dependen del dato que contengan las columnas TPC o TTC, por ello, cuando están contienen "SI", entonces a la columna de la Vista de Información se le colocará "NO", mientras a las Vistas de Participantes y Contexto "SI"; en caso contrario se invertirán los valores.

Tabla 16. **Tabla de especificación de la división de labor del Modelo.**

EOG	Políti_ca	Usua_rio	Rol	Esta_tus	D/O	Even_to	Ta_rea	Acti_vidad	Recur_so	TS	TP	TPC	TTC	Compo_nente
NOM_BRE DE LA EOG	NOM_BRE DE LA POLÍTICA DE LA EOG	U1	R1	2	DO-R1	E-T1	T1	A1-T1	RE1A1	SI	NO	NO	SI	CRE1
		U2						A2-T1	RE1A2					CRE1
						E-T6	T6	A1-T6	RE1A1	SI	NO	NO	NO	CRE1
		U3							RE2A1					CRE2
						E-T3	T3	A1-T3	RE1A1	NO	NO	NO	SI	CRE1
		U4	R2	3	DO-R2	E-T4	T4	A1-T4	RE1-A1	NO	SI	NO	NO	CRE1
		U1							RE2-A1					CRE2
		U2							RE3-A1					CRE3
		U7						A2-T4	RE1-A2					CRE1
		U8				E-T1	T1	A1-T1	RE1A1	NO	NO	SI	NO	CRE1
		U5						A2-T1	RE1A2					CRE1
		U1	R3	4	DO-R3	E-T1	T1	A1-T1	RE1A1	SI	NO	SI	NO	CRE1
		U5						A2-T1	RE1A2					CRE1
		U9				E-T2	T2	A1-T2	RE1A1	NO	NO	NO	SI	CRE1
		U6							RE2A1					CRE2
		U10	R4	1	DO-R4	E-T5	T5	A1-T5	RE1A1	NO	SI	NO	NO	CRE1
		U1				E-T2	T2	A1-T2	RE1A1	SI	NO	SI	NO	CRE1
		U7				E-T4	T4	A1-T4	RE1A1	NO	SI	NO	NO	CRE1
		U5				E-T6	T6	A1-T6	RE1A1	NO	NO	NO	SI	CRE1

Tabla 17. **Tabla de especificación de control de acceso del Modelo.**

ROL	PRECEDENCIA DE TAREA	TAREA	PRECEDENCIA DE FASE	FASE
R1	2	T1	3	F1
R1	1	T3		
R1, R4	3	T6		
R2	1	T1	4	F2
R3, R4	2	T2		
R3	1	T1	2	F3
R3, R4	4	T2		
R2, R4	3	T4		
R1	2	T1		
R3, R4	2	T2	1	F4
R1, R4	4	T6		
R1	3	T3		
R4	1	T5		

Tabla 18. **Tabla de especificación de la Vista.**

INTERFAZ DE USUARIO	VISTA DE INFORMACIÓN	VISTA DE PARTICIPANTES	VISTA DE CONTEXTO
Nombre de la Interfaz de Usuario	SI	NO	NO
Nombre de la Interfaz de Usuario	NO	SI	SI
Nombre de la Interfaz de Usuario	SI	NO	NO
Nombre de la Interfaz de Usuario	NO	SI	SI
Nombre de la Interfaz de Usuario	NO	SI	SI

- La tabla 19 que corresponde al *template del controlador* se establece conforme al paso 9 descrito en la sección 6.4. En esta tabla se debe indicar el nombre de la Sesión, que hace el mecanismo de notificación y concurrencia. La Notificación se realiza en todas las Tareas, mientras que la Concurrencia sólo se aplica para la TPC y TTC. De la siguiente forma: la columna de Notificación siempre se completa con "SI", mientras en la columna de Concurrencia se coloca "SI", cuando en la columna TPC o TTC (de la Tabla de Especificación de la División de Labor) tienen "SI", en caso contrario se coloca "NO".

Tabla 19. **Tabla de especificación del Controlador.**

SESIÓN	NOTIFICACIÓN	CONCURRENCIA
Nombre de la Sesión	Se realiza mediante …	Utiliza el mecanismo del *framework*.
Nombre de la Sesión	Se realiza mediante …	NO
Nombre de la Sesión	Se realiza mediante …	Utiliza un mecanismo que …
Nombre de la Sesión	Se realiza mediante …	Utiliza un mecanismo que …
Nombre de la Sesión	Se realiza mediante …	NO

6.5.2. Especificación de diseño

En esta etapa, se considera el diseño del modelo de datos y de la interfaz de usuario. Para ello, se deben analizar las cuatro Tablas que se completaron en la etapa anterior, de especificación de requisitos. En cuanto al modelo de datos, se revisan:

1. Los recursos que implican captura o procesamiento de información, de tal manera, que (casi) todos aquellos elementos relacionados con esto, deben ser parte del modelo de datos.
2. Los mecanismos de notificación y concurrencia, que modifican recursos (incluyendo los que se encuentran en las interfaces de usuario), ya que requiere que estos cambios sean almacenados y/o procesados. Considerar aquellos elementos relacionados con ello, como parte del modelo de datos.

Con respecto al diseño de interfaz de usuario, el desarrollador deberá centrarse en:

1. El *template del componente vista*; en particular, la interfaz de usuario y las vistas que esta contiene (Información, de Participantes y/o de Contexto). Establecen las interfaces de un usuario que se deben crear.
2. Los componentes del *template de la división de labor*, ya que indican los elementos que constituyen la interfaz de usuario. De esta forma, se puede hacer un bosquejo de cada interfaz.
3. El mecanismo de notificación, para determinar los elementos que se van a integrar en la interfaz de usuario para que los demás usuarios conozcan lo que está pasando en el entorno compartido. Así, se tendrían incorporadas en el diseño la interfaz de usuario características correspondientes a la **conciencia de grupo**.
4. El mecanismo de concurrencia, para establecer los elementos que servirán para generar, controlar y gestionar el historial (o **memoria**) **de grupo**. Estos elementos deben ser incorporados al diseño para especificar de manera completa los aspectos de grupo.

6.5.3. Especificación de la implementación

A partir de la especificación del diseño, es posible determinar qué se debería implementar para el desarrollo de *groupware.* Para ello, se recomienda utilizar MVC:

- **Modelo.** Crear la base de datos relacional u orientada a objetos para procesar y gestionar la información del *groupware*.
- **Vista.** Construir las interfaces de usuario con las vistas a mostrar, ya sea de Información, de Participantes y/o de Contexto; así como agregar los *widgets* o elementos que sean necesarios para suministrar la **conciencia y la memoria de grupo**.
- **Controlador.** Codificar los métodos de cada Tarea mediante servicios web (de preferencia), agregando el mecanismo de notificación y/o concurrencia que se requiera. Estos métodos deben permitir modificar el modelo de datos por los resultados de la interacción con los recursos compartidos o viceversa.

Con respecto a la implementación se recomienda utilizar un *framework* basado en MVC para el *front-end* como Codeigniter, Larabel, Backbone, Angularjs, Ruby and Rails, Django, etc. Mientras para el *back-end* Bootstrap, Materialize, Foundation, etc.

Con referencia a las pruebas, que permiten verificar el funcionamiento de una aplicación de acuerdo a diversos aspectos, tales como: usabilidad, correcto funcionamiento, desempeño, seguridad, responsividad,

etc., este trabajo recomienda realizar pruebas de usabilidad, robustez y desempeño, debido a que la metodología establece las pautas para diseñar e implementar las interfaces de usuario, dirigidas a la interacción entre los usuarios y la propia a aplicación.

6.6. Modelos derivados del enfoque metodológico

A partir del enfoque metodológico semántico descrito en la sección anterior, se han derivado dos modelos, esenciales en el dominio de CSCW: Modelo de Control de Acceso y Modelo de Conciencia y Memoria de Grupo.

6.6.1. Modelo de control de acceso

El control de acceso especifica los mecanismos usados para aplicar una directiva de seguridad que regula las acciones que un usuario puede realizar; previniendo la difusión de información sensible y acciones sobre el sistema, que pueda poner en peligro la seguridad tanto de los individuos como de las empresas [Cherif 12].

El Modelo de Control de Acceso controla el acceso del usuario al espacio de trabajo compartido y a los recursos compartidos; regulando la interacción entre los usuarios, con el fin de evitar conflictos derivados de actividades cooperativas y competitivas.Para implementar este modelo de forma efectiva y apropiada en las aplicaciones colaborativas, deben tenerse en cuenta los siguientes requisitos [Tolone 05, Cherif 12], de tal manera, que el control de acceso debe:

- Ser capaz de proteger cualquier tipo de información y/o recursos en diferentes niveles de granularidad.
- Facilitar el acceso transparente de los usuarios autorizados y la exclusión rigurosa de los usuarios no autorizados de una manera flexible que no restrinja el trabajo en grupo.
- Ser suficientemente expresivo para permitir una especificación de alto nivel de derechos de acceso, mejorando la gestión en el aumento de la complejidad que introduce el groupware.
- Ser dinámico, es decir, debería ser posible especificar y cambiar las políticas en tiempo de ejecución.
- Apoyar la delegación, revocación y gestión de políticas en tiempo de ejecución.
- Conceder control de acceso considerando el contexto actual del usuario.

Existen varios modelos de control de acceso para entornos colaborativos, tal como:

- El modelo de matriz de acceso [Lampson 74], representa la política de autenticación del usuario, presentando varias implementaciones como listas de control de acceso (ACL, *Access Control Lists*) y listas de capacidades (CL, *Capability Lists*). Se utiliza para describir qué usuarios tienen acceso a qué objetos. Este modelo proporciona el marco básico para describir los sistemas de protección [Shen 92]. Sin embargo, no especifica la naturaleza exacta de los sujetos, objetos o derechos de acceso.
- El modelo de control de acceso basado en roles (RBAC, *Role-based Access Control*) [Sandhu 96], donde una política basada en roles es una política que regula el control de acceso de usuarios a recursos u objetos en el sistema de acuerdo con las actividades y responsabilidades organizacionales de cada usuario en el sistema. RBAC es adecuado para las actividades empresariales, ya que naturalmente mapea la estructura de una organización [Samarati 00].

En el entorno colaborativo se tiene que gestionar el cambio frecuente de los derechos de acceso por parte de los usuarios. También se debe considerar que varios usuarios con diferentes niveles de privilegios intentarán acceder a recursos compartidos [Tolone 05].

Los enfoques basados en RBAC superan algunos problemas del cambio dinámico de derechos gracias a la noción de sesión [Jaeger 96]. Sin embargo, el concepto de sesión, también, impide una reasignación dinámica de roles, ya que los roles no se pueden cambiar dentro de una sola sesión. Para resolver este

problema, se propuso el Control de Acceso Espacial (SAC, *Spatial Access Control*) [Bullock 99], donde en lugar de dividir a los usuarios en grupos como en RBAC, SAC divide el entorno colaborativo en espacios abstractos. Esta solución requiere el bloqueo de estructuras de datos, ya que cada acceso necesita verificar las estructuras de datos subyacentes, lo que reduce el rendimiento de este entorno.

En este trabajo, se propone un modelo de control de acceso (véase la Figura 37) [Anzures 09b, Anzures 10a, Anzures 10b] que parte de la idea de RBAC, donde los permisos se asignan a los roles en lugar de a los usuarios. De esta forma, la política no debe cambiarse cuando los usuarios modifican su rol dentro de la organización. En dicho modelo se incorpora el concepto de Fase y los tipos de Tarea (TS, TP, TPC y TTC); de tal forma que, gracias al establecimiento de las Fases, los usuarios se dividen en grupos pequeños, ya que sólo algunos roles participan en un momento de colaboración. El tipo de tarea permite establecer si sólo habrá un rol interactuando con la aplicación (TS), dos o más roles pero modificando diferentes recursos (TP), dos o más roles actualizando el mismo recurso pero en diferente tiempo (TPC) y dos o más roles utilizando el mismo recurso al mismo tiempo. Lo cual simplifica el control de acceso a los recursos compartidos y obviamente a la propia aplicación.

Cada Fase controla los roles que pueden participar en ella y el tipo de Tarea determina cómo será la interacción; lo que facilita la autenticación, comunicación, colaboración y coordinación entre los usuarios en el espacio de trabajo compartido. Además, al estar basado este modelo en la ontología *workflow* (véase la sección 6.3), es posible añadir, modificar o actualizar un rol, tarea, actividad, recurso, fase u orden en que se ejecutarán de forma sencilla. En las Tablas de Especificación (sección 6.5) bastaría con agregar una fila o renglón correspondiente al cambio que se va a llevar a cabo.

Ninguno de los modelos de control de acceso existentes para aplicaciones colaborativas que hemos estudiado tiene en cuenta este concepto de Fase y de Tipo de Tarea, que facilitan el proceso de control de acceso en el espacio de trabajo compartido en g*roupware*.

El Modelo de control de acceso propuesto en este trabajo (véase la Figura 37) es formal y explícito, permitiendo establecer:

- ¿Quién accede y qué función desempeña en la aplicación?.
- ¿Qué hace?.
- ¿Qué usa para hacerlo?.
- ¿Cuándo lo hace? o ¿En qué momento?.
- ¿Dónde lo hace?.
- ¿Cómo interactúa?.

De esta manera, se controla el acceso a la aplicación y a los recursos compartidos formal y explícitamente.

6.6.2. Modelo de conciencia y memoria de grupo

Dos de los aspectos más importantes de las aplicaciones colaborativas son los conceptos de conciencia de grupo y memoria de grupo. El primero se refiere tanto a la percepción y conocimiento del grupo como a sus actividades. El segundo representa el historial de los recursos compartidos, permitiendo establecer un contexto sobre las actividades colaborativas desarrolladas.

En el dominio CSCW, se han realizado varios intentos para formalizar los conceptos relacionados con la conciencia de grupo [Arruda 03]; proponiendo teorías, marcos y taxonomías para considerar estos conceptos en el desarrollo de aplicaciones colaborativas. A continuación, describimos los más representativos:

- La Teoría de la Conciencia de Grupo [Gutwin 02] incluye un *framework* que define diferentes elementos de conciencia y hace posible la validación del soporte de conciencia mediante un conjunto de preguntas relevantes. Una de las principales contribuciones de este trabajo ha sido identificar elementos de conocimiento que conforman el núcleo de la conciencia de grupo del espacio de trabajo, cada uno relacionado con la respuesta para proporcionar ese elemento de conocimiento (véase la Tabla 20). Además, se definen seis elementos adicionales de conocimiento referentes a acciones pasadas. Estos elementos constituyen el historial de acción relacionado con la historia de la pregunta y el artefacto, historial de eventos, historial de presencia, historial de

ubicaciones e historial de acciones relacionados con la pregunta ¿Qué? Estos elementos proporcionan información adicional a la que se refleja en las diez dimensiones mencionadas y pueden considerarse en herramientas que tratan con la memoria de grupo.

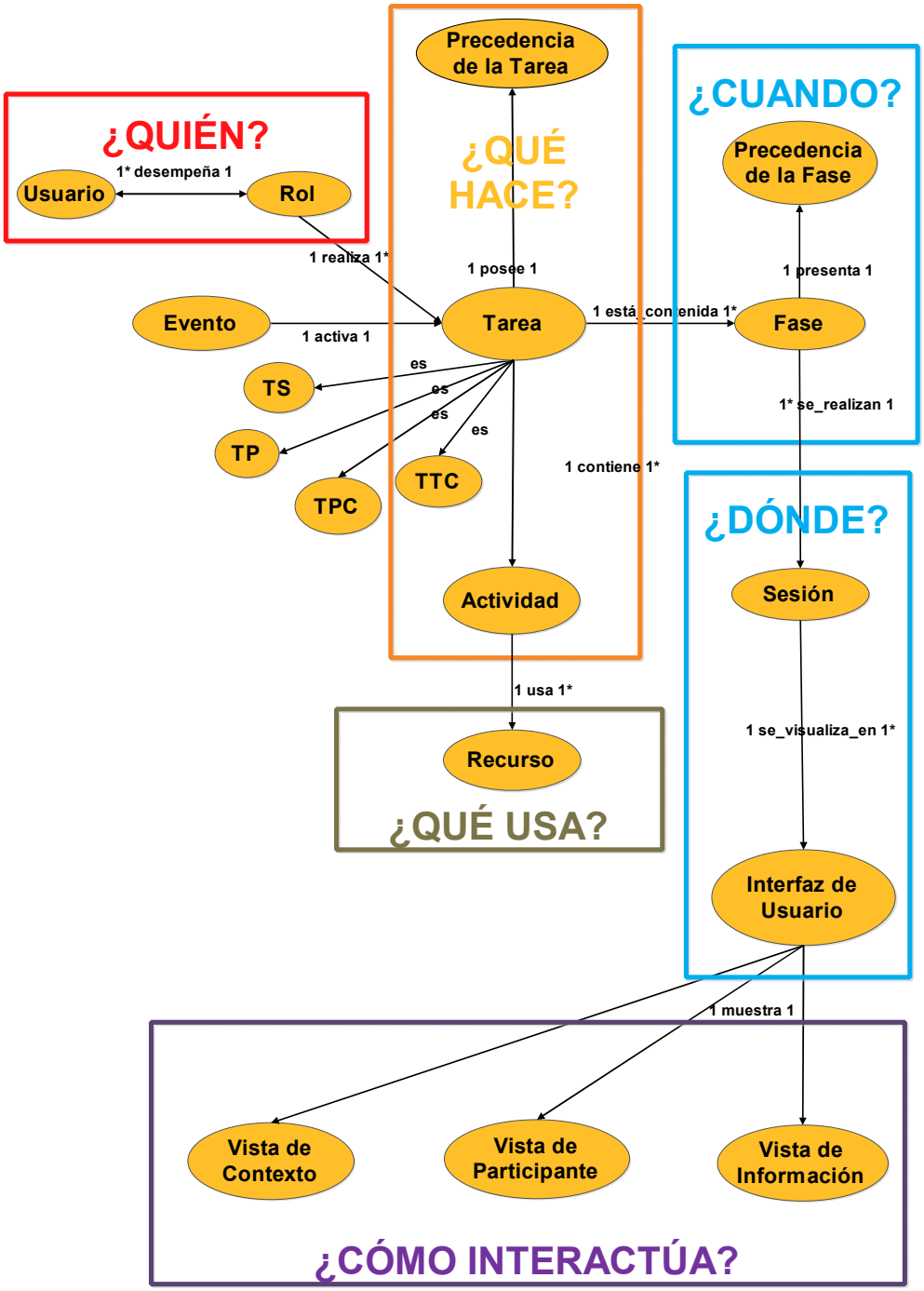


Figura 37. Modelo de Control de Acceso.

- En [Gutwin 96] se describen cuatro tipos de conciencia: la conciencia informal (quién está alrededor y qué está haciendo); la conciencia social (información que una persona tiene de los demás en un contexto social o conversacional); la conciencia estructural del grupo (conocimiento de los roles y responsabilidades de las personas, su estado, etc.); y el conocimiento del espacio de trabajo (interacciones de otras personas con el espacio de trabajo y sus artefactos).
- La extensión de Teoría de la Conciencia de Grupo [Tam 06], que incluye aspectos históricos y agrega una nueva clasificación de los elementos de conciencia. Esta clasificación se organiza en tres vistas: vista basada en artefactos (cambios en los artefactos presentes en el espacio de trabajo),

vista basada en persona (cambios en las personas que están trabajando) y vista del área de trabajo (cambios en el espacio de trabajo completo). Esta propuesta está orientada a la percepción de cambios en documentos o espacios de trabajo de una manera asíncrona.

Tabla 20. **Tabla de elementos de la Teoría de la Conciencia de Grupo [Gutwin 02].**

CATEGORÍA	ELEMENTOS DE LA CONCIENCIA DE GRUPO	PREGUNTAS ESPECÍFICAS
¿Quién?	Presencia	¿Hay alguien en el espacio de trabajo?
	Identidad	¿Quién está participando? ¿Quién es ese?
	Autoría	¿Quién está haciendo eso?
¿Qué?	Acción	¿Qué están haciendo?
	Intención	¿Cuál es el objetivo de esa acción?
	Artefacto	¿En qué objeto están trabajando?
¿Dónde?	Ubicación	¿Dónde están trabajando?
	Mirada	¿Dónde están mirando?
	Vista	¿Dónde pueden ver?
	Alcance	¿Dónde pueden llegar?

- GAKS (*Group Awareness Knowledge-based System*) [Decouchant 09] es un sistema fundamentado en conocimiento, basado en reglas utilizadas para mejorar el soporte de la conciencia de grupo en una herramienta para la coautoría de documentos web. Por lo tanto, este sistema proporciona una herramienta de comunicación contextual síncrona/asincrónica y un detector de proximidad de trabajo físico/lógico.
- Sistema de apoyo a la toma de decisiones basado en el contexto [Feng 09], incluye un motor basado en reglas para proporcionar apoyo a la conciencia de grupo.
- Enfoque formal basado en roles [Zhu 09] que permite utilizar un motor de inferencia para proporcionar soporte a la conciencia de grupo, utilizando grafos de roles o lógica.
- La ontología para la representación del contexto en sistemas colaborativos [Vieira 05], que es utilizada por un mecanismo de razonamiento basado en lógica para recomendar herramientas basadas en el contexto actual de los miembros del grupo.
- El enfoque de conceptualización ontológica para la conciencia en sistemas de modelado colaborativo [Gallardo 11] conceptualiza algunos de los aspectos más importantes de la conciencia de grupo; identificando diferentes puntos de vista que agrupan todos los conceptos relacionados con los diferentes aspectos y perspectivas a considerar en el diseño y desarrollo basado en modelos.

El modelo de conciencia y memoria de grupo (véase Figura 38) propuesto en este trabajo de tesis, parte de los trabajos de [Gutwin 96] y [Tam 06], además retoma algunas ideas de [GAKS 09] y [Gallardo 11], adecuándolas al enfoque metodológico semántico. Por tanto, el resultado es un modelo formal basado en conocimiento que suministra:

- La conciencia informal por medio de los conceptos: usuario, rol, tarea, TS, TP, TPC y TCP.
- La conciencia social mediante los conceptos: usuario, rol, tarea, TPC, TCP, fase, sesión, interfaz de usuario, vista de participantes y vista de contexto.
- La conciencia estructural del grupo a través de los conceptos: usuario, rol, estatus, D/O, tarea, evento, precedencia de tarea, actividad y recurso.
- El conocimiento del espacio de trabajo con los conceptos: usuario, rol, tarea, TS, TP, TPC, TCP, fase, sesión, interfaz de usuario, vista de información, vista de participantes y vista de contexto.
- La vista basada en artefactos llevando un historial de los recursos compartidos utilizados.
- La vista basada en persona registrando todos los cambios en los usuarios, roles que ha desempeñado, tareas que realizado, recursos utilizados.
- La vista de área de trabajo teniendo un historial de modificaciones sobre la interfaz de usuario, vista de información, vista de participantes y vista de contexto.

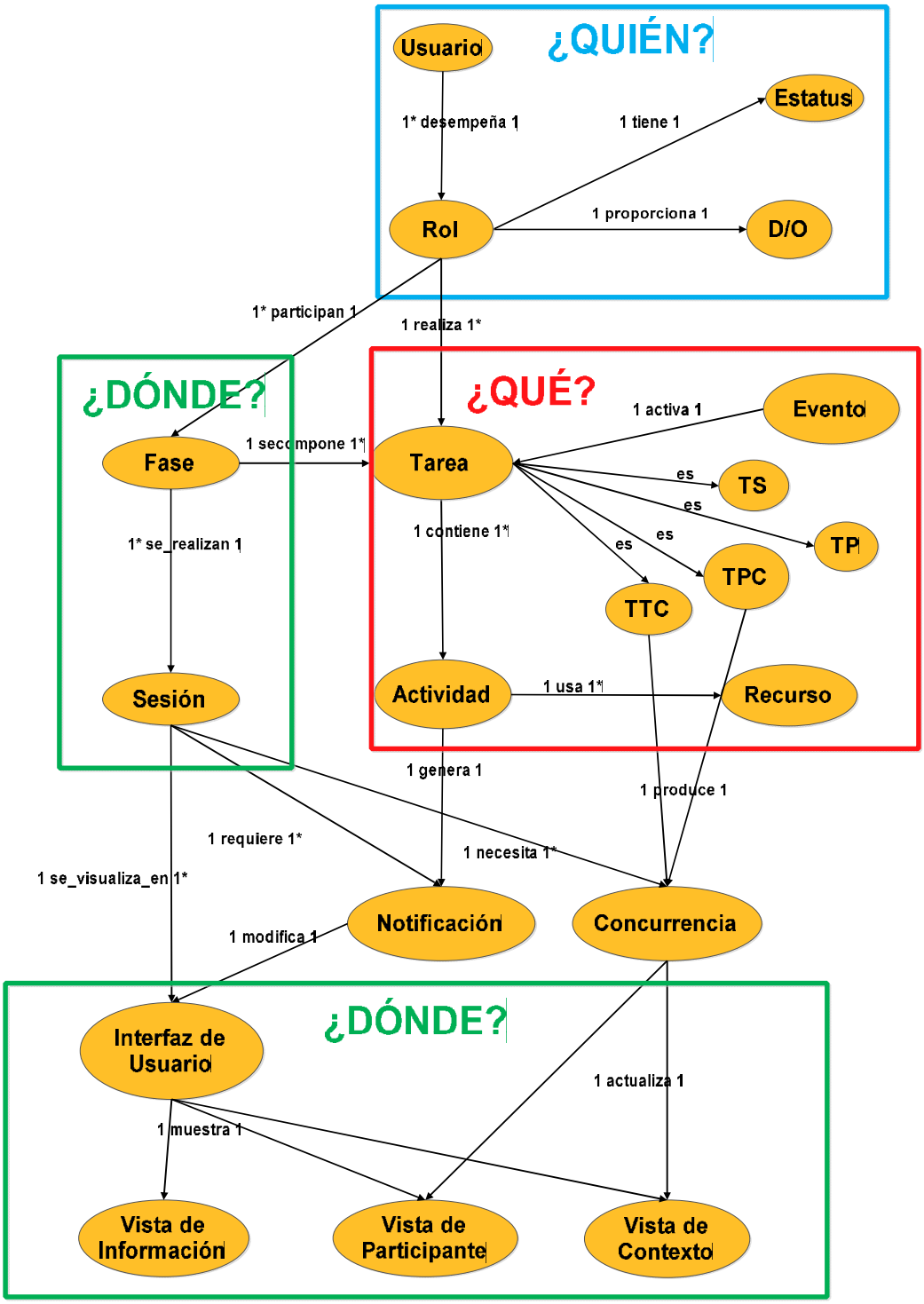


Figura 38. Modelo de conciencia y memoria de grupo.

El modelo propuesto, además, sustenta el núcleo de la conciencia de grupo como se puede apreciar en la Tabla 21.

- Los conceptos de usuario, rol, estatus, D/O y tarea fundamentan a la categoría ¿QUIÉN?.
- Los conceptos de tarea, actividad y recurso soportan a la categoría ¿QUÉ?.
- Los conceptos de fase, sesión, interfaz de usuario, vista de información, vista de participantes y vista de contexto establecen la categoría ¿DÓNDE?.

Tabla 21. **Tabla de elementos de la Teoría de la Conciencia asociados a nuestro modelo.**

CATEGO _RÍA	ELEMENTOS DE LA CONCIENCIA DE GRUPO	PREGUNTAS ESPECÍFICAS	CONCEPTO ASOCIADO
¿Quién?	Presencia	¿Hay alguien en el espacio de trabajo?	Usuario, Rol
	Identidad	¿Quién está participando? ¿Quién es ese?	Rol, Usuario, Estatus, D/O
	Autoría	¿Quién está haciendo eso?	Usuario, Tarea
¿Qué?	Acción	¿Qué están haciendo?	Actividad
	Intención	¿Cuál es el objetivo de esa acción?	Tarea
	Artefacto	¿En qué objeto están trabajando?	Recurso
¿Dónde?	Ubicación	¿Dónde están trabajando?	Fase, Sesión
	Mirada	¿Dónde están mirando?	Interfaz de usuario
	Vista	¿Dónde pueden ver?	Vista de Información, Participantes y Contexto
	Alcance	¿Dónde pueden llegar?	

6.7. Conclusiones del enfoque metodológico semántico

En este capítulo, se ha presentado el enfoque metodológico semántico que se sustenta en el estilo arquitectónico MVC personalizado para *groupware*; describiendo cada uno de los elementos que forman parte de estos tres componentes.

En este capítulo, se ha presentado el enfoque metodológico semántico que se sustenta en el estilo arquitectónico MVC personalizado para *groupware*; describiendo cada uno de los elementos que forman parte de estos tres componentes.

En el componente Modelo se ha explicado y detallado la ontología de la política de manejo de sesión, que modela de manera explícita y formal la estructura organizacional de grupo y permite adaptar dicha estructura a las necesidades cambiantes y a varios estilos organizacionales del grupo. Así como también, se ha expuesto el manejo de sistemas basados en políticas, parte fundamental de la ontología de la política de manejo de sesión.

Respecto al componente Vista, se han presentado los elementos que lo constituyen, destacando el establecimiento de tres vistas (Información, Participantes y Contexto) tomadas del Modelo Conceptual [Ellis 99] y adaptadas para simplificar y controlar la gestión de la interacción de los usuarios en la aplicación. Además, los recursos que se especifican son los elementos propios de cada interfaz de usuario (caja de texto, botón, formulario, *widgets*, etc.) y los elementos que se comparten (documentos, publicaciones, mensajes, etc.), de tal manera que se simplifica tanto el diseño como la implementación de las interfaces de usuario. Por otra parte, se ha determinado que las tareas parcialmente y totalmente concurrente deben desplegar la vista de participantes (conciencia de grupo) y contexto (memoria de grupo).

En el componente Controlador se han detallado los elementos que la constituyen y que permiten gestionar la comunicación, colaboración y coordinación entre los usuarios. Se ha señalado que el mecanismo de notificación se debe utilizar en cualquier tarea que se vaya a ejecutar; mientras el mecanismo de concurrencia es necesario cuando se llevan a cabo las tareas paralelas, parcialmente y totalmente concurrente.

También se ha planteado la ontología *workflow*, que es una extensión de la ontología de la política de manejo de sesión y que se deriva del estilo arquitectónico MVC. La ontología *workflow* permite establecer el enfoque metodológico semántico, ya que define el conjunto de pasos y la manera en que estos se llevan a cabo en el desarrollo de groupware. Después se han explicado las tres etapas del enfoque metodológico y se han propuesto cuatro tablas de especificación para simplificar cada una de las etapas de dicho enfoque. Finalmente, se han descrito dos modelos que son resultado del enfoque metodológico semántico: *modelo de control de acceso y modelo de conciencia y memoria de grupo*, explicando su importancia en el dominio CSCW.

7. Metodología de evaluación del enfoque propuesto

En este capítulo, se presenta una metodología de evaluación del enfoque metodológico semántico basado en un modelo arquitectónico para el desarrollo de *groupware*. Por tanto, en primer lugar se describen los elementos principales (SERVQUAL y Modelo lingüístico difuso) que la sustentan; en segundo lugar se proponen las dimensiones y el cuestionario para esta metodología, así como la aplicación de la misma; finalmente se presentan los resultados y recomendaciones obtenidas después de aplicar la metodología de evaluación.

7.1. Descripción de la metodología

Los enfoques metodológicos resultan útiles para conceptualizar y organizar las principales actividades relacionadas con el desarrollo de software (análisis, diseño, programación, pruebas, puesta en marcha, y mantenimiento) [Hernández 16]. De esta manera, la probabilidad de éxito del proyecto aumenta y su coste disminuye. Si bien es cierto que el uso de un enfoque metodológico no es una garantía de solución a los problemas suscitados en el desarrollo de software, sí proporciona los recursos con los cuales los desarrolladores pueden gestionar su trabajo de una manera adecuada, permitiendo la facilidad de entenderlo, implementarlo, adaptarlo, etc. Así que, la evaluación del enfoque metodológico le daría mayor certeza a los desarrolladores en el proceso de crear software.

Por tanto, se realizará la evaluación de la usabilidad del enfoque metodológico semántico propuesto en esta tesis; adaptada del trabajo [Sánchez-Gálvez 15]. Aunque este trabajo se centra en la usabilidad de la interfaz de este tipo de bibliotecas, se debe tomar en cuenta que la usabilidad no sólo se considera como la capacidad del producto de software para ser entendido, aprendido y usado por el usuario, a la vez que atractivo para el mismo, cuando se usa bajo condiciones específicas [Nielsen 93]. Sino que también se puede aplicar a los métodos para mejorar la facilidad de uso de las aplicaciones durante su proceso de diseño. Por consiguiente, los diseñadores y desarrolladores de software también se deben centrar en cuestiones de usabilidad.

En este sentido de medir la usabilidad se requiere el establecimiento de dimensiones, como una manera de medir la usabilidad de acuerdo a la ISO (*International Standard Organization*) 9241-11 [ISO 98] y Nielsen [Nielsen 93]. [Sánchez-Gálvez 15] considera cuatro dimensiones: *Eficacia, Eficiencia, Satisfacción y Facilidad de aprendizaje*; representadas en un cuestionario que contiene veinte ítems (preguntas) de medición para capturar las percepciones que tienen los usuarios sobre el grado de usabilidad. Además, ese trabajo para evaluar dicho cuestionario utiliza en el modelo lingüístico difuso (que usa operadores de agregación de información lingüística —LOWA y LWA— para operar con palabras directamente, que se corresponden con las valoraciones u opiniones expresadas por los usuarios) [Heradioa 13] y la metodología SERVQUAL [Parasuraman 88] (un instrumento basado en la teoría de la brecha de la calidad del servicio, que fue utilizado para evaluar las instituciones del sector privado).

Tomando como base estas ideas, el enfoque metodológico propuesto en el capítulo 6 se evaluará utilizando un cuestionario de 20 preguntas mediante cuatro dimensiones (Eficacia, Eficiencia, Satisfacción y Facilidad de aprendizaje), la brecha de calidad de servicio y los operadores de agregación lingüísticos (LOWA y LWA).

7.1.1. SERVQUAL

SERVQUAL es la medida de calidad de servicios más aceptada y extendida, se basa en el principio de que son los clientes los que deben juzgar la calidad de los servicios, los demás juicios son irrelevantes [Zeithaml 90]. Por tanto, la satisfacción del cliente será el elemento más importante a tener en cuenta en la valoración de la calidad percibida, de modo que en SERVQUAL la calidad del servicio está directamente relacionada con minimizar la distancia entre las expectativas del cliente con respecto al servicio en cuestión y la percepción que éste tiene de dicho servicio tras su utilización. Consecuentemente, en SERVQUAL, un cliente valorará negativamente o positivamente la calidad de un servicio cuando las percepciones que ha obtenido sean inferiores o superiores a las expectativas que tenía, respectivamente. Por ello, cuando se prestan servicios, donde uno de los objetivos es la diferenciación mediante un servicio de calidad, se debe poner especial énfasis en superar las expectativas de los clientes o usuarios.

En el trabajo que aquí se presenta, se busca evaluar la calidad del servicio considerando tres dimensiones: el valor afectivo del servicio ofrecido, el valor del enfoque metodológico, y el valor del control de la información. Por tanto, SERVQUAL evalúa la calidad del servicio a partir de: nivel de servicio mínimo exigible, nivel de servicio esperado, y nivel observado (percepción del usuario). En base a las respuestas de los usuarios, definen dos variables para detectar los puntos fuertes y débiles del servicio: *Adecuación del Servicio* (valor percibido − valor mínimo) que indica las áreas donde el servicio evaluado está por debajo del nivel esperado por el usuario, y *Excelencia del Servicio* (valor percibido – valor esperado) que identifica las áreas donde se da un mejor servicio que el esperado por el usuario.

7.1.2. Modelo lingüístico difuso

La información no siempre puede ser valorada de una forma cuantitativa, a veces es necesario hacerlo de una forma cualitativa. La existencia de variables cualitativas, inherentes al comportamiento humano, o de elementos del ambiente externo de difícil cuantificación objetiva, hace que a los individuos les resulte más adecuado expresar sus opiniones sobre un servicio recibido por medio de términos lingüísticos en lugar de utilizar valores numéricos exactos.

Una variable lingüística se diferencia de una numérica en que sus valores no son números, sino palabras u oraciones del lenguaje natural, o de un lenguaje artificial [Zadeh 75]. El uso de la Teoría de Conjuntos Difusos ha dado buenos resultados para modelar información cualitativa [Zadeh 65] y ha sido aplicada con éxito en diversas situaciones [Herrera 09, Cid-López 15, Cid-López 16, Cid-López 17].

Cuando se emplea un modelo lingüístico, se asume la existencia de un conjunto apropiado de términos o etiquetas, de acuerdo con el dominio del problema, con el que los individuos pueden expresar sus percepciones.

El modelo lingüístico difuso ordinal [Herrera 96] que se definirá formalmente a continuación es el que vamos a utilizar para evaluar nuestro enfoque metodológico.

Definición 7.1: Modelo lingüístico difuso ordinal [Herrera 96]**.**

Sea $S = \{s_i | i = 0,1, \dots, \tau\}$ un conjunto finito discreto, totalmente ordenado de términos lingüísticos, con cardinalidad impar (τ+1), el cual tiene definido las siguientes operaciones básicas:

Operación de negación:

- $Neg(s_i) = s_j \mid j = \tau - 1$

Operador de comparación:

- Operador máximo: $MAX(s_i, s_j) = s_i$, si $s_i \geq s_j$
- Operador mínimo: $MIN(s_i, s_j) = s_i$, si $s_i \leq s_j$
- Operador de distancia entre etiquetas lingüísticas:

$$D(s_i, s_j) = s_k, \text{ donde } k = \begin{cases} i - j, & \text{si } i > j \\ j - i, & \text{en caso contrario} \end{cases}$$

Los operadores de agregación que vamos a utilizar para agregar las valoraciones expresadas por el conjunto de todos los usuarios que han participado en la evaluación de nuestro enfoque metodológico

semántico son **LOWA** (*Linguistic Ordered Weighted Averaging*) y **LWA** (*Linguistic Weighted Averaging*), ambos basados en el operador **OWA (***Ordered Weighted Averaging***)**.

LOWA [Herrera 96]: En este operador toda la información lingüística a agregar tiene la misma importancia.

Definición 7.2. LOWA [Herrera 96]**.**

Sea $A = \{a_1, \dots, a_m\}$ un conjunto de etiquetas a ser agregadas, el operador LOWA ϕ está definido como:

$\phi(a_1, \dots, a_m) = W \cdot B^T = C^m\{w_k, b_k, k = 1, \dots, m\} = w_1 \odot b_1 \oplus (1 - w_1) \odot C^{m-1}\{\beta_h, b_h, h = 2, \dots, m\}$

Donde $W = [w_1, \dots w_m]$ es un vector de peso, tal que $w_i \in [0,1]$ y $\sum_i w_i = 1$, $\beta_h = w_h / \sum_2^m w_k$ y $B = [b_1, \dots, b_m]$ es un vector asociado a A, tal que $B = \sigma(A) = [a_{\sigma(1)}, \dots, a_{\sigma(m)}]$, con $a_{\sigma(j)} \leq a_{(i)}, \forall\, i \leq j$, siendo σ una permutación sobre el conjunto de etiquetas A, y C^m es el operador de combinación convexa de m etiquetas. Si $m = 2$, entonces está definido como:

$C^2\{w_i, b_i, i = 1,2\} = w_1 \odot s_j \oplus (1 - w_1) \odot s_i = s_k$,

tal que $k = min\{\tau, i + round\,(w_i \cdot (j - i))\}$, $s_j, s_i \in S$, $(j \geq i)$, donde round es el operador redondeo y $b_1 = s_j$, $b_2 = s_i$.

Si $w_j = 1$ y $w_i = 0$, con $i \neq j, \forall i$, entonces la combinación convexa está definida como:

$C^m\{w_i, b_i, i = 1, \dots, m\} = b_j$.

Un importante parámetro del operador LOWA es el vector de peso W, el cual determina su comportamiento, de tal modo que:

$\phi(a_1, \dots, a_m) = MAX(a_i)$, si $W^* = [1,0, \dots, 0]$;

$\phi(a_1, \dots, a_m) = MIN(a_i)$, si $W_* = [0,0, \dots, 1]$;

$\phi(a_1, \dots, a_m) = Average(a_i)$, si $W_A = \left[\frac{1}{m}, \frac{1}{m}, \dots, \frac{1}{m}\right]$.

Con el fin de clasificar el operador de agregación LOWA, se considera su localización entre los operadores lógicos *or* y *and*. [Yager 88] introduce la medida de *orness* asociada con cualquier vector W

$$orness(W) = \frac{1}{m-1} \sum_{i=1}^{m} (m - i) w_i$$

Note que $orness(W^*) = 1$, $orness(W_*) = 0$ y $orness(W_A) = 0.5$. Si la orness se acerca a 1, el operador está próximo al operador *or*, mientras que si esta medida se aproxima a 0 se dice que está más cercano a un operador *and*. De esta manera, un $orness(W) \geq 0.5$ corresponde a un operador *orlike*, mientras que si $orness(W) < 0.5$, se trata de un operador *andlike*.

Una manera para calcular los pesos es mediante los cuantificadores lingüísticos difusos Q [Herrera 96], dados por la siguiente formula:

$$w_i = Q\left(\frac{i}{m}\right) - Q\left(\frac{i-1}{m}\right),\ i = 1, \dots, m$$

El cuantificador lingüístico difuso Q está dado por:

$$Q(r) = \begin{cases} 0, & \text{si } r < a \\ \dfrac{r - a}{b - a}, & \text{si } a \leq r \leq b \\ 1, & \text{si } r > b \end{cases}$$

Se ha asignado nombres concretos a ciertos conjuntos de valores que pueden tomar los cuantificadores lingüísticos difusos proporcionales no decrementales para a y b, como son: "*most*" (0.3,0.8), "*at least half*" (0.0,0.5) y "*as many as possible*" (0.5,1.0).

Cuando los pesos se calculan a través de un cuantificador lingüístico en el operador de agregación LOWA, se denota como ϕ_Q.

LWA. En este operador, la información lingüística a agregar tiene diferente importancia, es decir, debe ser ponderada según su importancia.

Definición 7.3. LWA

Sea $A = \{(c_1, a_1), \dots, (c_m, a_m)\}$, $c_i, a_i \in S$, donde a_i representa la opinión ponderada y c_i el grado de importancia de a_i, entonces el operador Φ, está definido como:

$$\Phi[(c_1, a_1), \dots, (c_m, a_m)] = \phi\big(h(c_1, a_1), \dots, h(c_m, a_m)\big),$$

donde h es una función de transformación definida dependiendo del vector de peso W usado para el operador LOWA ϕ, tal que $h = LC_v^{\rightarrow}$, si $orness(W) \geq 0.5$, y $h = LI_v^{\rightarrow}$, si $orness(W) < 0.5$.

Las funciones $h = LC_v^{\rightarrow}$ son el siguiente grupo de funciones de conjunción lingüística, las cuales son monotónicamente no decrecientes en los pesos [Herrera 96]:

- Operador clásico MIN:

$$LC_1^{\rightarrow}(c, a) = MIN\ (c, a)$$

- Operador nilpotente MIN:

$$LC_2^{\rightarrow}(c, a) = \begin{cases} MIN(c, a), & \text{si } c > Neg(a) \\ s_0, & \text{en caso contrario} \end{cases}$$

- Conjunción débil:

$$LC_3^{\rightarrow}(c, a) = \begin{cases} MIN(c, a), & \text{si } MAX(c, a) = s_\tau \\ s_0, & \text{en caso contrario} \end{cases}$$

Las funciones $h = LI_v^{\rightarrow}$ son el siguiente grupo de funciones de inferencia lingüística, las cuales son monotónicamente no crecientes en los pesos [Herrera 96]:

- Función de inferencia de Kleene-Dienes:

$$LI_1^{\rightarrow}(c, a) = MAX\ (Neg(c), a)$$

- Función de inferencia de Gödel:

$$LI_2^{\rightarrow}(c, a) = \begin{cases} s_\tau, & \text{si } c \leq a \\ a, & \text{en caso contrario} \end{cases}$$

- Función de inferencia de Fodor:

$$LC_3^{\rightarrow}(c, a) = \begin{cases} s_\tau, & \text{si } c \leq a \\ MAX\ (Neg(c), a), & \text{en caso contrario} \end{cases}$$

7.2. Metodología de evaluación

La metodología de evaluación del enfoque metodológico semántico propuesto en esta tesis consiste en la aplicación de una serie pasos, a partir de los cuales se puedan proporcionar una serie de recomendaciones para mejorar el mencionado enfoque. Los pasos que se consideran en esta propuesta metodológica son:

1. Identificación de las dimensiones.
2. Elaboración del cuestionario.
3. Selección de las etiquetas lingüísticas a emplear.
4. Evaluación global del enfoque metodológico semántico mediante los operadores de agregación.
5. Presentación de los resultados de la evaluación.
6. Recomendaciones para mejorar el enfoque metodológico semántico.

7.2.1. Dimensiones de la evaluación del enfoque metodológico

La usabilidad es un concepto multidimensional. En este trabajo se proponen cuatro dimensiones de usabilidad: *eficacia, eficiencia, satisfacción y facilidad de aprendizaje*, como una manera de medir la facilidad de uso del enfoque metodológico semántico

La usabilidad es un concepto multidimensional. En este trabajo se proponen cuatro dimensiones de usabilidad: *eficacia, eficiencia, satisfacción y facilidad de aprendizaje*, como una manera de medir la utilidad y la facilidad de uso del enfoque metodológico semántico.

Para establecer las dimensiones a considerar para la evaluación de nuestro trabajo, se revisaron los modelos de usabilidad de [Nielsen 94] y el estándar ISO 9241-11 [ISO 98]. El primero, que es uno de los más citados en el área de ingeniería de la usabilidad, postula cinco atributos: facilidad de aprendizaje, eficiencia, memorización, baja tasa de error (fácil detección de errores), y satisfacción subjetiva.

De acuerdo con el estándar ISO 9241-11 [ISO 98], la usabilidad se define como: *"el grado en que un producto puede ser utilizado por los usuarios para lograr objetivos específicos con eficacia, eficiencia y*

satisfacción en un contexto de uso específico". Como indica la definición, tres dimensiones se utilizan para la usabilidad en este estándar: eficacia, eficiencia y satisfacción.

La definición para cada una de estas dimensiones es la siguiente:

- **Eficacia** se refiere a que vale para aquello para lo que está pensado, de modo que sirve para completar tareas en las que los usuarios logran objetivos específicos.
- **Eficiencia** se refiere al tiempo que se emplea y a los recursos utilizados en la realización de una determinada tarea.
- **Satisfacción** revela las actitudes positivas hacia el uso del enfoque [ISO 98]. Por tanto, para esta dimensión se considerará la opinión del usuario con respecto a la *facilidad de uso*, el *etiquetado* claro y la *detección de errores*.

Además, a las tres dimensiones anteriores se ha añadido la siguiente dimensión:

Facilidad de aprendizaje, es decir, el enfoque debe ser fácil de aprender y entender. De este modo, debe ser fácil para el usuario realizar su trabajo o tarea utilizando el enfoque propuesto.

7.2.2. Cuestionario

En esta evaluación sobre el grado de utilidad y usabilidad de nuestro enfoque, una parte importante ha sido la concerniente al diseño del cuestionario. Después de establecer las dimensiones indicadas en el apartado anterior, para generar los ítems de medición para cada una de ellas, se revisaron una serie de investigaciones relacionadas con marcos de usabilidad, directrices de usabilidad y pruebas de usabilidad [Nielsen 94, Hom 00]. Todos los elementos de medición elegidos fueron modificados para reflejar las características específicas del enfoque metodológico. De esta manera, veinte preguntas conforman el cuestionario (véase la Tabla 22), para capturar las percepciones que tienen los usuarios sobre el grado de utilidad y usabilidad del enfoque metodológico, sobre la base de las cuatro dimensiones de usabilidad propuestas: **Eficacia, Eficiencia, Satisfacción y Facilidad de aprendizaje.** Los usuarios deberán responder a las preguntas sobre su propia experiencia al aplicar el enfoque metodológico semántico.

El cuestionario de la encuesta contiene cuatro preguntas referentes a cada dimensión, excepto la de satisfacción, que incluye siete preguntas; de ellas, dos corresponden a la facilidad de uso y cada una de las restantes están relacionadas con uno de estos subcriterios: etiquetado claro y detección de errores. La última pregunta del cuestionario es concerniente a medir la usabilidad global del enfoque metodológico semántico que se quiere evaluar, y estaría en consonacia con el indicador OSI (*Overall Satisfaction Index*) utilizado en SERVQUAL.

El cuestionario para evaluar la usabilidad consta de una escala de medición con cinco etiquetas lingüísticas (MB=Muy Bajo, B=Bajo, M=Medio, A=Alto, MA=Muy alto). Esta escala es utilizada para detectar tanto la *Adecuación* como la *Excelencia* del enfoque metodológico semántico propuesto; por ello, se debe determinar el *valor mínimo*, el *valor percibido* y el *valor esperado* de cada usuario con respecto a cada uno de los ítems a evaluar para dicho enfoque metodológico. El primero se refiere al grado de usabilidad mínimo que el usuario cree que debe tener el enfoque metodológico semántico evaluado. El segundo se refiere al grado de usabilidad que dicho usuario percibe al usarlo. El tercero se refiere al grado de usabilidad que se espera, es decir, se relaciona con las expectativas que el usuario tiene con respecto a dicho enfoque metodológico.

7.2.3. Procedimiento de evaluación aplicado

La encuesta para evaluar la utilidad y usabilidad de nuestro enfoque metodológico semántico se aplicó a 32 estudiantes de la Institución Educativa de la BUAP, que, como se mencionó anteriormente, cursan los últimos cursos de la Ingeniería en Tecnologías de la Información de la Facultad de Ciencias de la Computación. A estos estudiantes se les indicó que debían desarrollar un *groupware* que resolviera un problema, y que lo podían hacer en equipo (de cinco estudiantes como máximo) o de manera individual. La aplicación colaborativa a desarrollar podía ser de soporte a un trabajo en grupo formal o de ocio. Se conformaron siete equipos: tres con cinco integrantes, dos con cuatro y dos con tres miembros. Tres estudiantes trabajaron de manera individual. Siete proyectos groupware fueron de índole formal y tres de ocio, de tal manera que el enfoque metodológico semántico se probó en los dos tipos de *groupware* que existen.

En primer lugar, se les explicó el enfoque metodológico (véanse las secciones 6.1 y 6.2) y los conceptos del dominico CSCW requeridos para entenderlo (véase la sección 6.3).

Tabla 22. **Cuestionario para la evaluación de la usabilidad del enfoque.**

Dimensión	Pregunta
Eficacia	¿Puedes especificar los requisitos usando el enfoque metodológico?
	¿Has tenido éxito en la descripción del Modelo (estructura organizacional del grupo) mediante el enfoque metodológico?
	¿Las tablas del Modelo, del Controlador y de la Vista cubren las necesidades para crear una aplicación colaborativa?
	En general, ¿el enfoque metodológico es útil para ayudarte a desarrollar una aplicación colaborativa?
Eficiencia	¿Puedes especificar los requisitos rápidamente mediante el enfoque metodológico?
	Cuando utilizas el enfoque metodológico, ¿obtienes rápidamente resultados al desarrollar una aplicación colaborativa?
	¿El enfoque metodológico te permite definir rápidamente los elementos de una aplicación colaborativa?
	¿La cumplimentación de las tablas del Modelo, Controlador y Vista es fácil y rápido de hacer?
Facilidad de Aprendizaje	¿Fue fácil aprender a utilizar el enfoque metodológico?
	¿Los términos utilizados en el enfoque metodológico son comprensibles fácilmente?
	¿El enfoque metodológico ofrece elementos para especificar fácilmente los requisitos?
	¿Los nuevos usuarios del enfoque metodológico podrían utilizarlo para desarrollar groupware sin gran esfuerzo?
Satisfacción	¿Cuál es tu grado de satisfacción global al utilizar el enfoque metodológico?
	¿Cuál es el grado de utilidad del enfoque metodológico?
	¿La organización y distribución de la información (Modelo, Vista, Controlador) en el enfoque metodológico es clara y satisfactoria?
	¿La terminología utilizada en el enfoque metodológico es apropiada, de modo que puede ser usada satisfactoriamente sin mayores problemas?
	¿El enfoque metodológico te ha permitido identificar los elementos de la interfaz de usuario compartida satisfactoriamente?
	¿El enfoque metodológico permite la detección de errores fácilmente?
	¿Las tablas que ofrece el enfoque metodológico como resultado son satisfactorias?
Usabilidad Global	¿Cuál es la facilidad de uso global del enfoque metodológico?
Comentarios	

En segundo lugar, se les pidió que realizaran la especificación de requisitos (véase la sección 6.5.1), empezando con la obtención de requisitos. Para ello, seis equipos utilizaron diagramas de caso de uso de UML y cuatro emplearon historias de usuario de las metodologías ágiles. Después, se les solicitó rellenar las cuatro tablas de especificación correspondientes (véase la sección 6.5).

En tercer lugar, se les requirió llevar a cabo la especificación de diseño (véase la sección 6.5.2), tomando como base las cuatro tablas de especificación anteriormente completadas. Además, se les ordenó elaborar los bocetos de diseño de modelo de datos con *workbench* y los correspondientes a las interfaces de usuario con *balsamiq*.

Finalmente, llevaron a cabo la especificación de la implementación (véase la sección 6.5.3), de acuerdo con las cuatro tablas de especificación completadas anteriormente. En esta parte, se les sugirió que utilizaran un *framework* basado en MVC para desarrollo web.

7.2.4. Análisis de resultados

El cuestionario de evaluación de la utilidad y usabilidad se examinó considerando los siguientes pasos:

Paso1. Los usuarios expresan su opinión rellenando los cuestionarios. La escala o conjunto de etiquetas lingüísticas que los usuarios deben emplear para cumplimentar el cuestionario de la encuesta es: S={MB=Muy Bajo, B=Bajo, M=Medio, A=Alto, MA=Muy Alto}. Como resultado de cada uno de los usuarios, $u_j \in \{u_1, u_2, \dots, u_n\}$, para cada pregunta del cuestionario, $i_k \in \{i_1, i_2, \dots, i_m\}$, se obtiene una tupla con el valor *mínimo* (*vm*), el *valor percibido* (*vp*) y el *valor esperado* (ve), $(vm_{jk}, vp_{jk}, ve_{jk})$, por cada usuario participante u_j para cada pregunta i_k del cuestionario.

Paso2. Calcular la opinión global de los usuarios para cada pregunta. La idea es obtener una tupla (vm_k, vp_k, ve_k) para cada pregunta i_k del cuestionario, que representa la opinión agregada de todos los usuarios para esa pregunta. Para ello, se utiliza uno de los siguientes operadores de agregación, dependiendo del caso a considerar:

- **LOWA**, si consideramos que las opiniones de todos los usuarios tienen la misma importancia.

$$vm_k = \phi_Q(vm_{1k}, \dots, vm_{nk})$$
$$vp_k = \phi_Q(vp_{1k}, \dots, vp_{nk})$$
$$ve_k = \phi_Q(ve_{1k}, \dots, ve_{nk})$$

- **LWA**, si a las opiniones de cada usuario se le otorga un grado de importancia distinto.

$$vm_k = \Phi_Q((UI(u_1, i_k), vm_{1k}), \dots, (UI(u_n, i_k), vm_{nk}))$$
$$vp_k = \Phi_Q((UI(u_1, i_k), vp_{1k}), \dots, (UI(u_n, i_k), vp_{nk}))$$
$$ve_k = \Phi_Q((UI(u_1, i_k), ve_{1k}), \dots, (UI(u_n, i_k), ve_{nk}))$$

  donde $UI(u_j, i_k) \epsilon S$ es el grado de importancia lingüística relativa asignada a un usuario u_j para la pregunta i_k.

Paso 3. Calcular la opinión global de todas las preguntas del cuestionario. Se trata de obtener la tupla (vm, vp, ve), que representa la opinión agregada global de todo el cuestionario, teniendo en cuenta los valores calculados en el paso anterior. Se calcula de forma similar a cómo se ha operado en el paso 2, usando también uno de los siguientes operadores de agregación, según el caso:

- **LOWA**, si se considera que todas las preguntas tienen el mismo grado de importancia.

$$vm = \phi_Q(vm_1, \dots, vm_m)$$
$$vp = \phi_Q(vp_1, \dots, vp_m)$$
$$ve = \phi_Q(ve_1, \dots, ve_m)$$

- **LWA**, si se le otorga diferente importancia a cada una de las preguntas.

$$vm = \Phi_Q((II(i_1), vm_1), \dots, (II(i_m), vm_m))$$
$$vp = \Phi_Q((II(i_1), vp_1), \dots, (II(i_m), vp_m))$$
$$ve = \Phi_Q((II(i_1), ve_1), \dots, (II(i_m), ve_m))$$

donde $II(i_k) \epsilon S$ es el grado de importancia lingüística relativa para la pregunta i_k.

Paso 4. Análisis de la brecha para cada pregunta. La zona de tolerancia está situada entre el valor mínimo *vm* y el valor *esperado ve*. La diferencia entre el valor percibido y el mínimo se denomina *brecha de adecuación SA*. Y la *brecha de excelencia o de superioridad* SS se define como la diferencia entre el valor esperado ve y el percibido vp. Para cada pregunta i_k, se calcula SA_k y SS_k.

$$SA_k = (vp_k - vm_k)$$
$$SS_k = (ve_k - vp_k)$$

Finalmente, un aspecto importante al utilizar cuestionarios para medir una determinada cualidad, como es en este caso la evaluación de la utilidad y usabilidad del enfoque metodológico semántico propuesto, consiste en demostrar su **fiabilidad**.

El alfa de Cronbach [Bojórquez 13] permite cuantificar el nivel de fiabilidad de una escala de medida construida a partir de las *k variables observadas*. Suponiendo que las variables están relacionadas con la cualidad de interés, las *k variables* deberían realizar mediciones estables y consistentes, con un elevado nivel de correlación entre las mismas. Se considera un cuestionario **fiable** *siempre y cuando* el *alfa de Cronbach sea mayor a* **0.80**.

La fórmula para calcular el *alfa de Cronbach* es:

$$\propto = \left[\frac{k}{k-1}\right]\left[1 - \frac{\sum_{i=1}^{k} S_i^2}{S_t^2}\right]$$

donde S_i^2 es la varianza del ítem o pregunta *i*, S_t^2 es la varianza de los valores totales observados, y k es el número de ítems o preguntas en el cuestionario.

7.3. Resultados

En esta sección se presentan los resultados obtenidos del análisis cuantitativo y cualitativo realizado a los datos correspondientes a las percepciones del usuario sobre la utilidad y usabilidad del enfoque metodológico propuesto en esta tesis.

7.3.1. Resultados del Cuestionario

En esta parte se presenta tanto el análisis cuantitativo como cualitativo llevado a cabo en la evaluación de la usabilidad del enfoque metodológico semántico. El análisis cualitativo se centra en calcular los operadores de agregación LOWA y LWA, a partir de las etiquetas lingüísticas propuestas en la escala.

Como se ha comentado anteriormente, el operador LOWA se aplica cuando se considera que los usuarios tienen el mismo grado de importancia y se requiere agregar las percepciones de los usuarios con respecto a cada elemento a evaluar. De esta manera, la Tabla 23 resume el resultado de la agregación combinada de la percepción de los usuarios con respecto al enfoque metodológico semántico para cada una de las 20 preguntas del formulario, considerando los tres valores evaluados: *valor mínimo, valor percibido y valor esperado,* así como las brechas correspondientes. Por otra parte, la Figura 39 muestra un gráfico radial que resume las respuestas de los usuarios a los ítems (o preguntas) del cuestionario sobre los niveles: mínimo, percibido y esperado. Se utilizó este tipo de gráfico porque se aconseja su uso para mostrar los resultados obtenidos con el operador LOWA [Herrera 96]. Como se puede observar en el mismo, la

usabilidad del valor mínimo es percibida con un nivel medio por la mayoría de los usuarios, en cambio el valor percibido y el valor esperado tienen una tendencia hacia un nivel de usabilidad alto del Enfoque Metodológico Semántico.

El operador LWA permite conocer la opinión de todos los usuarios sobre los ítems (o preguntas) con un diferente grado de importancia. Lo cual es apropiado para la evaluación realizada en este trabajo, ya que se consideran cuatro dimensiones: *eficiencia, eficacia, facilidad de aprendizaje y satisfacción.* Así que el nivel de importancia variará de acuerdo a la dimensión que se vaya a evaluar, en el caso de medir la *eficacia*, las preguntas 1, 2, 3 y 4 tendrían un grado de importancia MA (Muy Alto) mientras el resto de preguntas un grado de importancia MB (Muy Bajo), dando como resultado la Tabla 24.

Con respecto a la *eficiencia* las preguntas 5, 6, 7 y 8 tendrán el grado de importancia MA; dando como resultado, después de calcular el operador LWA, un valor Medio (M) para el nivel mínimo y un valor Alto (A) tanto para el nivel percibido como para el esperado (véase la Tabla 25).

Tabla 23. **Evaluación del enfoque metodológico semántico.**

Pregunta	Mínimo	Percibido	Esperado	Brecha de Adecuación	Brecha de Excelencia
1	M	A	A	B	MB
2	M	A	A	B	MB
3	M	A	A	B	MB
4	M	A	A	B	MB
5	M	A	A	B	MB
6	M	M	A	B	MB
7	M	A	A	B	MB
8	M	M	A	B	MB
9	M	M	A	B	MB
10	B	A	A	M	MB
11	B	A	A	M	MB
12	B	A	M	M	B+
13	B	A	M	M	B+
14	B	A	M	M	B+
15	M	A	A	B	MB
16	M	A	A	B	MB
17	B	A	A	M	MB
18	B	M	A	M	MB
19	M	A	A	B	MB
20	M	A	A	B	MB

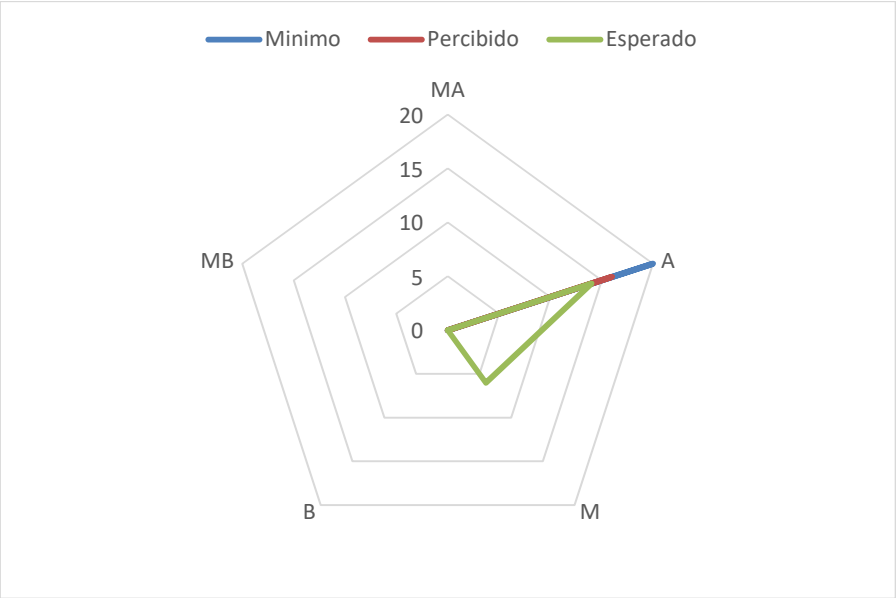


Figura 39. Gráfica de evaluación del enfoque**.**

Tabla 24. Evaluación global del enfoque relacionada a la dimensión de eficacia.

LWA Eficacia		
Mínimo	**Percibido**	**Esperado**
M	A	A

Tabla 25. Evaluación global del enfoque relacionada a la dimensión de eficiencia.

LWA Eficiencia		
Mínimo	**Percibido**	**Esperado**
M	A	A

Para la *facilidad de aprendizaje* el grado de importancia MA sería para las preguntas 9, 10, 11 y 12; por tanto al aplicar el operador LWA, el valor mínimo tendría en promedio el valor B, mientras el percibido y esperado el valor A (véase la Tabla 26).

Tabla 26. Evaluación global del enfoque de la dimensión de facilidad de aprendizaje.

LWA Facilidad de Aprendizaje		
Mínimo	**Percibido**	**Esperado**
B	A	A

Finalmente, en el caso de la *satisfacción* las preguntas 13, 14, 15, 16, 17, 18 y 19 tendrían el grado de importancia MA, así que el valor mínimo resultante después de aplicar el operador LWA sería B, para el percibido y el esperado el resultado sería A (véase la Tabla 27).

Tabla 27. Evaluación global del enfoque de la dimensión de satisfacción.

LWA Satisfacción		
Mínimo	**Percibido**	**Esperado**
B	A	A

Por otra parte, en lo que respecta al análisis cuantitativo. La pregunta 20 se ha planteado para medir la *satisfacción global* del usuario con respecto al enfoque metodológico. En la Figura 40 se puede observar que 19 de los 32 encuestados tienen un nivel Alto (A) de satisfacción global al evaluar la utilidad y usabilidad del enfoque metodológico, mientras que 5 le otorgan un nivel Muy Alto (MA), 7 un nivel Medio (M) y 1 un nivel Bajo (B).

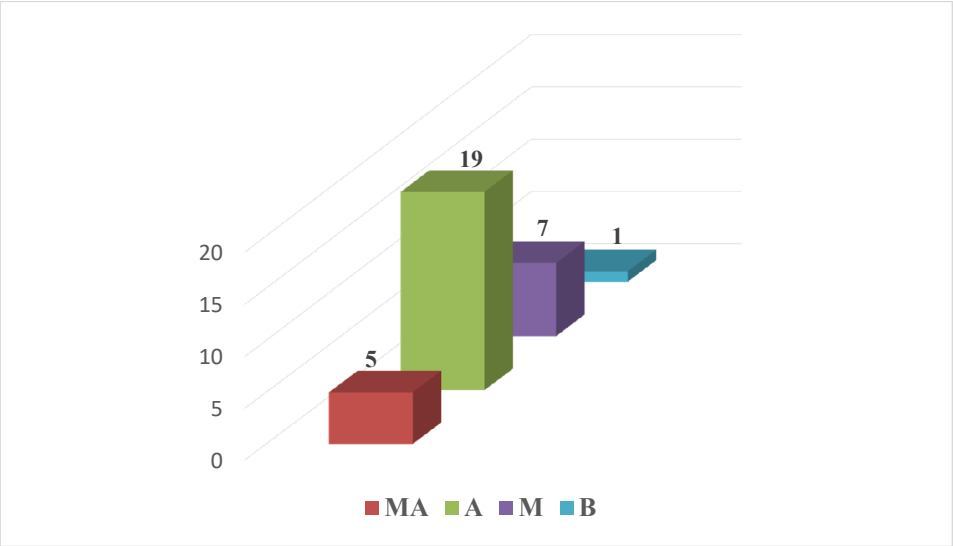


Figura 40. Satisfacción global del enfoque metodológico.

La fiabilidad del cuestionario de evaluación, como se mencionó anteriormente, se ha calculado mediante el *alfa de Cronbach*, obteniendo un valor de **0.91 para el valor *mínimo***, **0.91 para el *percibido* y 0.92 para el *esperado*,** con lo cual se puede decir que la **fiabilidad** del cuestionario **es bastante aceptable**.

Además de medir la **fiabilidad** del cuestionario con el coeficiente *alfa de Cronbach,* se aplicará estadística descriptiva, utilizada en las escalas de Likert, a los datos obtenidos en el cuestionario, con la finalidad de comparar los resultados obtenidos con los operadores LOWA y LWA, para comprobar que se obtienen los mismos resultados. Para ello, se asignan valores a las etiquetas lingüísticas, de tal manera que MA es igual a 5, A equivale a 4, M a 3, B a 2 y MB a 1.

A continuación se presenta los resultados de aplicar la estadística descriptiva a cada valor establecido (*mínimo, percibido y esperado*).

En primer lugar, para el valor *mínimo*, se presenta una Tabla de Frecuencias (véase la Tabla 28) agrupada por cada dimensión considerada, a excepción de la *satisfacción*, donde además se detalla cada uno de los sub-criterios que la constituyen (*facilidad* de uso, etiquetado claro y detección de errores). Como se aprecia en la última fila de la Tabla 28 la Moda es la etiqueta M, al igual que en el operador LOWA (véase la 23 y Figura 41). Además, por ejemplo: en la dimensión de *eficiencia* predomina la etiqueta M, igual que en el operador LWA aplicado a esa dimensión (véase la Tabla 25) y en la dimensión de *facilidad de aprendizaje* coincide el valor de B para la estadística descriptiva y el operador LWA (véase la Tabla 26).

Tabla 28. Frecuencia de las etiquetas lingüísticas para el valor mínimo.

Dimensión	Preguntas	MA	A	M	B	MB	Número de Respuestas
Eficacia	1, 2, 3 y 4	20	40	56	8	4	128
Eficiencia	5, 6, 7 y 8	10	42	64	8	4	128
Facilidad de Aprendizaje	9, 10, 11 y 12	11	57	50	5	5	128
Facilidad de Uso	13, 14, 15, 16, 19	12	52	50	30	16	160
Etiquetado	17	1	18	7	3	3	32
Detección de errores	18	2	10	7	10	3	32
Usabilidad Global	20	1	7	6	16	2	32
	Total	**57**	**226**	**240**	**80**	**37**	**640**

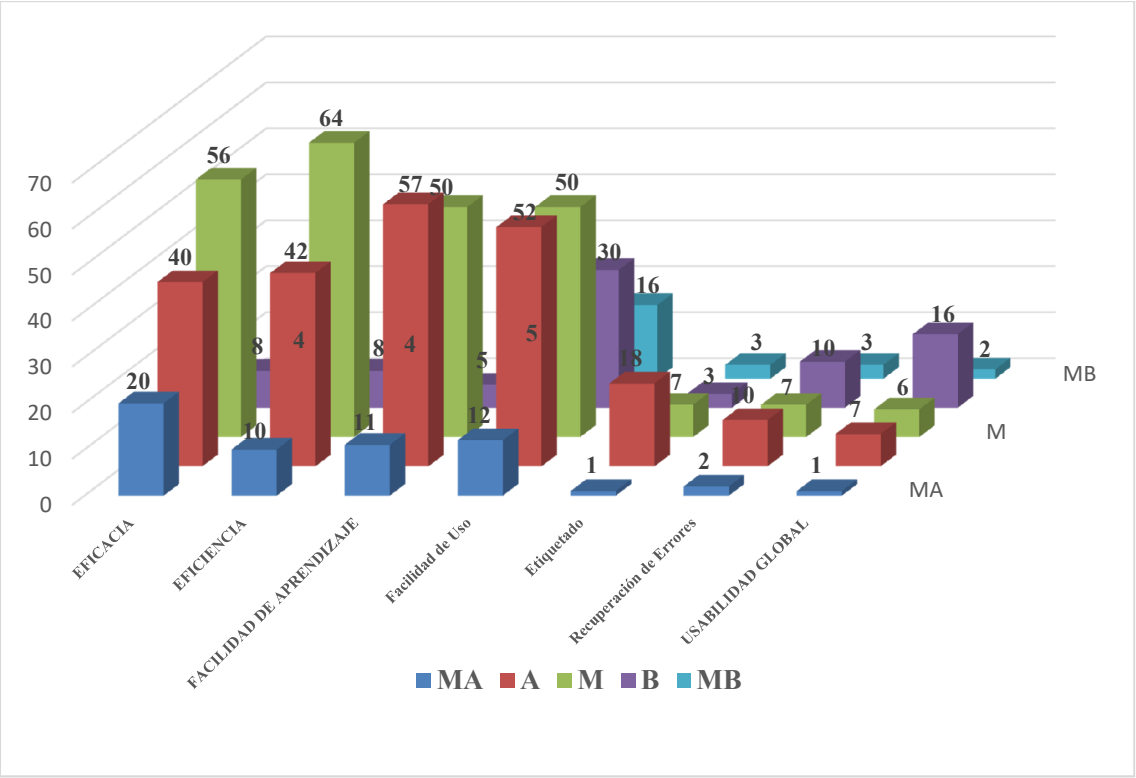


Figura 41. Frecuencia de las etiquetas lingüísticas para el valor mínimo.

En segundo lugar, para el valor percibido se puede apreciar en la Tabla 29 y en la Figura 42 que la etiqueta que prevalece es la A al igual que ocurre al calcular el operador LOWA (véase la Tabla 23 y Figura 43) y LWA para cada dimensión (véase las Tablas 24 a 27).

Tabla 29. Frecuencia de las etiquetas lingüísticas para el valor percibido.

Dimensión	Preguntas	MA	A	M	B	MB	Número de Respuestas
Eficacia	1, 2, 3 y 4	28	48	42	7	3	128
Eficiencia	5, 6, 7 y 8	19	47	50	8	4	128
Facilidad de Aprendizaje	9, 10, 11 y 12	8	62	55	2	1	128
Facilidad de Uso	13, 14, 15, 16, 19	32	70	42	10	6	160
Etiquetado	17	4	16	10	1	1	32
Detección de errores	18	9	11	10	1	1	32
Usabilidad Global	20	5	16	7	2	2	32
	Total	**105**	**270**	**216**	**31**	**18**	**640**

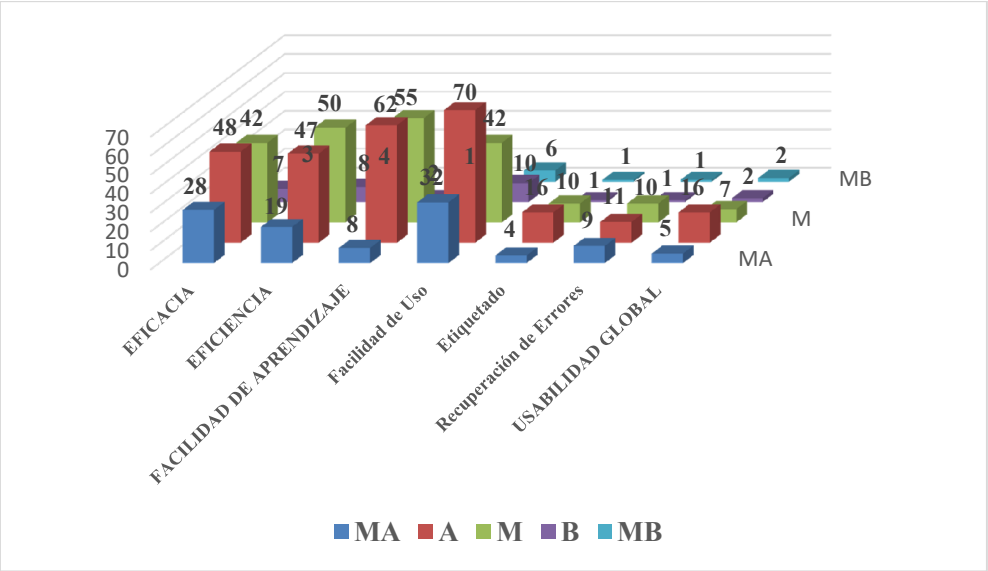


Figura 42. Frecuencia de las etiquetas lingüísticas agrupadas para el valor percibido.

Finalmente para el valor esperado, se generan la Tabla 30 y la Figura 43, dónde se observa que la etiqueta que prevalece es la A al igual que ocurre al calcular el operador LOWA (véase la Tabla 23 y Figura 43) de manera general y LWA para cada dimensión (véase las Tablas 24 a 26).

Tabla 30. Frecuencia de las etiquetas lingüísticas para el valor esperado.

Dimensión	Preguntas	MA	A	M	B	MB	Número de Respuestas
Eficacia	1, 2, 3 y 4	38	52	30	5	3	128
Eficiencia	5, 6, 7 y 8	35	56	37	4	0	132
Facilidad de Aprendizaje	9, 10, 11 y 12	40	42	41	7	2	132
Facilidad de Uso	13, 14, 15, 16, 19	47	50	46	13	4	160
Etiquetado	17	15	3	8	4	2	32
Detección de errores	18	9	5	8	6	4	32
Usabilidad Global	20	11	14	5	1	1	32
	Total	**195**	**222**	**175**	**40**	**16**	**648**

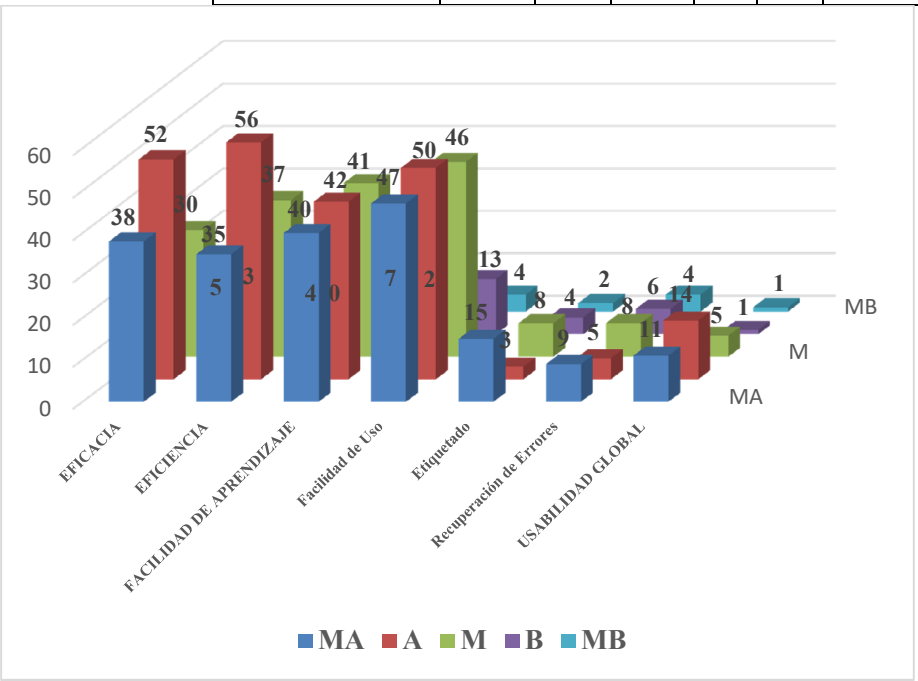


Figura 43. Frecuencia de las etiquetas lingüísticas para el valor esperado.

7.3.2. Recomendaciones resultantes del análisis efectuado

En general, los resultados de la evaluación sobre la utilidad y usabilidad del enfoque metodológico propuesto han sido favorables. Sin embargo, la brecha de adecuación indica que deben hacerse mejoras, centradas principalmente en dos dimensiones: *la facilidad de aprendizaje y la satisfacción*.

En la primera, se deben orientar a mejorar *la facilidad de aprendizaje* del enfoque metodológico, para ello, se debe capacitar al desarrollador en la elicitación y obtención de requisitos.

Con respecto a la segunda, *satisfacción*, las modificaciones se deben orientar principalmente a mejorar la facilidad de uso y la detección de errores. Por tanto, se recomienda proporcionar a los desarrolladores una interfaz web —que permita completar los cuatro *templates* de la especificación de requisitos de manera rápida y fácil — y una herramienta —que genere semi-automáticamente el *groupware* — (trabajos futuros).

8. Conclusiones y trabajo futuro

En este capítulo se presentan las conclusiones finales, las principales aportaciones realizadas y la las descripción de las posibles líneas de investigación abiertas, que sería interesante abordar en trabajos futuros.

8.1. Conclusiones

En este trabajo se ha presentado la propuesta, aplicación y validación del enfoque metodologico semántico para desarrollar *groupware,* principal contribución del presente proyecto de tesis doctoral.

Para el desarrollo de este enfoque metodologico se integraron conocimientos y tecnologías de distintas áreas de investigación, como son: CSCW, *groupware,* gestión de sistemas basados en políticas, arquitecturas de software, patrones de diseño de software, ontologías, gestión del conocimiento, *workflows*, servicios web, ingeniería de software, usabilidad, entre otras. Por tanto se realizó:

- Una revisión exhaustiva de la bibliografía relacionada con el proceso de desarrollo de *groupware* (véase el capìtulo 2), a partir de cuatro enfoques de desarrollo: *ad-hoc, basado en toolkits, basado en componentes y modelo conceptual*; que ha permitido: centrar el trabajo de investigacion en el enfoque de desarrollo basado en el modelo conceptual; determinar los elementos esenciales que sustentan y fortalecen este desarrollo, especificar la estructura organizacional de grupo como eje central de dicho desarrollo y establecer como principal objetivo un enfoque metodológico que guía al desarrollador en tal proceso.
- Un análisis profundo de estilos arquitectónicos relacionados con *groupware* (véase el capìtulo 3), para determinar el estilo arquitectónico apropiado en el desarrollo de este tipo de aplicaciones y proponer un patrón arquitectónico MVC personalizado con los elementos esenciales determinados en la revisión del proceso de desarrollo de *groupware*.
- Un estudio de los principales modelos que han permitido especificar los elementos necesarios para el desarrollo de software. Gracias a este estuido y a la identificación de las carencias en el desarrollo de *groupware,* este estudio se centró en buscar un modelo formal que permita explícitamente representar el dominio de CSCW. Por consiguiente, se revisaron las ontologías (véase el capìtulo 4) para especificar dicho desarrollo mediante un vocabulario explìcito y fomal; así como también la propia metodología.
- Una revisión de la gestión de sistemas basados en políticas (véase la sección 6.2.1.2) para permitir separar la lógica de la aplicación de las reglas que la regulan y determinan su comportamiento. Lo cual permitió deducir que las ontologías son ideales para modelar tal dominio; así como crear un modelo ontológico de la política de manejo de sesión, que permite regular el comportamiento de la estructura organizacional de grupo.
- Un análisis de *workflow* (véase el capítulo 5) para gestionar y controlar tanto la estructura organizacional de grupo como la metodología; permitiendo establecer el conjunto de pasos a seguir para desarrollar groupware de una manera formal y explícita. Lo que condujo a establecer la ontología *workflow* de este enfoque metodológico.
- Un estudio del ciclo de vida de desarrollo de software (véase la sección 6.5) para determinar las etapas que comprende el enfoque metodológico propuesto. Así como el uso de *templates* para simplificar la especificación de las instancias de la ontología *workflow.*
- Una revisión de metodologías de evaluación para validar el enfoque metodológico semántico centrado en la utilidad y la usabilidad del mismo. El cual demostró que el enfoque metodológico es una guía que ayuda al desarrollador en el análisis, diseño e implementación de *groupware* con una fiabilidad cercana al 1. Esta metodología de evaluación combinó los principios básicos de la usabilidad (establecimiento de cuatro dimensiones, el cuestionario de 20 pregutnas y la escala

de medición) con modelos de calidad de servicio (la brecha de calidad de servicio) y modelos de la lógica difusa (operadores lingüísticos LOWA y LWA) aplicados específicamente en la utilidad y usabilidad del enfoque metodológico.

8.2. Aportaciones

La contribución primordial del presente trabajo de investigación es la propuesta de un enfoque metodológico semántico para desarrollar *groupware;* para lo cual, se ha realizado una profunda revisión de la bibliografía relacionada y se ha logrado obtener un conjunto de aportaciones que sustentan y fortalecen este trabajo de tesis:

- La separación de los mecanismos para establecer la sesión de las formas de implantar la organización del trabajo en grupo (la política de manejo de sesión) en dicha sesión. Esto permite lograr la modularidad y la flexibilidad en el desarrollo de *groupware*, facilitando la modificación del trabajo de grupo de forma que se adapte a diferentes formas de organización, necesidades e incluso grupos.
- La especificación de los conceptos del modelo ontológico de la política de manejo de sesión, que permiten la comunicación, coordinación y colaboración de los usuarios simplificando la interacción en el entorno colaborativo.
- La incorporación de conceptos como *fase*, precedencia de tarea, precedencia de fase, tarea secuencial, tarea paralela, tarea parcialmente concurrente y tarea totalmente concurrente, contribuyen a suministrar la coordinación a nivel de actividad y a nivel de objeto.
- Se ha presentado el modelo ontológico de la política de manejo de sesión, que permite especificar de manera formal y explícita la estructura organizacional del grupo, permitiendo lograr su adaptación al agregar, cambiar o eliminar clases, relaciones y/o axiomas de ese modelo.
- El estilo arquitectónico Modelo-Vista-Controlador personalizado para el desarrollo de *groupware*, tanto para reuniones de trabajo profesional como de ocio, que permite organizar el desarrollo de *groupware* en tres partes y facilita la implementación al existir *frameworks* basados, precisamente, en MV*C*.
- La especificación de los componentes de la capa Modelo, que establecen los elementos necesarios que deben ser almacenados, gestionados y procesados para el buen funcionamiento del *groupware*.
- La especificación de los componentes de la capa Vista, que constituyen los elementos que serán presentados en la vista de información, participantes y/o contexto del *groupware a desarrollar*.
- La especificación de los componentes de la capa Controlador, que implantan los mecanismos que controlan como el modelo será utilizado de acuerdo a la interacción entre los usuarios o de estos con el sistema, y como las vistas serán actualizadas con respecto a esa interacción y procesamiento de la información.
- Los *templates* para capturar las instancias de la ontología *workflow*, facilitando su manejo y utilización.
- La utilización de fases como una parte fundamental en la estructura que define la organización del grupo, para simplificar y mejorar el control de acceso de los usuarios (o actores) al *groupware.*
- Los tipos de tarea para determinar tanto el mecanismo a utilizar (notificación o concurrencia) como la vista a presentar (de información, participantes y/o contexto).
- La ontología *workflow* para guiar al desarrollador en la creación de un *groupware.*
- Los modelos que forman parte de la metodología para desarrollar *groupware*.
- La metodología de la evaluación de la usabilidad del enfoque metodológico semántico.
- El enfoque metodológico propuesto se puede aplicar tanto al desarrollo de groupware como cualquier otro tipo de software.

8.3. Trabajo futuro

El enfoque metodológico semántico propuesto ha abierto la posibilidad de explorar nuevas líneas de investigación, tales como:

- Reglas semánticas para establecer de manera automática las instancias de una ontología en OWL.
- Integración de metodologías ágiles con enfoques semánticos para el desarrollo de sistemas en el dominio CSCW.
- Interacción del desarrollo Web con la Web semántica, para simplificar la especificación de instancias de la ontología *workflow,* de tal manera, que sea posible:
  - Almacenar la información de los distintos proyectos del *groupware* realizados con el enfoque, tanto en la ontología de la herramienta protégé como en la base de datos correspondiente.
  - Generar los modelos de control de acceso, de conciencia y memoria de grupo de forma automática.
  - Crear una base de conocimiento que permita deducir el comportamiento de los usuarios en el trabajo de grupo, a partir de la experiencia en situaciones colaborativas similares almacenadas en esta base de conocimiento. .
- Desarrollo de un *framework* basado en el enfoque metodológico semántico para la creación de *groupware* de forma semiáutomática:
  - Proporcionar un conjunto de librerías que permitan generar de manera semi-automática el componente Modelo del groupware a crear.
  - Suministrar una serie de librerías para determinar y codificar los métodos necesarios para el funcionamiento adecuado de la aplicación.
  - Ofrecer un conjunto de elementos que permitan crear las interfaces de usuario de la aplicación de manera semi-automática.
  - Integrar una librería para realizar las pruebas del *groupware* desarrollado de manera automática; validando de esta manera su funcionamiento.

Referencias

[Aalst 04] Aalst, W & Hee, K. (2004). Workflow management: models, methods, and systems; MIT Press.

[Addy 98] Addy, E. A. (1998). Report from the First annual work-shop on software architectures in product line acquisitions. ACM SIGSOFT Software Engineering Notes, vol. 23-3.

[Addy 15] Addy, O. (2015). Learning javascript design patterns. O'Reilly,

[Allen 01] Allen, R. (2001). Workflow: An introduction. In workflow handbook, L. Fisher, (Ed.) Future Strategies, pp. 15–38.

[Alonso 04] Alonso, G., Casati, F., Kuno, H. & Machiraju, V. (2004) Web services: concepts, architectures and applications. Springer–Verlag.

[Andersen 00] Andersen A., Gordon S. & Blair, F. E. (2000). A reflective component-based middleware with quality of service management. Accepted for publication in Conference on Protocols for Multimedia Systems.

[Angele 04] Angele, J. & Lausen, G. (2004). Ontologies in F-logic. In Steffen Staab and Rudi Studer, editors, Handbook on Ontologies, Springer, pp. 29–50.

[Anzures 06a] Anzures-García M., Hornos M. J. & Paderewski, P. (2006). SOA-based generic architecture for CSCW systems. CEUR Workshop Proceedings Series, vol. 208, pp. 117-124.

[Anzures 06b] Anzures-García M., Hornos M. J. & Paderewski, P. (2006). Propuesta arquitectónica para el desarrollo de aplicaciones colaborativas de calidad. Revista Internacional Gerencia Tecnológica Informática (GTI) - Informatics Technology Management, vol. 5-13, pp. 31-40.

[Anzures 06c] Anzures-García M., Hornos M. J. & Paderewski, P. (2006). Servicio de registro basado en servicios web para aplicaciones colaborativas. In Proc. II Jornadas Científico-Técnicas en Servicios Web (JSWEB), pp. 114-121.

[Anzures 07a] Anzures-García M., Hornos M. J. & Paderewski P. (2007). Development of extensible and flexible collaborative applications using a web service-based architecture. Lecture Notes in Computer Science, vol. 4401, pp. 66–80. Springer.

[Anzures 07b] Anzures-García M., Sánchez-Gálvez L. A., Hornos M.J. & Paderewski-Rodríguez, P. (2007). Ontology-based modelling of session management policies for groupware applications. Lecture Notes in Computer Science, vol. 4739, pp. 57–64, Springer.

[Anzures 07c] Anzures-García M., Hornos M. J. & Paderewski, P. (2007). Architecture for developing adaptive and adaptable collaborative applications. Lecture Notes in Computer Science, vol. 4758, pp. 271-274.Springer.

[Anzures 08a] Anzures-García M. & Sánchez-Gálvez L. A. (2008). Policy-based group organizational structure management using an ontological approach. In Proc. International Conference on Availability, Reliability and Security (ARES), pp. 807-812.

[Anzures 08b] Anzures-García M. & Sánchez-Gálvez L. A. (2008). Group organizational structure adaptation for groupware applications. In Proc. International Conference on Complex, Intelligent and Software Intensive Systems (CISIS), pp. 445-440.

[Anzures 09a] Anzures-García M., Sánchez-Gálvez L. A., Hornos M.J. & Paderewski-Rodríguez, P. (2009). Service-based layered architectural model for building collaborative applications in heterogeneous environments. In Proc. X Encuentro Nacional en Computación, pp, 79-80.

[Anzures 09b] Anzures-García M., Sánchez-Gálvez L. A., Hornos M.J. & Paderewski-Rodríguez, P. (2009). Service-based access control using stages for collaborative systems. Research in Computing Science, vol. 42, pp. 311-322.

[Anzures 10a] Anzures-García M., Sánchez-Gálvez L. A., Hornos M.J. & Paderewski-Rodríguez, P. (2010). Service-based models to provide group interaction and dynamism in collaborative applications. In Proc. of the 6th International Conference on Networking and Services (ICNS), pp. 67-72.

[Anzures 10b] Anzures-García M., Sánchez-Gálvez L. A., Hornos M.J. & Paderewski-Rodríguez, P. (2010). Security and adaptability to groupware applications using a set of SOA-based services. Research in Computing Science, vol. 45, pp. 279-290.

[Anzures 11a] Anzures-García M., Sánchez-Gálvez L. A., Hornos M.J. & Paderewski-Rodríguez, P. (2011). SAMCA: Service-based architectural model for collaborative applications. In Proc. 15TH IASTED International Conference on Software Engineering and Applications (SEA), pp. 269-276.

[Anzures 11b] Anzures-García M., Sánchez-Gálvez L. A., Hornos M.J. & Paderewski-Rodríguez, P. (2011). Methontology-based ontology representing a service-based architectural model for collaborative applications. In Proc. 10th Mexican International Conference on Artificial Intelligence. Research in Computing Science, vol. 54, pp. 77-90.

[Anzures 14a] Anzures-García M., Sánchez-Gálvez L. A., Hornos M.J. & Paderewski-Rodríguez, P. (2014). Utilizando un modelo arquitectónico en el desarrollo de una aplicación colaborativa. Aportaciones de Redes Innovadoras en Tecnología Educativa. Capítulo 4. Dirección de Fomento Editorial. Primera Edición, pp. 45 -61.

[Anzures 14b] Anzures-García M., Sánchez-Gálvez L.A., Hornos M.J. & Paderewski-Rodríguez, P. (2014). A semantic approach to develop groupware. Research in Computing Science: Advances in Computer Science and Engineering, vol. 83, pp. 71-83.

[Anzures 15a] Anzures-García M., Sánchez-Gálvez L.A., Hornos M.J. & Paderewski-Rodríguez, P (2015). A Knowledge Base for the Development of Collaborative Applications. Engineering Letters, vol. 23-2, pp. 65-71.

[Anzures 15b] Anzures-García M., Sánchez-Gálvez L.A., Hornos M.J. & Paderewski-Rodríguez, P. (2015). A software architecture for defining a methodologic approach to develop collaborative applications. Research in Computing Science: Advances in Computer Science and Engineering, vol. 105, pp. 9-20, 2015,

[Anzures 16a] Anzures-García M., Sánchez-Gálvez L.A., Hornos M.J. & Paderewski-Rodríguez, P. (2016). A semantic formalism to model the group interaction. Research in Computing Science: Advances in Computer Science and Engineering, vol. 188, pp. 137-147.

[Anzures 16b] Anzures-García M., Sánchez-Gálvez L.A., Hornos M.J. & Paderewski-Rodríguez, P. (2016). A knowledge-based workflow ontology for group organizational structure. Research in Computing Science: Advances in Languages and Knowledge Engineering vol. 123 pp. 79-90.

[Anzures 17a] Anzures-García M., Sánchez-Gálvez L. A., Hornos M. J. & Paderewski-Rodríguez, P. (2017). Tutorial function groupware based on a workflow ontology and a directed acyclic graph. IEEE Latin American Transactions. Accepted.

[Anzures 17b] Anzures-García M., Sánchez-Gálvez L.A., Hornos M.J. & Paderewski-Rodríguez, P. (2017). Facilitating the development of collaborative applications with the MVC architectural pattern. Chapter 15 of the Book Software Engineering: Methods, Modeling, and Teaching, vol. 4, pp. 268-290. (Ed.) Bonaventuriana, Universidad de San Buenaventura.

[Araújo 97] Araújo, R., Dias, M. & Borges, M. (1997). A framework for the classification of computer supported colaborative design approaches. III CRIWG, pp. 91-100.

[Arch 92] Arch. A. (1992). A Metamodel for the runtime architecture of an interactive system. The UIMS Tool Developers Workshop, SIGCHI Bulletin, ACM vol. 24-1, pp. 32-37.

[Arruda 03] Arruda, C., Bulcao, R. & Pimentel, M.G. (2003). Open context-aware storage as a web service. In Proceedings of the International Workshop on Middleware for Pervasive and Ad-Hoc Computing– ACM/IFIP/USENIX International Middleware Conference, pp. 81–87.

[Arsanjani 09] Arsanjani, A. (2009). Service-oriented modeling and architecture. How to identify, specify, and realize services for your SOA. IBM Technical Library.

[Baader 03] Baader, F., Calvanese, D., MacGuinness, D., Nardi, D. & Patel-Schneider. P. (2003). The Description Logic Handbook, Cambrigde.

[Barbosa 03] Barbosa, S.D.J. & de Paula, M.G. (2003). Designing and evaluating interaction as conversation: a modeling language based on semiotic engineering. In: Jorge, J.A., Jardim Nunes, N., Falcão e Cunha, J. (eds.) DSV-IS 2003. LNCS, vol. 2844, pp. 16–33. Springer.

[Barbosa 13] Barbosa, E.F., Fioravanti, M.L., Nakagawa, E.Y. & Maldonado, J.C. (2013). Towards the establishment of a reference architecture for developing learning environments, in Software Engineering Knowledge, pp. 350–355.

[Bayle 98] Bayle, E. (1998). Putting it all together: Towards a pattern language for interaction design, SIGCHI Bulletin, vol. 30-1, pp.17-24.

[Beaudouin 94] Beaudouin-Lafon, M. (1994). Beyond the workstation: Media spaces and augmented reality. In Proceedings of the Conference on People and computers IX, vol. 9, pp. 9-18,

[Beaudouin 99] Beaudouin-Lafon, M. (1999). Computer supported cooperative work. Trends in Software, John Wiley & Sons.

[Becker 00] Becker, J., Rosemann, M. & von Uthmann, C. (2000). Guidelines of Business Process Modeling, W. van der Aalst et al. (Eds.): Business process management, LNCS vol.1806, pp. 30-49, Springer,

[Bell 08] Bell, M. (2008). Service-oriented modeling (SOA): Service analysis, design, and architecture, (Ed.) Wiley.

[Benford 98] Benford, S. (1998). Understanding and constructing shared spaces with mixed-reality boundaries. ACM Transactions on Computer-Human Interaction, vol. 5-3, pp. 185-223.

[Bloomberg 06] Bloomberg, J. & Schmelzer, R. (2006). Service Oriented Architecture or Be Doomed!: How Service Orientation Will Change Your Business. John Wiley & Sons, Hoboken.

[Bojórquez 13] Bojórquez, J. A., López, L., Hernández, M. E. & Jiménez, E. (2013). Utilización del alfa de Cronbach para validar la confiabilidad de un instrumento de medición de satisfacción del estudiante en el uso del software Minitab. Eleventh LACCEI Latin American and Caribbean Conference for Engineering and Technology (LACCEI'2013). "Innovation in Engineering, Technology and Education for Competitiveness and Prosperity", pp. 14 - 16.

[Booch 06] Booch, G., Rumbaugh, J. & Jacobson, I. (2006). El lenguaje unificado de modelado. 2a. Edición. Pearson Educación, Madrid.

[Borchers 01] Borchers, J. (2001). A Pattern Approach to Interaction Design. Wiley.

[Borghoff 98] Borghoff, U. M. & J. H. Schlichter. (1998). Computer-Supported Cooperative Work. Springer-Verlag.

[Borst 97] Borst. W. (1997). Construction of engineering ontologies. PhD thesis, Institute for Telematics and Information Technology, University of Twente, Enschede, The Netherlands.

[BPMI 05] BPMI http://www.bpmi.org. (2005).

[Brachman 85] Brachman, R. J. & Schmolze, J.G. (1985). An overview of the KL-ONE knowledge representation system. Cognitive Science, vol. 9-2, pp. 171–216,

[Bray 06] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E. & Yergeau, F. (2006). Extensible Markup Language (XML) 1.0 (Fourth Edition). W3C Recommendation. Available http://www.w3.org/TR/REC-xml/.

[Brown 98] Brown, A. & Wallnau, K. (1998). The current state of CBSE. IEEE Software, pp.37-46.

[Bulcao 04] Bulcao, R., Jardim, C., Camacho-Guerrero, J. & Pimentel, M.C. (2004). A web service approach for providing context information to CSCW applications. Springer-Verlag.

[Bullock 99] Bullock, A. & Benford, S. (1999). An access control framework for multi-user collaborative environments. ACM GROUP.

[Burridge 99] Burridge, R. (1999). Java Shared Data Toolkit User Guide version 2.0. Sun Microsystems, JavaSoft Division.

[Buschmann 96] Buschmann, F. (1996). Pattern-oriented Software Architecture. A System of Patterns, vol. 1. John Wiley & Sons Ltd.

[Calvary 97] Calvary, G., Coutaz, J. & Nigay, L. (1997). From single-user architectural design to PAC*: A generic software architecture model for CSCW. Proc. of Computer-Human Interaction, pp. 242-249,

[Casanave 09] Casanave, C. (2009). Enterprise service oriented architecture using the omg SoaML standard. A model driven solutions, Inc. White Paper.

[Chabert 98] Chabert, A., Grossman, E., Jackson, L., Pietrowicz, S. & Seguin, C. (1998). Java object sharing in Habanero. Communications of the ACM, vol. 41-6, pp. 69-76.

[Cherif 12] Cherif, A. (2012). Access control models for collaborative applications. Doctoral dissertation.

[Christensen 01] Christensen, E., Curbera, F., Meredith, G. & Weerawarana, S. (2001) Web services description language (WSDL) 1.1. W3C Note 15 March 2001.

[Cid-López 15] Cid-López, A., Hornos, M. J., Carrasco, R. A. & Herrera-Viedma, E. (2015). SICTQUAL: A fuzzy linguistic multi-criteria model to assess the quality of service in the ICT sector from the user perspective. Applications Software Computer, vol. 37, pp. 897-910.

[Cid-López 16] Cid-López, A., Hornos, M. J., Carrasco, R. A. & Herrera-Viedma, E. (2016). Applying a linguistic multi-criteria decision-making model to the analysis of ICT suppliers' offers. Experts Systems Applications, vol. 57, pp. 127-138.

[Cid-López 17] Cid-López, A., Hornos, M. J., Carrasco, R. A., Herrera-Viedma, E. & Chiclana, F. (2017). Linguistic multi-criteria decision-making model with output variable expressive richness. Experts Systems Applications, vol. 83, pp. 350-362.

[Clemen 05] Clemen, L., Hately, A., von Riegen, C. & Rogers, T. (2005). Universal description, discovery, and integration (UDDI).

[Corkill 91] Corkill, D. (1991). “Blackboard systems”. AI Experts, vol. 6-9, pp. 40-47.

[Coutaz 97] Coutaz, J. (1997). PAC-ing the architecture of your user interface. Proc. of the Fourth Eurogra-phics Workshop on DSVIS, Sringer-Verlag, pp. 15-32.

[Crowston 06] Crowston, K., Rubleske J. & Howison J. (2006). Coordination theory: A ten-year retrospective. Human-Computer Interaction in Management Information Systems. M. E. Sharpe, Inc.

[Cruz 12] Cruz, A., Correia, A, Paredes, H., Fonseca, B., Morgado, L. & Martins, P. (2012). Towards an overarching classification model of cscw and groupware: A socio-technical perspective. V. Herskovic et al. (Eds.): CRIWG 2012, LNCS vol. 7493, pp. 41–56, 2012. Springer-Verlag.

[Cuesta 00] Cuesta, C. E., de la Fuente, P., Barrio-Solórzano, M. & Beato E. (2000). Arquitectura de software dinámica basada en reflexión. V Jornadas de Ingeniería del Software y Bases de Datos (JISBD'2000), pp. 203-214.

[Cuesta 01] Cuesta, C. E., de la Fuente, P., Barrio-Solórzano, M. & Beato E. (2001). Dynamic coordination architecture through the use of reflection. In Proceedings of 16th ACM Symposium on Applied Computing (SAC2001), pp. 134-140, ACM Press.

[Cuesta 03] Cuesta, C. E. (2003). Tesis doctoral: Arquitectura de software dinámica basada en reflexión. Universidad de Valladolid.

[Damianou 01] Damianou, N., Dulay, N., Lupu, E. & Sloman, M. (2001). The ponder policy specification language. LNCS vol. 1995, pp. 18-38. Springer-Verlag,

[De Souza 15] De Souza, L.G. & Barbosa, S.D.J. (2015). Extending MoLIC for collaborative systems design. Springer International Publishing Switzerland M. Kurosu (Ed.): Human-Computer Interaction, Part I, HCII 2015, LNCS vol. 9169, pp. 271–282.

[Decouchant 09] Decouchant, D., Escalada-Imaz, G., Martinez, A. M., Mendoza, S. & Muhammad, A. (2009). Contextual awareness based communication and coauthoring proximity in the internet. Expert Systems with Applications, vol. 36, pp. 8391–8406.

[Dewan 95a] Dewan, P. (1995). Multiuser architectures. Proc. of the IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction, pp. 15-32.

[Dewan 95b] Dewan, P. & Choudhary, R. (1995). Coupling the user interfaces of a multiuser program. ACM Transactions on Computer Human Interaction, vol. 2-1, pp. 1–39.

[Dewan 99] Dewan, P. (1999). Architectures for collaborative applications. In Beaudouin-Lafon (eds.), Computer Supported Cooperative Work, John Wiley & Sons Ltd, pp. 169-194.

[Dey 01] Dey, A.K. (2001). Understanding and using context. Personal and Ubiquitous Computing Journal, vol. 5-1, pp. 4–7.

[Dix 03] Dix, A. (2003). Human-computer interaction. Pearson/Prentice-Hall, Harlow.

[Dommel 97] Dommel H.P. & Garcia-Luna-Aceves J.J. (1997). Floor control for multimedia conferencing and collaboration. ACM Multimedia, vol. 5-1, pp. 23-38.

[Eastlake 01] Eastlake, D., Reagle, J. & Solo, D. (2001). XML-Signature Syntax and Processing. http://www.ietf.org/rfc/rfc3075.txt.

[Edwards 94] Edwards, W.K. (1994). Session management for collaborative applications. In proceedings CSCW, pp. 323-330.

[Edwards 96] Edwards, W.K. (1996). Policies and roles in collaborative applications. In Proceedings of the ACM Conference on CSCW, pp. 11–20.

[Ellis 89] Ellis, C.A. & Gibbs, S.J. (1989). Concurrency control in groupware systems. In proceedings ACM SIGMOD International Conference on Management of Data, pp. 399-407

[Ellis 91] Ellis, C.A., Gibbs, S.J. & Rein, G.L. (1991). Groupware: some issues and experiences. Communications of the ACM, vol. 34-1, pp. 39-58.

[Ellis 94] Ellis, C. & Wainer, J. (1994). A conceptual model of groupware. Proceedings of the 1994 ACM Conference on CSCW, pp. 79-88.

[Ericksson 00] Eriksson, H. E. & Penker, M.. Business Modeling with UML: Business Patterns at Work. John Wiley & Sons, 2000.

[Erl 04] Erl, T. (2004). Service-oriented architecture: Concepts, technology, and design. Prentice Hall Pearson PTR.

[Erl 15] Erl, T. F. (2015.). Service-oriented architecture (SOA) modeling. Google Patents.

[Farquhar 97] Farquhar, A., Fikes, R. & Rice, J. (1997). The Ontolingua Server: A Tool for Collaborative Ontology Construction. International Journal of Human Computer Studies, vol. 46-6 pp. 707–727.

[Feng 09] Feng, Y. H., Teng, T. H. & Tan, A. H. (2009). Modelling situation awareness for context-aware decision support. Expert Systems with Applications, vol. 36, pp. 455–463.

[Fensel 01] Fensel, D., van Harmelen, F., Horrocks, I., McGuinness, D.L. & Patel-Schneider, P.F. (2001). OIL: An ontology infrastructure for the Semantic Web. IEEE Intelligent Systems & their applications, vol.16-2, pp. 38–44.

[Fioravanti 17] Fioravanti, M. L., Duarte, F. N., Bortolini, L. & Francine, E. (2017). Towards a Mobile Learning Environment using Reference Architectures. Proceedings of the 50th Hawaii International Conference on System Sciences, pp. 6191-6200.

[Fischer 04] Fischer, L. (2004). Workflow Handbook. Future Strategeis Inc., Lighthouse Point, FL.

[Fitzpatrick 95] Fitzpatrick, G., Kaplan, S.M. & Tolone, J. (1995). Work, locales and distributed social worlds. In Proceedings ECSCW, pp. 1-16.

[Fitzpatrick 96] Fitzpatrick, G., Kaplan, S. M. & Mansfield, T. (1996). Physical spaces, virtual places and social worlds: A study of work in the virtual. In Proceedings CSCW, pp. 334-343.

[Fonseca 06] Fonseca, B. & Carrapatoso, E. (2006). SAGA: A Web services architecture for groupware applications, In Proc. of the CRIWG, LNCS vol. 4154, pp. 246-261 Springer-Verlag.

[Fowler 02] Fowler, M. (2002). Patterns of enterprise application architecture. Martin. Addison-Wesley Professional.

[Fox 02a] Fox, G. & Pallickara, S. (2002). The Narada event brokering system: overview and extensions. In Proccedings of the International Conference on Parallel and Distributed Processing Techniques and Applications.

[Fox 02b] Fox G., Sung-Hoon K., Kangseok K., Sangyoon O. & Sangmi L. (2002). Integration of Hand-Held Devices into Collaborative Environment. In Proceedings of the International Conference on Internet Computing (IC-02).

[Fox 03] Fox, G., Sangmi, L., Sung-Hoon, K., Kangseok, K. & Sangyoon, O. (2003). CAROUSEL Web service: Universal accessible Web service architecture for collaborative application. In Proceedings of 1st International Conference on Web Services.

[Fucks 04] Fuks, H., Raposo, A. B., Gerosa, M. A. & Lucena, C. J. P. (2004). Applying the 3C model to groupware engineering. In: Monografias em Ciência da Computação. PUC-Rio.

[Gacitua 09] Gacitua, R., Arguello-Casteleiro, M., Sawyer, P., Des, J., Perez, R., Fernandez-Prieto, M.J. & Paniagua, H. (2009). A collaborative workflow for building ontologies: A case study in the biomedical field. Research Challenges in Information Scienc. pp. 121-128.

[Gallardo 11] Gallardo, J., Molina, A.I., Bravo, C., Redondo, M.A. & Collazos, C.A. (2011). An ontological conceptualization approach for awareness in domain-independent collaborative modeling systems: Application to a model-driven development method. Expert Systems with Applications, vol. 38, pp. 1099–1118.

[Gamma 95] Gamma, E., Helm R., Johnson, R. y Vlissides J. (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Publishing Company.

[García 03] García, P. & Gómez-Skarmeta, (2003). A. ANTS framework for cooperative work environment. In IEEE Computer, vol. 36-3, pp. 56-62.

[Garlan 95] Garlan, D. (1995). What is style? In Proceedings of IWASS'1.

[Garrido 03] Garrido, J. L. (2003). AMENITIES: Una metodología para el desarrollo de sistemas cooperativos basada en modelos de comportamiento y tareas. Departamento de Lenguajes y Sistemas Informáticos. Phd. Thesis, Granada, Universidad de Granada.

[Gea 00] Gea M., Padilla N., Garrido J.L. & Gutiérrez F.L. (2000). Diseño de entornos cooperativos. In Proccedings of the X Congreso Español de Informática Gráfica, pp.143-156.

[Genesereth 92] Genesereth, M.R. & Fikes, R.E. (1992). Knowledge Interchange format. Version 3.0. Reference manual. Technical report logic-92-1. Computer Science Department. Stanford University, California. 1992. http://meta2.stanford.edu/kif/Hypertext/kif-manual.html.

[Giraldo 08] Giraldo, W., Molina, A., Collazos, C., Ortega, M. & Redondo, M. (2008). Taxonomy for integrating models in the development of interactive groupware systems, Journal of Universal Computer Science, vol. 14-19, pp. 3142-3159.

[Goldberg 84] Goldberg, A. (1984). Smalltalk-80: The Interactive Programming Environment. Addison-Wesley.

[Gómez-Pérez 04] Gómez-Pérez, A., Fernández-López, M & Corcho, O. (2004). Ontological engineering with examples from the areas of knowledge management, e-Commerce and the Semantic Web. Springer.

[Graham 95] Graham, T. (1995). Declarative Development of Interactive Systems, vol. 243, Springer Verlag.

[Graham 96] Graham, T., Morton, C.A. & Urnes, T. (1996). ClockWorks: Visual Programming of Component-Based Software Architectures. Journal of Visual Languages and Computing, pp. 175-196.

[Greenberg 94] Greenberg, S. & Marwood, D. (1994). Real time groupware as a distributed system: Concurrency control and its effect on the interface. In Proceedings of the ACM Conference on CSCW, pp. 207–217.

[Greenberg 01] Greenberg, S. (2001). Context as a dynamic construct. Human-Computer Interaction Journal, vol. 16, pp. 257–268.

[Greiner 02] Greiner U. & Rahm E. (2002). Quality-Oriented Handling of Exceptions in Web-Service-Based Cooperative Enterprise Application Integration. pp. 11-18. GITO-Verlag,

[Grimm 07] Grimm, S., Hitzler, P. & Abecker, A. (2007). Knowledge representation and ontologies. In Semantic Web Services: Concepts. Technologies and Applications, pp. 51-105, Springer-Verlag.

[Gruber 93] Gruber, R. (1993). A translation approach to portable ontology specification. Knowledge Acquisition, vol. 5, pp. 199-220.

[Gruber 93b] Gruber, T.R. (1993). Toward principles for the design of ontologies used for knowledge sharing. In: Guarino N, Poli R (Eds.) International Workshop on Formal Ontology in Conceptual Analysis and Knowledge Representation. Padova, Italy. (Formal Ontology in Conceptual Analysis and Knowledge Representation) Kluwer Academic Publishers, Deventer. The Netherlands. http://citeseer.nj.nec.com/gruber93toward.html

[Gruber 95] Gruber, T. R. (1995). Toward Principles for the Design of Ontologies Used for Knowledge Sharing. International Journal of Human Computer Studies, vol. 43(5–6), pp. 907–928.

[Grudin 94] Grudin, J. (1994). Computer-Supported Cooperative Work: Its History and Participation. IEEE Computer, vol. 27-5, pp. 19-26.

[Grudin 97] Grudin, J. & Poltrock, S.E. (1997). Computer-Supported Cooperative Work and groupware. In: Zelkowitz, M. (Ed.) Advances in Computers, vol. 45, pp. 269–320. Academic Press.

[Grudin 12] Grudin, J. & Poltrock, S. (2012). Taxonomy and Theory in Computer Supported Cooperative Work. The Oxford Handbook of Organizational Psychology, vol. 2. (Ed.) by Kozlowski, S.W.J.

[Guarino 98] Guarino, N. (1998). Formal Ontology in Information Systems. In: Guarino N (ed) 1st International Conference on Formal Ontology in Information Systems (FOIS'98). Trento, Italy. IOS Press, Amsterdam, pp 3–15.

[Guicking 05] Guicking, A., Tandler, P. & Avgeriou, P. (2005). Agilo: A highly flexible groupware framework in groupware: Design, Implementation, and Use: 11th International Workshop CRIWG, pp. 49-54, Springer-Verlag.

[Gutwin 02] Gutwin, C. & Greenberg, S. (2002). A descriptive framework of workspace awareness for real-time groupware. CSCW Journal, vol. 11, pp. 411–446.

[Gutwin 96] Gutwin, C., Greenberg, S. & Roseman, M. (1996). Workspace awareness in real-time distributed groupware: framework, widgets, and evaluation. In People and computers XI (Proceedings of the HCI'96).

[He 03] He, H. (2003). What is service-oriented architecture, [en línea], Xml.com.Acceso através de: http://www.xml.eom/pub/a/ws/2003/ 09/30/soa.html.

[Heffner 07] Heffner, R. (2007). Topic Overview: Service-oriented architecture. Forrester Research, Inc., Cambridge, MA - USA.

[Heradioa 13] Heradioa, R., Cabrerizo, F.J., Fernández-Amorós, D., Herrera, M. & Herrera-Viedmad, E. (2013). A. fuzzy linguistic model to evaluate the quality of Library 2.0 functionalities. International Journal of Information Management, vol. 33, pp. 642– 654.

[Hernández 16] Hernández, R. V. R., López, A., Juárez, C. M. & Mota. S. (2016). Evaluación de las características de un enfoque metodológico para desarrollar software en cursos universitarios. Revista FACPYA. UANL, vol 1, pp. 976-993.

[Herrera 96] Herrera, F., Herrera-Viedma, E. & Verdegay, J.L. (1996). Direct approach processes in group decision making using linguistic OWA operators. Fuzzy Sets and Systems 79, pp. 175–190.

[Herrera 09] Herrera, F., Alonso, S., Chiclana, F. & HerreraViedma, (2009). E. Computing with words in decision making: foundations, trends and prospects. Fuzzy Optimization and Decision Making, vol. 8, pp. 337–364.

[Herzog 15] Herzog, J. (2015). Software architecture in practice. Third Edition Written by Bass, L., Clements, P. & Kazman, R. ACM SIGSOFT Softw. Eng. Notes, vol. 40-1, pp. 51–52.

[Hofstede 10] ter Hofstede, A.H.M., van der Aalst, W., Adams, M. & Russell, M. Modern business process automation. YAWL and its support environment, 2010, Springer.

[Hofte 98] ter Hofte, G. H. (1998). Working Apart Together: Foundations for component groupware. PhD Thesis, Telematics Institute, Enschede, the Netherlands.

[Hollan 00] Hollan, J., Hutchins, E. & Kirsh, D. (2000). Distributed cognition: toward a new foundation for human-computer interaction research. ACM Transactions on Computer-Human Interaction (TOCHI) Special issue on HCI in the new millennium, vol. 7-2.

[Hom 00] Hom, J. (2000). The usability methods toolbox: Heuristic evaluation. Retrieved November 6, 2007, from http://jthom.best.vwh.net/usability

[Horrocks 99] Horrocks, I., Sattler, U. & Tobies, S. (1999).Practical reasoning for expressive description logics. In: Ganzinger H, McAllester D, Voronkov A (eds) 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99). Lecture Notes in Artificial Intelligence LNAI vol. 1705, pp 161–180, Springer-Verlag.

[Horrocks 00] Horrocks, I. (2001). A Denotational Semantics for OIL-Lite and Standard OIL. Technical report. http://www.cs.man.ac.uk/~horrocks/OIL/Semantics/

[Horrocks 01] Horrocks, I., van Harmelen, F. (2001). (eds) Reference Description of the DAML+OIL: Ontology Markup Language. Technical report. http://www.daml.org/2001/03/reference.html

[Horrocks 03] Horrocks, I., Patel-Schneider, P.F. & van Harmelen, F. (2003). From SHIQ and RDF to OWL: The making of a web ontology language. J. of Web Semantics, vol. 1-1, pp. 7-26.

[Howard 03] Howard, S. & Fingar, P. (2001). Business process management (BPM): The third wave, Meghan-Kiffer Pr.

[ISO 98] International Organization for Standardization, Technical Committee of Ergonomics. (1998). Ergonomic requirements for office work with visual display terminals (VDTs): Part 11: Guidance on usability (ISO No. 9241-11).

[Jaeger 96] Jaeger, T. & A.Prakash. (1996). Requirements of role-based access control for collaborative systems. In RBAC '95, pp. 16-36.

[Jasper 99] Jasper, R. & Uschold, M. (1999).A Framework for Understanding and Classifying Ontology Applications. In IJCAI Workshop on Ontologies and Problem-Solving Methods (KKR5),

[JBPM 10] JBPM (2010). https://www.jbpm.org/

[Jeners 12] N. & Prinz W. (2012) From groupware to social media - A comparison of conceptual models. In Camarinha-Matos L.M., Xu L., Afsarmanesh H. (eds) Collaborative Networks in the Internet of Services. PRO-VE 2012. IFIP Advances in Information and Communication Technology, vol. 380. Springer, Berlin, Heidelberg.

[Jeners 13] Jeners N., Prinz W. & Franken, S. (2013). A meta-model for cooperation systems. Luis M. Camarinha-Matos; Raimar J. Scherer. 14th Working Conference on Virtual Enterprises, (PROVE), Sep 2013, Dresden, Germany. Springer, IFIP Advances in Information and Communication Technology, AICT-408, pp. 239-246, Collaborative Systems for Reindustrialization.

[Juric 06] Juric, M.B. (2006). Business Process Execution Language for Web Services BPEL and BPEL4WS 2nd Edition. Packt Publishing.

[Kaplan 92] Kaplan, S.M. & Caroll, A.M. (1992).Supporting collaborative processes with conversation builder. Computer Communications, vol. 15-8, pp. 489-501.

[Karakostas 08] Karakostas, B. & Zorgios, Y. (2008). Engineering Service Oriented Systems: A Model Driven Approach, (Ed.) IGI Publishing.

[Kazman 94] Kazman, R., Bass, L., Abowd G. & Webb, M. (1994). Saam: A method for analyzing the properties of software architectures. Proc. of International Conference on Software Engineering, pp. 81-90.

[Kifer 95] Kifer, M., Lausen, G. & Wu, J. (1995). Logical Foundations of Object-Oriented andFrame-Based Languages. Journal of the ACM, vol. 42-4 pp. 741–843.

[Kohlhoff 03] Kohlhoff, C. & Steele, R. (2003). Evaluating SOAP for High Performance Business Applications: Real-Time Trading Systems. WWW2003, Budapest, Hungary.

[Kozaki 07] Kozaki, K., Sunagawa, E., Kitamura, Y. & Mizoguchi, R. (2007). A Framework for Cooperative Ontology Construction Based on Dependency Management of Modules. vol. 292 of CEUR Workshop Proceedings, pp. 33-44. CEUR-WS.org.

[Kuutti 91] Kuutti, K. (1991). The concept of activity as a basic unit of analysis for CSCW research. In proceedings of the Second European Conference on CSCW.

[Lampson 74] Lampson, B. (1974). Protection. In Princeton Symposium on Information Science and Systems, pp. 437-443. Reprinted in ACM Operatives Systems Rev, vol. 8-1, pp. 18-24.

[Laurillau 02] Laurillau, Y. & Nigay, L. (2002). Clover architecture for groupware. Proc. of the ACM Conference on CSCW, pp. 236-245.

[Lee 15] Lee, C.P. & Paine, D. (2015). From The Matrix to a Model of Coordinated Action (MoCA): A Conceptual Framework of and for CSCW. Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing, pp. 179-194.

[Lenat 90] Lenat, D.B. & Guha, R.V. (1990). Building Large Knowledge-based Systems: Representation and Inference in the Cyc Project. Addison-Wesley, Boston, Massachusetts.

[Liao 00] Liao, T. (2000). Light-weight Reliable Multicast Protocol. INRIA.

[Litiu 99] Litiu, R. & Prakash, A. (1999). DACIA: A Mobile Component Framework for Building Adaptive Distributed Applications. Technical Report CSE-TR-416-99, University of Michigan, EECS.

[Litiu 00] Litiu, R. & Prakash, A. (2000). Developing adaptive groupware systems using a mobile component framework. In Proceedings of the ACM 2000 Conference on Computer Supported Cooperative Work (CSCW'00), pp. 107-116.

[Louridas 08] Louridas, P. (2008). Orchestrating web services with bpel, IEEE Software, vol. 25, pp. 85–87.

[Maccagnan 10] Maccagnan, A., Riva, M., Feltrin, E., Simionati, B., Vardanega, T., Valle, G. & Cannata, N. (2010). Combining ontologies and workflows to design formal protocols for biological laboratories, Automated Experimentation, vol. 2-3.

[MacGrath 84] MacGrath, J.E. (1984). Groups: interaction and performance. Prentice-Hall, Englewood Cliffs.

[MacGregor 91] MacGregor, R. (1991). Inside the LOOM classifier. SIGART bulletin, vol. 2-3, pp, 70–76.

[Malone 94] Malone, T.W. & Crowston, K. (1994). The interdisciplinary study of coordination. ACM Computing Surveys vol. 26-1, pp. 87-119.

[Marinescu 02] Marinescu, D. (2002). Internet-based workflow management: Toward a semantic web. wiley, New York.

[Martin 05] Martin, R. (2005). Principles, Patterns, and Practices: The Factory Pattern.

[Martínez 05] Martínez-Acuña, M.I. (2005). Trabajo cooperativo asistido por computadora. Revista de divulgación científica y tecnológica de la Universidad Veracruzana, vol. XVIII-2.

[Martínez-Prieto 07a] Martínez-Prieto, M.A., De la Fuente, P. & Cuesta, C.E. (2007). AGORA: A layered architecture for cooperative work environments. In Proccedings of the Eighth Mexican International Conference on Current Trends in Computer Science (ENC 2007), pp. 73-80.

[Martínez-Prieto 07b] Martínez-Prieto, M.A., Cuesta, C.E. & De la Fuente, P. (2007). Aqueducts: A layered pipeline-based architecture for xml processing. In proccedings of the First European Conference on Software Architecture (ECSA'07).

[May 03] May, M. (2003). Business process management: Integration in a web-enabled environment, Prentice Hall FT.

[Medvidovic 00] Medvidovic, N. & Taylor, R. N. (2000). A classification and comparison framework for software architecture description languages. IEEE Transactions on Software Engineering, vol. 26-1, pp. 70-93.

[Mentzas 93] Mentzas, G. (1993). Coordination of joint tasks in organizational processes. Journal of Information Technology vol. 8, pp.139–150.

[Mikelakis 12] Mikelakis, M. & Papatheodorou, C. (2012). An ontology-based model for preservation workflows. In Proceedings of the 9th International Conference on Digital Preservation.

[Mittleman 08] Mittleman, D.D., Briggs, R.O., Murphy, J. & Davis, A. (2008). Toward a Taxonomy of Groupware Technologies. In: Briggs, R.O., Antunes, P., de Vreede, G.-J., Read, A.S. (Eds.) CRIWG 2008. LNCS, vol. 5411, pp. 305–317. Springer, Heidelberg.

[Mizoguchi 95] Mizoguchi, R., Vanwelkenhuysen, J. & Ikeda, M. (1995). Task ontology for reuse of problem solving knowledge. In: Mars N (Eds.) Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing (KBKS'95). pp. 46–57, IOS Press.

[Molina 06] Molina A.I., Redondo M.A. & Ortega M. (2006). A conceptual and methodological framework for modeling interactive groupware applications. In: Dimitriadis Y.A., Zigurs I., Gómez-Sánchez E. (eds) Groupware: Design, Implementation, and Use. CRIWG 2006. Lecture Notes in Computer Science, vol 4154, pp. 413-420, Springer-Verlag.

[Molina 07] Molina, A.I., Redondo, M.A., Ortega, M. & Hope, U. (2007). ClAM: A methodology for the development of groupware user interfaces. Journal of Universal Computer Science.

[Molina 09] Molina A.I., Redondo M.A. & Ortega M. (2009). A. Molina, M. Redondo, and M. Ortega, A review of notations for conceptual modeling of groupware systems. In New Trends on Human-Computer Interaction, Eds. J. Macías, A. Granollers, P. Latorre), pp. 1-12.

[Motta 98] Moota, E. (1998). An overview of the OCML modelling language. In Proceedings KEML, 8th Workshop on Knowledge Engineering, Methods, and Language.

[Nakagawa 06] Nakagawa, E.Y. (2006). Uma Contribuição ao Projeto Arquitetural de Ambientes de Engenharia de Software, Phd. Thesis, ICMC/USP, Universidade de São Carlos.

[Nielsen 93] Nielsen, J. (1993). Usability engineering. Cambridge, Mass.: Academic Pr.

[Nielsen 94] Nielsen, J. & Robert L. Mack, eds. (1994). Usability inspection methods. New York: Wiley.

[Nigay 91] Nigay, L. & Coutaz, J. (1991). Building user interfaces: Organizing software agents. In Proceedings ESPRIT'91 Conference, pp. 707-719.

[O'Leary 07] O'Leary, M.B. & Cummings, J.N. (2007). The spatial, temporal, and configurational characteristics of geographic dispersion in teams. MIS Quarterly, vol. 31-3, pp. 433–452.

[OCML 99] OCML: Operacional Conceptual Modelling Language Page", (1999) en: http://kmi.open.ac.uk/projects/ocml/.

[OKBC 95] Open Knowledge Base Connectivity Working http://www.ai.sri.com/~okbc/. (1995).

[OMG 01] The Object Management Group http://www.omg.org (2001).

[Orozco 04] Orozco, P. (2004). A decoupled architecture for action-oriented coordination and awareness management in CSCL/W frameworks; 10th International Workshop, CRIWG, Springer-Verlag, pp. 246-261.

[Ousterhout 94] Ousterhout, J. (1994). Tcl and the Tk Toolkit. Addison-Wesley.

[Ouyang 04] Ouyang, Y., Tang, M., Lin, J. & Jin-xiang, D. (2004). Distributed collaborative CAD system based on Web Service. Journal of Zhejiang University SCIENCE, pp. 1009-3095.

[Paderewski 03] Paderewski, P. (2007). Tesis doctoral: Un modelo arquitectónico evolutivo para sistemas software basados en agentes. Universidad de Granada.

[Pan 04] Pan, Y., Duanqing, X., Chun, C. & Ying Z. (2004). Using Web services implementing collaborative design for CAD systems. In Proceedings of the 2004 IEEE International Conference on Services Computing (SCC'04).

[Papazoglou 06] Papazoglou, M., Traverso, P., Dustdar, S. & Leymann, F. (2006). Service-oriented computing. Research Roadmap, Accesible en: http://drops.dagstuhl.de/opus/volltexte/2006/524/.

[Papazoglou 08] Papazoglou, M. (2008). Web service: principle and technology. (Ed.) Pearson Prentice Hall.

[Parasuraman 88] Parasuraman, A., Zeithaml, V.A. & Berry, L.L. (1988). SERVQUAL: A multiple-item scale for measuring customer perceptions of service quality. Journal of Retailing, vol. 64, pp.12-40.

[Patterson 94] Patterson J. F. (1994). A. taxonomy of architectures for synchronous groupware applications. In Proceedings of the Workshop on Software Architectures for Cooperative Systems.

[Penichet 07a] Penichet, V.M.R. (2007). Modelo de proceso para el desarrollo de interfaces en entornos CSCW centrado en los usuarios y dirigido por tareas. Tesis Doctoral. Universidad de Castilla-La Mancha, Departamento de Sistemas Informáticos.

[Penichet 07b] Penichet, V.M.R., Marin, I., Gallud, J.A., Lozano, M.D. & Tesoriero, R. (2007). A classification method for CSCW systems. Electronic Notes in Theoretical Computer Science, vol. 168, pp. 237–247.

[Penichet 08] Penichet, V.M.R., Lozano, M.D. & Gallud. J.A. (2008). An Ontology to Model Collaborative Organizational Structures in CSCW Systems. In Engineering the User Interface, Springer, pp.127-139.

[Pfaff 85] Pfaff, G. (1985). User Interface Management Systems. Eurographics Seminars. Springer Verlag.

[Phillips 99] Phillips, W.G. (1999). Architectures for synchronous groupware. Technical Report 1999-425, Queen's University.

[Preece 00] Preece, J. (2000). Online communities: Designing usability and supporting sociability. John Wiley & Sons, Ltd, England.

[Prinz 99] Prinz,W. (1999). NESSIE: An awareness environment for cooperative settings. Proc. of the European Conference of CSCW, pp. 391-410.

[Pumareja 02] Pumareja, D.T. & Sikkel, K. (2002). An evolutionary approach to groupware implementation: the context of requirements engineering in the socio-technical frame. Technical Report TRCTIT-02-30, Centre for Telematics and Informational Technology, University of Twente, Enschede.

[Qiu 03] Qiu X., Carpenter B. & Fox G. (2003). Collaborative Web services and the W3C document object model. Technical Report. Community Grids Lab.

[Rajan 95] Rajan, S., Venkat, R.P. & Vin, H.M. (1995). A formal basis for structured multimedia collaborations. In Proceedings of the 2nd IEEE International Conference on Multimedia Computing and Systems, pp. 194-201.

[RDF 10] RDF and OWL Compatibility. W3C Recommendation, 22 June 2010. Available at http://www.w3.org/TR/rif-rdf-owl/

[Reagle 01] Reagle, J. (2001). XML Encryption Requirements. http://www.w3.org/TR/2001/WD-xml-encryption-req-20010420.

[Rodríguez 12] Rodríguez, D. & García-Martínez, R. (2012). Modeling the interactions in virtual spaces oriented to collaborative work. Chapter 10, Eds., C. M. Zapata, G. González, R. Manjarrés, F. B. Vargas, and W. Arévalo, Software Engineering: Methods, Modeling, and Teaching, vol. 1, Lima, Perú.

[Roseman 92] Roseman, M. & Greenberg, S. (1992). GroupKit: A groupware toolkit for building real-time conferencing applications. In Proceedings of the ACM CSCW'92 Conference on Computer Supported Cooperative Work, pp. 43-50.

[Roseman 94] Roseman, M. & Greenberg, S. (1994). Registration for real time groupware. Research Report 94/533/02, Department of Computer Science, University of Calgary, Alberta, Canada.

[Roseman 96a] Roseman, M. & S. Greenberg. (1996). Building real-time groupware with GroupKit. A groupware toolkit. ACM Trans. Computer-Human-Interaction, vol. 3, pp. 66-106.

[Roseman 96b] Roseman, M, & Greenberg, S. (1996). TeamRooms: Network places for collaboration. In Proc. of ACM CSCW '96.

[Roth 00] Roth, J. & Unger, C. (2000). Developing synchronous collaborative applications with TeamComponents. In Designing Cooperative Systems:the Use of Theories and Models, Proceedings of the 5th International Conference on the Design of Cooperative Systems (COOP'00), pp. 353-368.

[Russel 95] Russel, S. & P. Norvig. (1995). Artificial Intelligence – A Modern Approach. Prentice-Hall.

[Samarati 00] Samarati, P. & di Vimercati, S.D.C. (2000). Access control: Policies, models, and mechanisms. In FOSAD, pp. 137–196.

[Sanchez-Galvez 15] Sanchez-Galvez, L.A. & Fernandez-Luna, J.M. (2015). A Usability Evaluation Methodology of Digital Library. The Seventh International Conference on Information, Process, and Knowledge Management (eKnow 2015), pp. 23-28.

[Sandhu 96] Sandhu, R S., Coyne, E.J., Feinstein, H.L. & Youman, C. E. (1996). Role-based access control models. Computer, vol. 29-2, pp- 8–47.

[Schmidt 90] Schmidt, K. (1990). Analysis of cooperative work – a conceptual framework. Risø National Laboratory, DK-4000 Roskilde, Denmark (Risø-M-2890).

[Schmidt 00] Schmidt, D.C. (2000). Pattern-oriented software architecture. Patterns for concurrent and distributed objects, vol. 2, John Wiley & Sons Ltd.

[Schmidt 13] Schmidt, K. & Bannon, L. (2013). Constructing CSCW: The first quarter century. Computer Supported Cooperative Work (CSCW), vol. 22, 4-6, pp.345-372.

[SchmidtD 13] Schmidt, D.C., Stal, M. Rohnert, H. & Buschmann, H. (2013). Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects, vol. 2. John Wiley & Sons.

[Schuckmann 96] Schuckmann, C. (1996). Designing object-oriented synchronous groupware with COAST. In CSCW, pp. 30-38, ACM Press.

[Schulzr 96] Schulzrinne, H. (1996). Audio-video transport working group. Standards track network working group.

[Sebastian 08] Sebastian, A., Noy, N.F., Tudorache, T. & Musen, M.A. (2008). A generic ontology for collaborative ontology-development workflows. Proceedings of the 16th international conference on Knowledge Engineering: Practice and Patterns.

[Shadbolt 93] Shadbolt, N., Motta, E. & Rouge. A. (1993). Constructing knowledge-based systems. IEEE Software, vol. 10-6, pp. 34–39.

[Shaohua 03] Shaohua, L., Jun, W. & Wei, X. (2003). Towards dynamic process with variable structure by reflection. In Proceeding of the 27th Annual International Computer Software and Applications Conference (COMPSAC2003).

[Shaohua 04] Shaohua, L., Jun, W., Yinglong, M. & Yu, L. (2004). Web service cooperation ideology. In proccedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI 2004), 20-24.

[Shaw 95] Shaw, M. & Garlan, D. (1995). Formulations and Formalisms in Software Architecture. In Computer Science Today, vol. 1000 Lecture Notes in Computer Science. Springer-Verlag.

[Shaw 96] Shaw, M. & Garlan, D. (1996). Software architecture. Perspectives and emergencing discipline. Prentice Hall.

[Shen 92] Shen, H. & Dewan, P. (1992). Access control for collaborative environments. In Proceedings of the ACM Conference on CSCW, pp. 51-58.

[Slagter 01] Slagter, R. & ter Doest, H. (2001). The CoCoWare component architecture. GigaCSCW deliverable (D2.1.4).

[Sloman 94] Sloman, M. (1994). Policy driven management for distributed systems J. Network System Management, vol. 2-3, pp. 33-60.

[Smith 13] Smith, J. (2013). Advanced MVVM. Kinddle.

[Sommerville 15] Sommerville, I. (2015). Software engineering. 10th Edition. Pearson.

[Sosa 06] Sosa, M., Zarco, R. & Postiglioni, A. (2006). Modelando aspectos de grupo en entornos colaborativos para proyectos de investigación, Revista de Informática Educativa y Medios Audiovisuales, vol. 3, pp. 22–31.

[Sowa 92] Sowa, J. F. & Zachman, J. A. (1992). Extending and formalizing the framework for information systems architecture, IBM System Journal, vol. 31-3.

[Stefik 87] Stefik, M., Bobrow, D. G., Foster, G., Lanning, S. & Tatar., D. (1987). WYSIWIS revised: Early experiences with multiuser interfaces, ACM Transactions on Office Information Systems, vol. 5, No. 2, pp. 147-167.

[Stiemerling 99a] Stiemerling, O., Hinken, R. & Cremers, A.B. (1999). Distributed component-based tailorability for CSCW applications. In Proceedings of the 4th International Symposium on Autonomous Decentralized Systems (ISADS '99), pp. 345-352.

[Stiemerling 00] Stiemerling, O. (2000). Component-based tailorability. PhD thesis, University of Bonn, Bonn, Germany.

[Studer 98] Studer, R., Benjamins, R. & Fensel, D. (1998). Knowledge engineering: Principles and methods. Data & Knowledge Engineering, vol. 25(1–2), pp. 161–198.

[Suri 03] Suri, N. & Uszok, A. (2003). Semantic web languages for policy representation and reasoning: A comparison of KAoS, Rei, and Ponder. Springer-Verlag, LNCS vol. 2870, pp. 419-437.

[Swenson 04] Swenson, K.D. (2004). ASAP/Wf-XML 2.0 Cookbook. Fujitsu Software Corporation.

[Szyperski 02] Szyperski, C.A. (2002). Component software: Beyond Object-Oriented programming. Reading, Addison-Wesley.

[Tam 06] Tam, J. & Greenberg, S. (2006). A framework for asynchronous change awareness in collaborative documents and workspaces. International Journal of Man–Machine Studies, vol. 64-7, pp. 583–598.

[Terai 03] Terai, K., Izumi, M. & Yamaguchi, T. (2003). Coordinating Web Services based on Business Models, In Proc. of the 5th international Conference on Electronic Commerce, pp. 473-478.

[Tolone 05] Tolone, W., Ahn, G.J., Pai, T. & Hong, S.P. (2005). Access control in collaborative systems. ACM Comput. Surv, vol. 37, pp. 29–41.

[UDDI 15] UDDI Consortium. http://uddi.xml.org/. 2015.

[Uhrmach 01] Uhrmacher A. M. (2001). Dynamic structures in modeling and simulation: A reflective approach. ACM Transactions on Modeling and Computer Simulation, vol. 11-22, pp. 206–232.

[Uschold 96] Uschold, M. & Grüninger, M. (1996). Ontologies: Principles, methods and applications. Knowledge Engineering Review, vol. 11-2, pp. 93-155.

[Uschold 04] Uschold, M. & Gruninger, M. (2004). Ontologies and semantics for seamless connectivity. In SIGMOD Record, vol. 33-4, pp. 58-64.

[Van Welie 98] Van Welie, M., van der Veer, G.C. & Eliëns, A. (1998). An Ontology for Task World Models, Design, Specification & Verification of Interactive System, Springer Computer Science, pp. 57-70.

[Venkat 91] Venkat, R.P. & Vin, H.M. (1991). Multimedia conferencing as a universal paradigm for collaboration. In L. Kjelldahl (Ed.), Multimedia: Systems, Interaction and Application, 1st Eurographics Workshop, Springer-Verlag, pp. 173-185.

[Verginadis 10] Verginadis, Y., Papageorgiou, N., Apostolou, D. & Mentzas, G. (2010). A review of patterns in collaborative work. In: Proceedings of the 16th ACM International Conference on Supporting Group Work (GROUP 2010), pp. 283–292. ACM.

[Vieira 05] Vieira, V., Tedesco, P. & Salgado, A. C. (2005). Towards an ontology for context representation in groupware. In H. Fuks, S. Lukosch & A. C. Salgado (Eds.), CRIWG 2005. LNCS, vol. 3706, pp. 367–375). Springer.

[W3C 16] W3C consortium. www.w3c.org.

[Wang13] Wang, M. (2013). Academic Library, e-Science/e-Research, and Data Services in a Broader Context. Indianapolis.

[Weiseth 06] Weiseth, P.E., Munkvold, B.E., Tvedte, B. & Larsen, S. (2006). The wheel of collaboration tools: A typology for analysis within a holistic framework. In Proceedings of the ACM Conference on Computer-Supported Cooperative Work (CSCW 2006), pp. 239–248.

[Weske 07] Weske, M. (2007). Business process management: Concepts, languages, architectures, Springer-Verlag.

[Whittle 16] Whitle, R. & Myrick, C. B. (2016). Enterprise business architecture: The formal link between strategy and results. CRC Press.

[WMC 13] Workflow Management Coalition http://www.wfmc.org/ 2013

[Woerner 01] Woerner, J. & Laengle, T. (2001). Co-operative work in plant production processes. In proccedings VIDOP-Workshop, pp. 26-30.

[Woerner 02a] Woerner J. & Woern H. (2002). Distributed and secure co-operative engineering in virtual plant production. In Proceedings of the International Conference on Advanced Production Management Systems: Collaborative Systems for Production Management.

[Woerner 02b] Woerner, J., Laengle, T. & Woern, H. (2002). Corporate planning and simulation of plant production facilities in the virtual world. In proceedings of the 18th International Conference on CAD/CAM, Robotics and Factories of the Future, 109-116, 2002

[Woerner 02c] Woern, H., Frey, D. & Keitel, J. (2000). Digital factory -planning and running enterprises of the future. In proceedings of the 26th Annual Conference of the IEEE Electronics Society, 1286-1291, IEEE Press.

[Wulf 93] Wulf, W.A. (1993). The collaboratory opportunity. Science261, vol. 5123, pp. 854–855.

[XPDL 02] XPDL, Workflow process definition interface-xml process definition language, (2002). WFMC (Workflow Management Coalition), WFMCTC-1025.

[Yager 88] Yager, R.R. On ordered weighted averaging aggregation operators in multicriteria decision making. IEEE Transactions on Systems, Man and Cybernetics 18, pp. 183–190, 1988.

[Zadeh 65] Zadeh, L.A. (1965). Fuzzy sets. Information and Control, pp. 338–353.

[Zadeh 75] Zadeh, L.A. (1975). The concept of a linguistic variable and its applications to approximate reasoning. Part III. Information Sciences, 9(1), pp. 43–80.

[Zeigler 00] Zeigler B.P. (2000). Theory of modeling and simulation. Academic Press Ltd.

[Zeithaml 90] Zeithaml, V.A., Berry, L.L., Parasuraman, A. Delivering Quality Services - Balancing Customer Perceptions and Expectations. The Free Press, New York, 1990.

[Zhu 09] Zhu, J., Tang, Y., Li, J. & Liu, H. (2009). A role hierarchical group awareness model and its time possibility analysis based fuzzy time. In Proceedings of the 2009 13th international conference on computer supported cooperative work in design. pp. 38–43.

[Zimmermann 04] Zimmermann, O. & Krogdahl, P. (2004). Service-Oriented Analysis and Design. First thoughts on an emerging discipline. In proceedings on the OOSPLA.

Acrónimos

ADT	*Abstract Data Types*
AEE	*Action Execution Engine*
AM	*Awareness Manager*
AMENITIES:	*A MEthodology for aNalysis and desIgn of cooperaTIve systEmS*
ANS	*Action Notification Service*
AORTA	*Action-oriented decOupled aRchitecture for coordinaTion and Awareness*
API	*Application Programing Interface*
BPEL	*Business Process Execution Language*
BPMI	*Business Process Management Initiative*
BPMN	*Business Process Modelling Notación*
BPMS	*Business Process Modelling Systems*
CAD	*Computer-Aided Design*
CAT	*Component Architecture for Tailorability*
CBSE	*Component-Based Software Engineering*
CIAM:	*Collaborative Interactive Applications Methodology*
CIAN	*Collaborative Interactive Applications Notation*
CM	*Coordination Manager*
COAST:	*COoperative Application Systems Toolkit*
CoCoWare:	*Collaborative Component software*
CooPS	*People and Systems Cooperative*
CORBA	*Common Object Request Broker Architecture*
CPN	*Color Petri Nets*
CSCL/W	*Computer Supported Colaborative Learning/Work*
CSCW	*Computer Supported Collaborative Work*
CSCW:	*Computer Supported Cooperative Work*
CTT	*Concur Task Tree*
DACIA:	*Dynamic Adjustment of Component InterActions*
DAML	*DARPA Agent Markup Language*
DARPA	*Defence Advanced Research Proyects Agency*
DB	*Data Base*
DBMS	*Data Base Management System*
DEC:	*Digital Equipment Corporation*
DLP	*Description Logic Programs*
F-Logic	*Frame Logic*
FO	*Frame Ontology*
FOL	*First Order Logic*
GDSS	*Group Decision Support Systems*
GFP	*Generic Frame Protocol*
GUI	*Graphic User Interface*
HCI	*Human-Computer Interaction*
HHMP	*HendHeld Message Protocol*
HTTP	*HyperText Transfer Protocol*
HTTP	*HyperText Transfer Protocol*
IMI	*Incoming Message Indicator*
IST	*Information Society Technologies*
jBPM	java *Business Process Management*
JMS	*Java Message Service*
JSDT:	*Java Shared Data Toolkit*
KIF	*Knowledge Interchange Format*
KIF	*Knowledge Interchange Format*
KSL	*Knowledge Systems Lab*
LAN	*Local Area Network*
LD	*Description Logics*
LRMP	*Lightweight Reliable Multicast Protocol*
MIT:	*Massachussets Institute of Technology*
MoCA	*Model of Coordinated Action*
MOLIC	*Modeling Language for Interaction as Conversation*
MOLLICC	*Modeling Language for Interaction as Conversation, Collaborative*
MOM	*Message-Oriented Middleware*

MOM	*Message-Oriented Middleware*
MVC	*Model-View-Controller*
MVP	*Model-View-Presenter*
MVVM	*Model-View-ViewModel*
NAT	Non Atomic Terms
NCoP	*Number of Communities of Practice*
OCML	*Operational Conceptual Modeling Language*
OIL	*Ontology Inference Layer*
OKBC Protocol	*Open Knowledge Base Connectivity Protocol*
OMG	*Object Management Group*
OMI	*Outgoing Message Indicator*
ORB	*Object Request Broker*
OSI	*Open System Interconnection*
OWL	*Web Ontology Language*
PAC	*Presentation-Abstraction-Control*
PROC	*Processing and ROuting Components*
ProSA	*Process based on Software Architecture*
RDF	*Resource Description Framework*
RMI	*Remote Method Invocation*
RPC	*Remote Procedure Call*
SAGA	*Web Services Architecture for Groupware Applications*
SMTP	*Simple Mail Transfer Protocol*
SOA	*Service-Oriented Architecture*
SOAP	*Simple Object Access Protocol*
SQL	*Structured Query Language*
SSI	*Shared Space Indicator*
SSL	*Secure Sockets Layer*
TCL	*Tool Command Language*
TCP/IP	*Transmission Control Protocol/Internet Protocol*
TOUCHE:	*Task-Oriented and User Centered process model for developing interfaces for human-Computer-Human Environments*
UDDI	*Universal Description, Discovery and Integration*
URI	*Uniform Resource Identifier*
W3C	*World Wide Web Consortium*
WfMC	*Workflow Management Coalition*
WfMS	*Workflow Management Systems*
Wf-XML	*Workflow-eXtend Markup Language*
WSDL	*Web Services Description Language*
XML	*eXtensible Markup Language*
XPDL	*XML Process Definition Language*
YAWL	*Yet Another Workflow Language*