Universidad Nacional de La Plata
Facultad de Informática
Laboratorio Lifia

Modelado de Sistemas Colaborativos

Luis Mariano Bibbo

Directoras: Dra. Claudia Pons y Dra. Roxana Giandini

Co-Director: Dr. Gustavo H. Rossi

Enviado como requerimiento para el grado de:
Doctor en Ciencias Informáticas de la UNLP
14 de octubre de 2021

Abstract

Building Collaborative systems with awareness (or groupware) is a very complex task. This thesis presents the design and implementation of the domain specific language CSSL v2.0 - Collaborative Software System Language -built as an extension of UML, using the metamodeling mechanism. CSSL provides simplicity, expressiveness and precision to model the main concepts of collaborative systems, especially collaborative processes, protocols and awareness. The CSSL concrete syntax is defined via a set of editors through which collaborative systems models are created. According to the MDD methodology, models are independent of the implementation platform and are formally prepared to be transformed. The target of the transformation is a web application based on JavaScript, MongoDB and Websockets technology, that provides a set of basic functions that developers can refine to complete the development of the collaborative system. Finally, evaluation, validation and verification of the language is performed, determining that the CSSL tools allow developers to solve central aspects of collaborative systems implementation in a simple and reasonable way, representing an advance to the current state of these technologies.

Resumen

La construcción de sistemas colaborativos con awareness es una tarea muy compleja. En este trabajo se define el lenguaje específico de dominio CSSL v2.0 - Collaborative Software System Language - construido como extensión de UML, usando el mecanismo de metamodelado. Se analiza la simplicidad, expresividad y precisión del lenguaje para modelar los conceptos principales de estos sistemas, especialmente los procesos colaborativos, protocolos y awareness. A partir de casos de modelado se muestra una sintaxis concreta implementada a través de editores gráficos que permiten construir modelos de sistemas colaborativos. Estos son independientes de la plataforma de implementación y están formalmente preparados para ser derivados en implementaciones concretas aplicando transformaciones utilizando el paradigma MDD (Model Driven Development). Las transformaciones de modelo a texto generan versiones Web implementadas con tecnologías JavaScript, MongoDB y Websockets que brindan un conjunto de funcionalidad básica que los desarrolladores pueden refinar para completar la implementación de los sistemas colaborativos con awareness. Finalmente se realiza una evaluación, validación y verificación del metamodelo que finalizan con la comprobación que comprueban que el lenguaje

CSSL v2.0, permite definir de forma precisa, concisa y amigable los conceptos abstractos de los sistemas colaborativos, incluyendo los procesos colaborativos, protocolos y awareness, lo cual representa un avance al estado actual de estas tecnologías.

Agradecimientos

A mi padres, hermanos y a mis amores Luján, Paloma y Emilio.

También agradezco a:

- Mis directoras Claudia Pons y Roxana Giandini. Por su apoyo y colaboración en este camino.
- Mis compañeros del laboratorio LIFIA.
- Mis amigos.

Índice general

Índice de cuadros

Índice de figuras

Capítulo 1

Introducción

1.1. Motivación

Los sistemas colaborativos permiten que un conjunto de usuarios se comuniquen, compartan información y coordinen actividades. De acuerdo con Ellis et all [1], las plataformas de colaboración son: "Sistemas de computadoras que proveen una interfaz a un entorno compartido y que soportan a un grupo de usuarios que tienen un objetivo común". La tecnología que soporta este tipo de sistemas fue presentada y comentada en los trabajos iniciales de Ellis [1], Grudin [2, 3] y resumidos en la tesis de G. Henri Ter Hofte y D. H. J. Van Der Lugt [4] donde aparece la denominación "groupware", "sistemas groupware" o "sistemas colaborativos".

Estos sistemas no solo contemplan a un usuario en particular, sino que atienden las necesidades de un grupo. Se caracterizan por controlar tres conceptos principales. Por un lado, requieren herramientas de **comunicación** entre los participantes, ya que los usuarios pueden estar en diferentes sitios y necesitan comunicarse. Por otro soportan la **colaboración**, que implica que el sistema mantiene un conjunto de datos compartidos y que los usuarios pueden modificar. Finalmente, la **coordinación**, donde se define el orden en que los usuarios participan dentro de las actividades colaborativas. Una efectiva colaboración se mejora cuando la participación de los usuarios está coordinada. Por ejemplo, para llegar a un objetivo grupal, los usuarios deberán pasar por distintas actividades y el sistema organiza el orden en que dichas tareas se llevan a cabo. Esto se logra a través de la especificación de procesos colaborativos. Además, cómo se trató en el trabajo de Briggs et all. [5], la interacción de los participantes dentro de

una actividad puede ser diseñada usando los protocolos que describen las operaciones concretas que pueden realizar los roles en cada estado de una actividad.

Por otro lado, para lograr una efectiva colaboración, los usuarios tienen que estar informados sobre las acciones que los otros participantes realizan en el ambiente y cómo esas acciones afectan el entorno de trabajo. Esa información que brinda el sistema se llama: "**awareness**". El Awareness según Dourish [6] es "el conocimiento de las actividades de otros que provee contexto para tus propias actividades". De esta forma los usuarios pueden coordinar su trabajo basado en el conocimiento de lo que otros hacen o han hecho en el espacio compartido. Un ejemplo conocido es el caso de los sistemas de mensajería que muestran cuando un usuario está escribiendo o grabando audio. Esta información de awareness funciona como una herramienta de coordinación, ya que los otros usuarios pueden esperar a que el colaborador finalice su mensaje. En otros casos, podrán ver por ejemplo, los cambios en los documentos o el grado de avance de alguna tarea, como se explicó en los trabajos de Gutwin [7, 8].

El awareness también estimula la colaboración espontánea, ya que, cuando un usuario ve a otros en el sistema, sabe dónde están y que acción están realizando, puede motivarlo a comunicarse o interactuar con los otros. Es decir que los usuarios no están solos, y el awareness funciona como un disparador para colaborar con otros en el sistema, como se documentó en la experiencia realizada por Dourish [9]. En otras situaciones como juegos, se puede ver la ubicación de otros usuarios lo que permite enfrentarlos o interactuar de otras maneras con ellos.

Entre los awareness más comunes aparecen: presencia, ubicación, densidad, datos del usuario (Edad, Nacionalidad, etc.), nivel de actividad, acciones, lugares donde estuvo, lugares donde realizó las acciones, cambios que realizó, objetos que controla, objetos que puede alcanzar, información que puede ver, intensiones, habilidades, influencia y datos históricos sobre los anteriores. La información de awareness también puede relacionarse, agrupándola para mejorar la calidad de esta. En general cuando se muestra la presencia del usuario se le puede agregar otro tipo de información como el estado, la ubicación o qué recursos o habilidades tiene.

El desarrollo de Sistemas Colaborativos con Awareness tiene una gran complejidad, dado que atienden a las necesidades de un grupo de usuarios interactuando en un entorno compartido. Es necesario contar con recursos de ingeniería de software que faciliten y organicen su descripción y desarrollo. En especial, en este trabajo, se pretende desarrollar modelos donde se describan aspectos de coordinación en alto nivel, a través de "procesos colaborativos" y de bajo nivel con

"protocolos". En esta línea, se intenta modelar también un aspecto específico de los sistemas colaborativos, que es el awareness, que dinámicamente informa el estado de la colaboración a todos los participantes.

Desde el punto de vista de la ingeniería de software, es necesario comentar acerca de la metodología de construcción de software de calidad y que pueda ser capaz de sobrevivir a la evolución de sus requisitos funcionales, manteniéndose flexible a los cambios en la tecnología que lo sustenta, y que es actualmente contemplado en los principios del paradigma de desarrollo de software conocido como MDD (por sus siglas en inglés: Model Driven software Development) que ofrece mejorar los procesos de construcción de software [10].

Una propuesta concreta utilizada en el ámbito MDD es la idea de crear modelos para un dominio específico a través de lenguajes DSLs (por su nombre en inglés: Domain-Specific Language), focalizado y especializado para dicho dominio. Estos lenguajes permiten especificar la solución usando directamente conceptos del dominio del problema. Los productos finales son luego generados automáticamente desde estas especificaciones de alto nivel [11, 12].

1.2. Objetivos

El objetivo de esta tesis es contribuir con recursos de ingeniería de software que guíen la construcción de sistemas colaborativos complejos y que faciliten su desarrollo, desde la especificación hasta la implementación en alguna plataforma particular. En el contexto del paradigma MDD, se propone trabajar con un lenguaje específico de dominio, con una sintaxis concreta y una semántica precisa, que permita especificar, diseñar y llegar a obtener versiones ejecutables de sistemas colaborativos. El lenguaje debe permitir especificar los elementos que componen el sistema, los procesos colaborativos, protocolos y awareness.

Las herramientas de diseño deben sustentar prácticas repetibles, generando recursos reusables y técnicas que organizan el desarrollo y mejoren la calidad de los productos. En pos de lograr este objetivo, en esta tesis se diseña e implementa el lenguaje específico de dominio CSSL (Collaborative Software System Language) [13–15] como una extensión de UML 2.0 [16]. Se desarrollan un conjunto de editores gráficos que permitan modelar los sistemas colaborativos, utilizando una sintaxis concreta que facilite la comunicación entre los diseñadores. A partir de

los modelos, se propone desarrolla un conjunto de herramientas de transformación, que toman los modelos colaborativos y automáticamente obtienen implementaciones concretas. Sin mucho esfuerzo, los diseñadores podrán ver en ejecución los modelos construidos con los editores.

1.3. Publicaciones

Las últimas publicaciones del autor sobre el tema de la tesis son:

- L. M. Bibbo, C. Pons, and R. Giandini, "Model Driven Development of Groupware Systems" Aprobado para ser publicado en International Journal of e-Collaboration (IJeC): Volume 18, Issue 3, Article 10. (Enero-2022)

- L. M. Bibbo, R. Giandini, and C. Pons, "Modelado de Derivación de Código para el Desarrollo de Sistemas Colaborativos con Awareness" Electronic Journal SADIO Vol. 19 Núm. 2 (2020)

- L. M. Bibbo, R. Giandini, and C. Pons, "Modelado y Derivación de Código" XLVIII Jornadas Argentinas de Informática e Investigación Operativa (48 JAIIO) - Simposio Argentino de Ingeniería de Software (ASSE 2019)

- L. M. Bibbo, R. Giandini, and C. Pons, "DSL for Collaborative Systems with Awareness" SLISW - Simposio Latinoamericano de Ingeniería de Software; XLIII CLEI - Conferencia Latinoamericana de Informática, 2017.

- L. M. Bibbo, R. Giandini, and C. Pons, "Sistemas Colaborativos con Awareness: Requisitos para su Modelado" XLV Jornadas Argentinas de Informática e Investigación Operativa (45 JAIIO) - Simposio Argentino de Ingeniería de Software (ASSE 2016), no. Ciencias Informáticas, pp. 111–122, 2016.

- L. M. Bibbo, D. García, and C. Pons, "Domain Specific Language for the Development of Collaborative Systems" 2008 International Conference of the Chilean Computer Science Society, pp. 3–12, 11 2008.

1.4. Estructura de la Tesis

Para lograr los objetivos planteados en este trabajo, en el capítulo 2 se presenta el marco teórico que está dividido en dos temas: en primera instancia, se discute el paradigma de MDD (por sus siglas en inglés: Model Driven software Development), focalizando los objetivos de mejora de la productividad, portabilidad, interoperabilidad y calidad del software a desarrollar; en segunda instancia, se tratan los sistemas colaborativos con awareness, donde se comentan las definiciones generales, se describen los procesos colaborativos, protocolos y se detallan las características del awareness. Sobre el final del capítulo se destacan la gran variedad de sistemas colaborativos que existen, cómo se clasifican; también se analizan las ventajas y desventajas de cada una de sus variantes.

En el capítulo 3, se revisan distintas herramientas de ingeniería de software (modelos conceptuales, frameworks, meta-modelos, etc.), a partir de trabajos de investigación publicados, que sirvan para describir los sistemas colaborativos con awareness. Como resultado de esta revisión, se presenta un glosario de conceptos y se plantea un conjunto de requisitos para un modelo conceptual.

Luego en el capítulo 4, se presenta el lenguaje CSSL para modelar sistemas colaborativos con awareness, cuya sintaxis es formalizada mediante un metamodelo que lo describe. El metamodelo incluye tres sub-modelos: el modelo conceptual, que representa los conceptos principales; luego el sub-modelo dinámico, donde se describen los procesos colaborativos y protocolos; finalmente el sub-modelo de awareness, que permite asociar el awareness a los distintos elementos del sistema.

En el capítulo 5, se presentan los editores gráficos del lenguaje CSSL, que implementan una sintaxis concreta para describir los sistemas colaborativos. Los editores permiten diagramar distintos aspectos del sistema y relacionarlos entre sí. Entre los conceptos destacados que se pueden modelar se encuentran: la estructura del sistema colaborativo, los procesos colaborativos, protocolos de cada actividad y asociar el awareness a los elementos del sistema.

A continuación, en el capítulo 6, se define una semántica del lenguaje CSSL. La semántica, se expresa a través de una herramienta de transformación, que toma los modelos escritos en lenguaje CSSL y los transforma en una implementación concreta. Esto sirve para prototipar los modelos colaborativos elaborados y comprobar si responden a los propósitos planteados para

estos modelos.

En el capítulo 7 se realiza una evaluación, validación y verificación del lenguaje CSSL. Con la evaluación se pretende determinar cuán configurable es, su complejidad y si depende de la semántica de UML. Luego con la validación se analiza que el modelo conceptual sea correcto y finalmente la verificación observa la posibilidades de llevarlo a una implementación concreta.

Finalmente, en el último capítulo, se presentan las conclusiones que se obtienen de este trabajo y las perspectivas de trabajos a futuro.

Capítulo 2

Marco Teórico

En este capítulo se presenta los conceptos teóricos que sustentan esta tesis. En particular se desarrollan los fundamentos de los sistemas colaborativos y se discuten paradigmas de desarrollo de software apropiados para construir estos sistemas.

2.1. Sistemas Colaborativos con Awareness

2.1.1. Introducción

Cuando se habla de sistemas colaborativos, se piensa en tecnologías basadas en redes de computadoras que permite que los usuarios se comuniquen, compartan información y coordinen actividades. Estos sistemas no contemplan a un usuario en particular sino que atiende las necesidades del grupo. De acuerdo a Ellis et al. [1], las plataformas de colaboración son:

Definición:

Sistemas basados en computadoras que soportan grupos de personas involucradas en una tarea común (u objetivo) y que proveen de una interfaz a un ambiente compartido.

Estas tecnologías brindan un equilibrio, entre el trabajo individual de los participantes y la colaboración que realizan los usuarios para lograr un objetivo grupal. Los conceptos de este tipo de sistemas fueron presentados y comentados en los trabajos iniciales de Ellis [1], Grudin [2, 3]

y resumidos en la tesis de G. Henri Ter Hofte y D. H. J. Van Der Lugt [4] donde aparece la denominación “groupware”, “sistemas groupware” o “sistemas colaborativos”. Los tres aspectos centrales que intervienen en los sistemas colaborativos son:

- La **comunicación**: permite a los participantes intercambiar información. El principal desafío de esta perspectiva es cómo hacer que la comunicación entre usuarios distribuidos sea tan efectiva como la comunicación cara a cara.
- La **colaboración**: implica mantener un conjunto de datos compartidos y son el resultado de las actividades colaborativas. La colaboración es el contenido que se manipula. Por ejemplo, si hay un grupo que se comunica con un chat, la colaboración es la conversación del chat.
- La **coordinación**: se refiere a organizar la participación de los usuarios. Para llegar a un objetivo grupal los usuarios deberán pasar por distintas actividades. Los sistemas que cuentan con herramientas de coordinación organizan las tareas para que los usuarios las lleven a cabo en algún orden determinado. Esto se logra a través de la especificación de procesos colaborativos, como se detalla en el trabajo de Solano [17]. Por otro lado, cómo se trató en el trabajo de Briggs et all. [5], la coordinación también establece cómo es la interacción de los participantes dentro de una actividad. Esta interacción, puede ser diseñada usando protocolos de interacción donde se describen las operaciones concretas que pueden realizar los usuarios/roles en cada momento de una actividad.

2.1.2. Procesos

Un “proceso” es una serie de pasos y decisiones con el objetivo que un trabajo/tarea/actividad sea completado/a (algún ejemplo puede ser preparar una comida, limpiar un cuarto o cualquier tarea que lleve más de un paso). Los procesos apuntan a organizar las tareas y sub-tareas, para cumplir con algún objetivo y tratando de minimizar los posibles errores. Uno de los aspectos a tener en cuenta en el modelado de procesos es la coordinación y la interdependencia de las tareas. Esto significa distribuir las tareas para balancear los esfuerzos y colocar controles para que algunas tareas se realizan antes que otras o al mismo tiempo.

En el caso de UML, los procesos son representados con diagramas de actividad con actividades

como nodos (activity nodes) y conectados con ejes (activity edges). Un nodo puede representar la ejecución de un comportamiento como una simple operación aritmética, o el llamado a un método, o la manipulación de algún contenido. Los nodos pueden también incluir estructuras de control como decisión, sincronización o concurrencia, y estos se llaman "nodos de control". Se ve en la Figura 2.1 un ejemplo de diagrama de actividad, resaltando los nodos de control que representa un proceso.

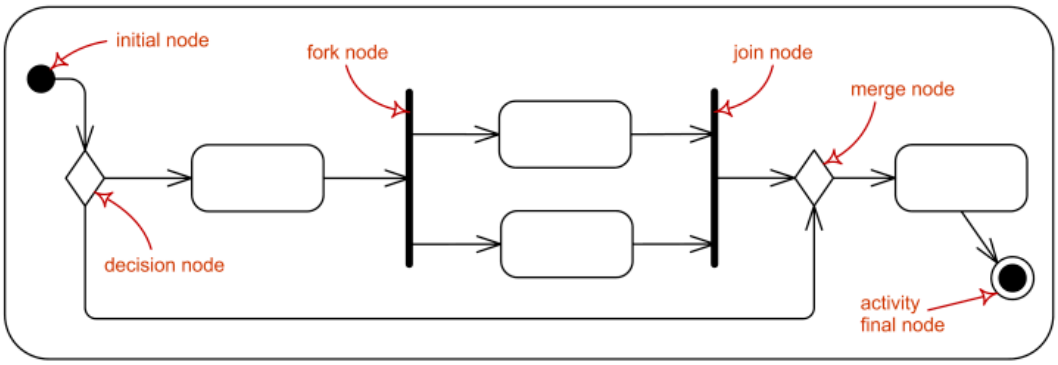


Figura 2.1: UML: Diagramas de Actividad [1]

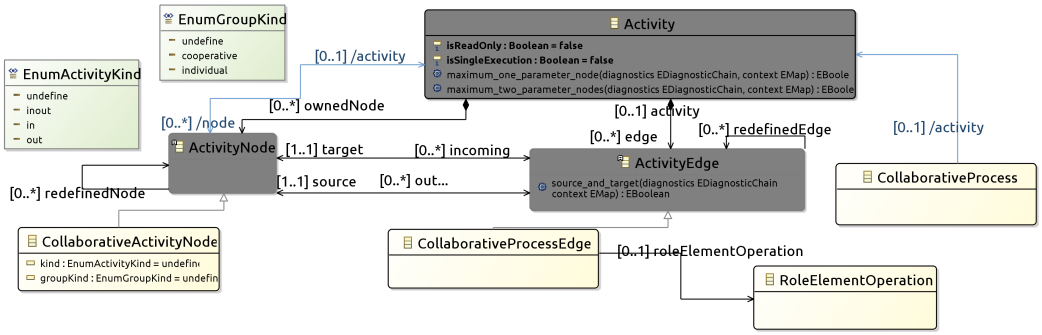


Figura 2.2: MetaClases para modelar procesos con UML2.0

Utilizando estos diagramas, los desarrolladores de software diseñan las características de los procesos y luego construyen programas que implementan estos modelos. Hasta los años 90, las implementaciones de procesos consistían en un código intrincado, difícil de cambiar y costoso de mantener. En una segunda generación, en los comienzos de los 90 cuando apareció el concepto de BPM (Business Brocess Management), se obtuvieron los primeros desarrollos de workflow management systems. En la tercera generación, aparece BPMN (Business Process Model an Notation), que es un modelado de procesos de negocio de punta a punta de la compañía, descripto en el trabajo de Bauer [18]. Esto brindó una alineación entre las tecnologías y los

[1] https://www.uml-diagrams.org/activity-diagrams-controls.html

negocios. Las principales tecnologías que soportan esta tercera generación son las siguientes: ARIS, EDOC, BPMN, UML2.0, BPMD.

Los diagramas de Actividad son los tipos de diagrama que tiene UML2.0 para modelar procesos. En la figura 2.2 se muestran las metaclases de UML que permiten describir los procesos.

En los diagramas de actividad se pueden incorporar actividades/tareas y actividades de control que permiten modelar procesos. Estos pueden contar con actividades en paralelo o consecutivas y/o se pueden modelar loops entre las actividades, como se muestra en la figura 2.3. También se muestra cómo se instancia un proceso con un editor estándar de UML (UML Designer).

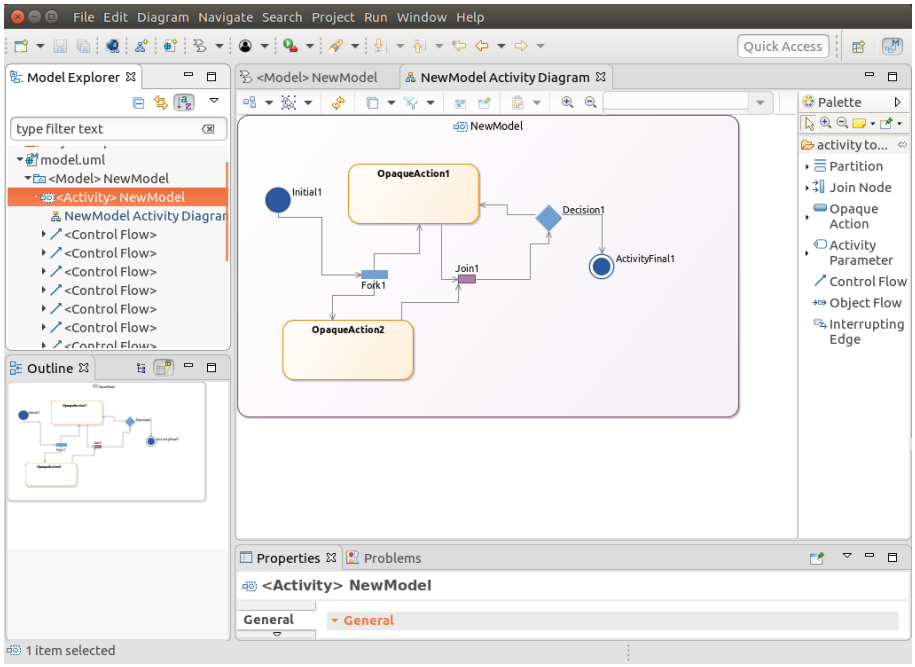


Figura 2.3: Diagramas de Actividad creados con UML Designer

2.1.3. Procesos Colaborativos

Los "procesos colaborativos" son una variante de los procesos donde los nodos (ActivityNodes) pueden ser actividades colaborativas. Al igual que en los diagramas de actividad, hay algunos nodos de control como, join, fork, decision, etc., que permiten modelar procesos complejos con loops y/o tareas en paralelo. Hay un ejemplo de proceso colaborativo que se muestra en la figura 2.4. Allí se representa un sprint en la metodología ágil Scrum [19], como un proceso colaborativo. En ese caso, se modelan las siguientes actividades colaborativas: Release Planning, Sprint Planning, Sprint Review, Daily Scrum, Retrospective, Deployment.

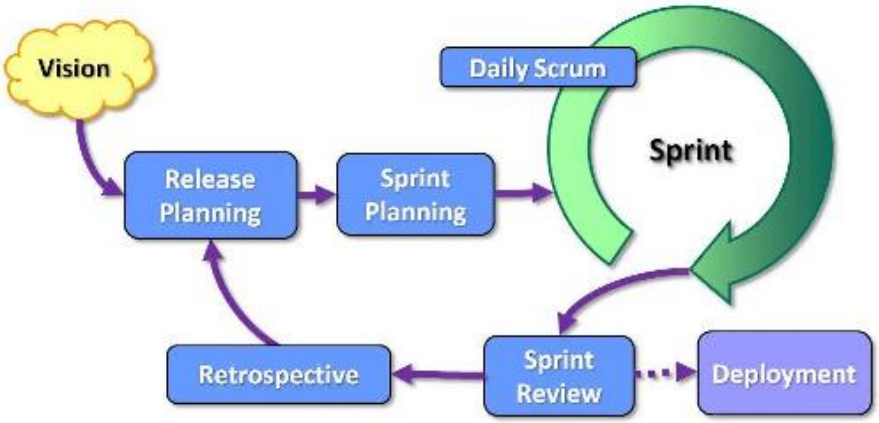


Figura 2.4: Scrum como proceso colaborativo [2]

En la figura 2.4, se puede ver un conjunto de actividades colaborativas que se realizan en un determinado orden, donde alguna de ellas hay que repetirlas (Daily Scrum) y que se llevan a cabo por un conjunto de usuarios que tienen distintos roles definidos en la metodología.

2.1.4. Awareness

Para lograr una efectiva colaboración, los usuarios tienen que estar informados sobre las acciones que los otros participantes realizan en el ambiente y cómo esas acciones afectan el entorno de trabajo. Compartir el espacio de trabajo físico, permite que las personas estén en conocimiento sobre la interacción de los demás con el espacio de trabajo, por ejemplo, hacia dónde miran los colaboradores, qué gestos están haciendo, qué herramientas tienen, etc. . Este conocimiento es el awareness del espacio de trabajo, que permite a los grupos colaborar de manera más efectiva. Esta información que brinda el sistema se llama "awareness", que según Gutwin y Greenberg [8, 20], es un concepto que solo se aplica a los sistemas colaborativos, ya que implica mantener información sobre las actividades que realizan los usuarios en el sistema e informarla al resto de los participantes.

Por otro lado, Dourish y Bellotti definen el awareness en [6, 9] como el "conocimiento de las actividades de otros que provee contexto para tus propias actividades".

- Esto permite a los usuarios coordinar su trabajo, basado en el conocimiento de lo que otros

[2] https://bmegias.wordpress.com/2010/11/22/motivacion-via-planificacion-en-scrum

hacen o han hecho. Los usuarios podrán ver, por ejemplo, los cambios en los documentos o el grado de avance de alguna tarea.

- Motiva la colaboración espontánea. Esto se realiza a partir de tener conciencia sobre el accionar de los otros usuarios. Es decir que los usuarios no están solos, sino que pueden ver que otros usuarios están en el mismo ambiente. Esto puede ser un disparador para comunicarse con los otros colaboradores. En algunas situaciones, como en los juegos multiusuarios, puede verse la ubicación de los jugadores lo que permite interactuar con ellos, como se ve en la figura 2.5.

- Otro aspecto interesante es que la información de awareness, permite que los miembros del equipo estén mejor informados sobre el avance del proyecto.

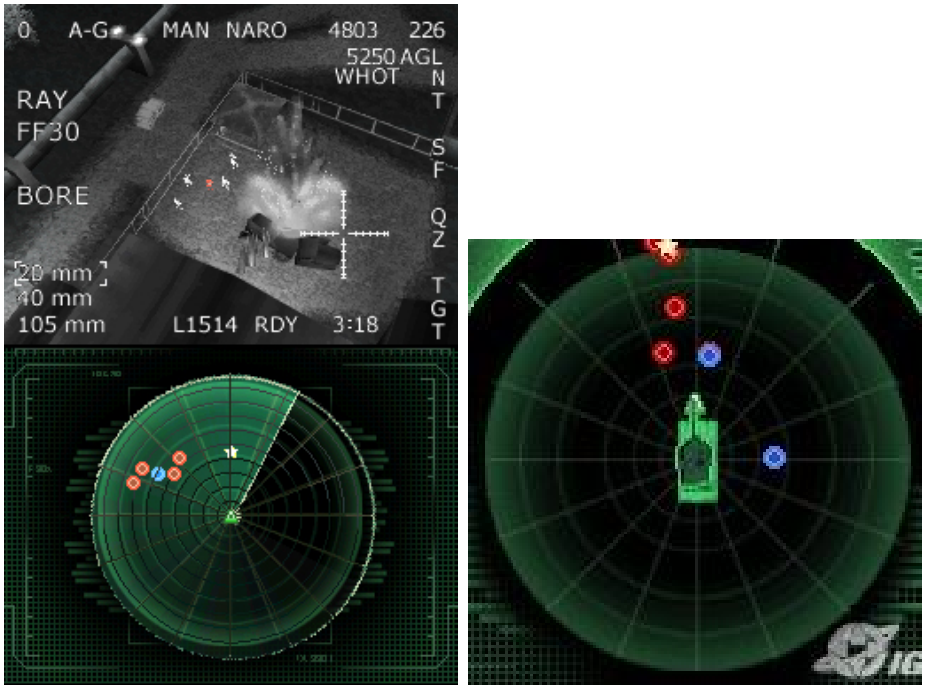


Figura 2.5: Awareness en juegos colaborativos [3]

También en el trabajo de Henri Ter Hofte [4], se define al awareness como la percepción o conocimiento del grupo y de las actividades que realizan los usuarios en el sistema compartido. Esto se refleja en la información que manejan los sistemas colaborativos, para brindar soporte a la colaboración. Por ejemplo, cuando un usuario ingresa al sistema, los otros que interactúan con él tienen que recibir información sobre el ingreso de su compañero. Este tipo de información es muy útil para que los usuarios puedan interactuar. Pero no solo se usa para informar el ingreso

[3] https://www.callofduty.com/

y egreso de los usuarios al ambiente compartido. El awareness es cualquier tipo de información que sirva para facilitar la colaboración y hay muchas situaciones donde es necesaria.

- Es necesario informar qué parte de la información está enfocando cada usuario. Esto sirve para que puedan hacer referencia a lo que está viendo cada uno.
- En un espacio compartido, es necesario saber qué rol tiene cada uno de los colaboradores.
- En algunos casos, es necesario conocer qué habilidades o qué herramientas tienen los colaboradores.
- En otros casos, se informa la ubicación de los usuarios, el avance de la tarea o cuándo fue realizada determinada tarea.

Element	**Relevant Questions**
Identity	Who is participating in the activity?
Location	Where are they?
Activity Level	Are they active in the workspace?
	How fast are the working?
Action	What are they doing?
	What are their current activities and task?
Intentions	What are they going to do?
	Where are they going to be?
Changes	What changes are they making?
	Where are changes being made?
Objects	What objects are they using?
Extents	What can they see?
Abilities	What can they do?
Sphere of Influence	Where can they have effects?
Expectations	What do they need me to do next?

Tabla 2.1: Elementos de Awareness en espacios de trabajo

En el trabajo de Carl Gutwin [7], se enumeran un conjunto de preguntas relevantes que sirven para identificar distintos tipos de awareness que pueden tener los sistemas colaborativos, como se muestran en la tabla 2.1.

Por otro lado, en el trabajo de Dourish [6], se brinda otra clasificación que sirve para entender mejor este concepto, puntualizado de la siguiente manera:

- **Awareness Informal**: Es el conocimiento general sobre los miembros del grupo, por ejemplo, su ubicación, presencia, foco, etc. Permite reconocer y aprovechar oportunidades para interactuar con otras personas de su entorno.
- **Awareness de estructura de grupo**: Informa la pertenencia a un grupo, roles, responsabilidades, particularmente dentro del equipo de trabajo.
- **Awareness social**: Muestra emociones (emoticones), intereses o vínculos entre los participantes.
- **Awareness del espacio de trabajo**: Información actualizada sobre modificaciones de la información compartida y sobre las actividades de los miembros del equipo.

En el trabajo de Collazos [21] se define un framework para ayudar a los ingenieros de sistemas colaborativos que quieran incorporar mecanismos de awareness en sus desarrollos. Este framework, cuenta con una propuesta metodológica o conjunto de fases a seguir, así como una taxonomía que incluye la información de awareness que se debe incorporar para mejorar la experiencia colaborativa.

El awareness por otro lado, viene a suplantar a la información con la que se cuenta en las reuniones presenciales, donde se sabe qué herramientas tiene cada uno de los colaboradores, hacia dónde están mirando, qué están haciendo, etc. En algunos casos, las reuniones virtuales pueden tener este mismo tipo de información, y de esta forma se aproximen a las experiencias que se tienen en las presenciales. Esa información suele llamarse "awareness sincrónico", ya que el sistema informa lo que está ocurriendo en ese momento. Una característica especial de los sistemas colaborativos es que el awareness, puede informar sincrónicamente situaciones que pasan en distintos lugares del sistema. Es como si en la vida real se pudiera estar en dos lugares al mismo tiempo y ver qué es lo que está pasando en ambos lugares.

En general, puede ser cualquier tipo de información que captura el sistema acerca de los usuarios y el estado de la colaboración. Los awareness pueden ser específicos para cada dominio en particular; por ejemplo, en un entorno colaborativo vinculado con educación, se puede querer informar cuántos trabajos tiene aprobados los alumnos o el grado de avance de alguna tarea.

Otra información con la que cuentan algunos sistemas groupware es el "awareness asincrónico". Esto se da, cuando el sistema informa a los usuarios de algo que ya pasó, por ejemplo, qué modificaciones hubo en algún documento o cuándo participaron otros usuarios en el entorno compartido.

El awareness relacionado con el espacio de trabajo, requiere mantener constantemente actualizada la información de los otros usuarios mientras colaboran, indicando al menos la identidad y la presencia de los usuarios. Ésta información también puede relacionarse, agrupándola para mejorar la calidad del awareness. En general, suele combinarse el awareness de presencia, con la información sobre la actividad que están desarrollando, su ubicación dentro del sistema, su estado, qué acciones va a desarrollar, qué cambios está realizando, objetos que se utilizan, etc.

Esta información facilita el trabajo de los usuarios, permitiéndoles tener una mejor percepción o conciencia de lo que está pasando con cada uno de ellos y con los otros usuarios en el ambiente colaborativo. Por ejemplo, cuando se usa algún sistema de mensajería instantánea (Messenger o WhatsApp o google talk) y aparece la indicación de que algún usuario está escribiendo ("juan está escribiendo" o "juan está grabando un audio", etc.), eso es awareness. Esa información no estaba en las versiones iniciales de Messenger o WhatsApp, pero se adoptó rápidamente debido a que es muy útil, ya que se establece un mecanismo de coordinación espontánea entre los usuarios que participan de la comunicación.

Todos los elementos del sistema pueden tener awareness (espacios, herramientas, procesos, etc.). Esto significa, que en el momento que ocurra algo en particular, el sistema mostrará información de awareness en alguno de los elementos del sistema. Los espacios pueden también tener información de awareness, por ejemplo, aquellos que muestran la lista de usuarios que ingresaron al mismo. En este caso, el espacio muestra la presencia de los usuarios como información de awareness. De la misma forma, una actividad puede tener información de awareness. Se puede indicar dentro de una actividad, cuáles son los roles de los usuarios que intervienen, como se mencionó en el trabajo de Belkadi et al [22].

El awareness es información que controla el sistema, no es información que introduce el usuario, es decir, el usuario no tiene que informar que ingresa o que sale del aula. Simplemente ingresa y ante la acción de ingresar, el sistema muestra donde corresponde el ingreso de ese usuario.

Entre los awareness más comunes aparecen: Presencia, Ubicación, Densidad (cuántos hay del

total), Información del Usuario (Edad, Nacionalidad, etc.), Nivel de actividad, Acciones (pueden mostrar un histórico de acciones), Lugares donde estuvo, Lugares donde realizó las acciones, Cambios que realizó, Objetos que controla, Qué puede alcanzar, Qué puede ver, Intensiones, Habilidades, Influencia y datos históricos sobre los anteriores.

2.1.5. Entornos virtuales y presenciales

En esta sección se analizan las características de los sistemas colaborativos en entornos virtuales en relación a actividades presenciales.

Para iniciar, la virtualidad tiene la ventaja lógica de reunir a un grupo de personas que pueden estar en lugares distantes. Esta posibilidad, evita costos de traslado y brinda la comodidad a los colaboradores de participar incluso desde su propia casa. A partir de esta ventaja inicial se pueden analizar otras características:

+ **Acceso a la información:** Los participantes al estar conectados a un sistema en red, pueden buscar todo tipo de información. Esto permite que los miembros del grupo accedan y compartan la información con el resto.

+ **Sincrónico y Asincrónico:** Con la aparición de los sistemas colaborativos, las personas trabajan y/o estudian con nuevas herramientas. Éstas permiten administrar reuniones que combinan actividades sincrónicas como chat o sistemas de conferencia, con otras asincrónicas que se atienden con foros o editores compartidos. También es posible que los participantes entren y salgan de las herramientas y compartan contenidos en distintos horarios.

+ **Trabajo en paralelo:** En el marco del modelado de procesos, una de las posibilidades es pensar a los procesos colaborativos como actividades que puedan llevarse a cabo en forma paralela. Estas actividades pueden combinar actividades individuales, con otras que son realizadas por un grupo de usuarios.

+ **Trabajo anónimo:** Otra posibilidad de las tareas a distancia, es pensar en algunas actividades donde los usuarios puedan participar de forma anónima. Esto puede ser útil en algunas situaciones, permitiendo que los usuarios se expresen libremente.

+ **Resultados de las reuniones:** Las reuniones virtuales utilizan distintas herramientas colaborativas, donde los usuarios editan distintos tipos de documentos. Luego de cada actividad queda como resultado el contenido de los documentos editados o compartidos por los participantes.

- **Dificultades para comunicarse:** Comparado con las reuniones presenciales, las virtuales tienen la dificultad de la comunicación restringida. Por un lado, en las reuniones presenciales no solo se escucha lo que se está diciendo, sino que se pueden ver los gestos, los movimientos de los participantes y se perciben otras cosas que pasan en el entorno. En las virtuales, en algunos casos, solo se puede enviar textos que comparten otros colaboradores. Esta comunicación restringida, puede ser un problema a la hora de explicar algunas cosas como chistes, o expresiones irónicas, etc. Como consecuencia de esto, los usuarios que participan de actividades colaborativas requieren mayor concentración para comprender lo que se está discutiendo.

  Hay varias formas de paliar estas dificultades: por un lado, contar con herramientas de comunicación que permitan transmitir sonido y video; otra, es mejorar la coordinación donde se organiza la participación de los colaboradores (no hablan todos al mismo tiempo) y finalmente, la incorporación de awareness, que aportan información adicional sobre lo que está pasando con los usuarios mientras colaboran.

- **Reduce la interacción:** Las actividades colaborativas distribuidas, tienden a ser más serias, los usuarios ponen el foco de atención sobre la tarea específica y en general no surgen temas extra laborales. Esto genera una ganancia en eficiencia y tiempo, pero reduce el intercambio social. Muchas de las actividades colaborativas "cara a cara" parecieran ser más intensas, con interacciones más ricas. Si bien raramente los miembros del grupo se miran directamente entre ellos, el hecho de estar en la misma habitación, incrementa la información de contexto sobre los otros miembros y puede dar pie a conversaciones que vayan más allá de los temas específicos de la reunión.

2.1.6. Espectro Sistemas Colaborativos

Los sistemas que soportan tareas comunes y ambientes compartidos varían ampliamente; por lo tanto, es apropiado analizarlos en términos de un espectro con múltiples dimensiones, en el que

los distintos sistemas se ubican en diferentes puntos del mismo. En esta sección se describen las dimensiones que permiten precisar las diferencias entre los sistemas de una manera exhaustiva y estable en el tiempo.

2.1.6.1. Nivel de Integración

Se describen aquí dos sub-dimensiones, una de acuerdo al grado de integración que tienen los usuarios en la tarea que realizan; y la otra en relación al grado de integración de los usuarios en el ambiente compartido. Asimismo se mencionan un par de ejemplos de aplicaciones que se ubicarán en los extremos de cada sub-dimensión.

Tarea Común: En la tabla 2.2 se muestra el grado de integración de acuerdo al tipo de tarea que se realiza.

Bajo:	**Alto:**
Los usuarios comparten un objetivo común pero realizan tareas en forma independiente. Cada usuario utiliza algún recurso del sistema pero el sistema no fomenta la colaboración.	Los usuarios realizan sus tareas en forma conjunta, dependen unos de otros y hay una constante percepción de las actividades que realizan los demás colaboradores. En el máximo nivel de integración se encuentran los sistemas de escritura colaborativa.

Tabla 2.2: Integración según la tarea

Los sistemas de bajo nivel de integración en relación a la tarea son similares a los de tiempo compartido [4], que atienden a un conjunto de usuarios que realizan tareas independientes entre sí. Cada usuario utiliza recursos del sistema para realizar su tarea. Existen algunas organizaciones que cuentan con grandes computadoras que pueden ser usadas en forma remota por un grupo de usuarios para realizar trabajos específicos, procesamiento de datos o cuentas de alta precisión o complejidad. Este escenario aporta un ambiente donde la integración de las tareas de los usuarios es mínima.

[4] **time sharing system:** https://en.wikipedia.org/wiki/Time-sharing

En el otro extremo, los sistemas de edición colaborativa permiten a un conjunto de usuarios escribir un documento en forma conjunta. En este caso, el grupo de usuarios está realizando una tarea con un alto nivel de integración desde el punto de vista de la tarea a realizar.

Ambiente compartido: La tabla 2.3 muestra el nivel de integración según cuan compartido es el ambiente.

Bajo:	**Alto:**
Los usuarios colaboran en ambientes independientes. Por ejemplo, los sistemas de correo electrónico.	Los usuarios colaboran en un ambiente donde comparten información y se comunican, como es el caso de los sistemas de aulas virtuales.

Tabla 2.3: Integración según el ambiente compartido

La dimensión que clasifica a los sistemas de acuerdo al nivel de utilización del ambiente, muestra en un extremo a los sistemas de mail (Electronic Mail System). Este tipo de sistema permite enviar y recibir correos electrónicos y cada usuario tiene su ambiente de trabajo para hacerlo. Por ello, es que tiene un bajo nivel de utilización del ambiente compartido. En el otro extremo encontramos, por ejemplo, a los sistemas de aulas virtuales, donde los usuarios trabajan en un ambiente compartido. Estos últimos, permiten a un conjunto de usuarios con distintos roles (instructores, profesores, alumnos) presentar material, discutir algún tema, preguntar y responder, emulando un aula tradicional. Todo se lleva a cabo en un ambiente de alta integración, ya que los usuarios comparten herramientas e información (por ejemplo, el material de clase o la lista de usuarios que están en la misma.)

2.1.6.2. Tiempo y Espacio

Los sistemas colaborativos pueden ser concebidos tanto para ayudar a grupos en los cuales las personas se encuentren en un mismo lugar, como para grupos en los cuales sus integrantes estén distribuidos en distintas ubicaciones. A su vez, un sistema colaborativo puede ser concebido para reforzar la comunicación y la colaboración, dentro de una interacción en tiempo real (sincrónica), o de una interacción que no se lleva a cabo en tiempo real (asincrónica).

Estas consideraciones de tiempo y espacio [4], sugieren cuatro categorías de sistemas colaborativos representados en la tabla 2.4 de doble entrada:

	Mismo Tiempo	**Distinto Tiempo**
Mismo Lugar	Interacción cara a cara	Interacción asincrónica
Distinto Lugar	Interacción distribuida Sincrónica	Interacción distribuida Asincrónica

Tabla 2.4: Integración según el tiempo y el espacio

La tecnología que se utiliza para soportar las reuniones (meeting room) se ubica dentro del cuadrante izquierdo superior; es decir, mismo lugar - mismo tiempo: este tipo de sistemas consiste de una habitación con una gran pantalla común conectada a un conjunto de computadoras interconectadas. Estos sistemas han surgido con el objetivo de mejorar los encuentros cara a cara y sirven para procesos como brainstorming, organización y evoluación de ideas.

La tecnología de pizarrón compartido (shared whiteboard) se encuentra dentro de la celda inferior izquierda; mismo tiempo - distinto lugar: fueron diseñados para dar soporte a actividades donde se trabaja sobre dibujos o diseños en el pizarrón virtual. En este tipo de encuentros, por ejemplo, las personas pueden dibujar algunos objetos y relacionarlos. Cada persona del encuentro puede referirse a los dibujos y realizar modificaciones. Los objetos dibujados en el pizarrón compartido serán visibles a todos los usuarios.

En el cuadrante superior derecho; mismo lugar - distinto tiempo: se ubican tecnologías de noticias (bulletin boards o news groups), que son una variante de los sistemas de mail. En lugar de enviar un mensaje a una o más personas (como en el caso del mail), estos sistemas permiten a los usuarios enviar un mensaje a un "lugar" identificado unívocamente en el espacio, utilizado para discutir acerca de un tema en particular.

Por último, el sistema de correo electrónico en la celda inferior derecha es el ejemplo más común de groupware. Permite intercambiar mensajes textuales en forma asincrónica. Otro ejemplo de sistemas que se pueden ubicar en esta celda son los sistemas de group scheduling. Programar un encuentro con un grupo de personas es una de las tareas de grupo que podría beneficiarse con el soporte de computadoras. Encontrar tiempo libre en varias agendas personales es una tarea que puede ser hecha más eficientemente mediante un sistema de groupware. Hay ejemplos de estos sistemas que utilizan un mecanismo de voto para organizar el plan de encuentro. Otros sistemas

más nuevos involucran el intercambio de una serie de mails que son invitaciones, respuestas de confirmación y negaciones.

Un sistema colaborativo podría estar compuesto por distintas herramientas, donde algunas se podrían ubicar en algún cuadrante y otras en otro. Por ejemplo, podría ser útil tener dentro del mismo entorno herramientas para editar un documento en tiempo real con un grupo (mismo tiempo / mismo lugar) y otras herramientas que permitan comentar entradas en un foro (diferente tiempo / mismo lugar).

2.1.6.3. Permisivo vs. Restrictivo

Otra dimensión del espectro que se analiza es el grado de libertad que tienen los usuarios para desarrollar sus tareas. Algunos sistemas colaborativos como los sistemas de workflows, restringen o dirigen el comportamiento de los usuarios. Este tipo de sistemas son considerados restrictivos, y típicamente mantienen un curso de acción posible o deseable que restringe las posibilidades de acción de los usuarios. Estos sistemas son habituales en empresas comerciales para organizar y optimizar los circuitos de trabajo; los usuarios hacen lo que el sistema les indica realizar. Asimismo, pueden o no tener instancias de colaboración sincrónica entre los usuarios.

Otros, como los pizarrones compartidos o sistemas de escritura colaborativa permiten que los usuarios puedan libremente realizar sus actividades (escribir, navegar, dibujar, entrar o salir, etc.) sin control del sistema. Estos, son habituales en ámbitos de construcción de conocimiento (encuentros de usuarios para diseño o investigación) y suelen tener espacios de colaboración sincrónica donde los usuarios se encuentran para compartir sus conocimientos [8, 23].

Los sistemas permisivos son más complejos de desarrollar. Por un lado, requieren soportar las distintas alternativas de acción que se le dan a los usuarios, brindando la funcionalidad que corresponda (por ejemplo, navegar entre los documentos, escribir mientras otros realizan otras actividades, etc.). Y por otro, requieren elementos de awareness que informen qué actividades los usuarios están realizando, dónde la están llevando a cabo, con qué elementos, etc. Como ejemplo simple, está el caso en que un grupo de usuarios se reúne para escribir conjuntamente un documento. Si el sistema es permisivo, los usuarios pueden cambiar la página que están visualizando del documento, el sistema debe informar qué parte del documento cada usuario

está visualizando. Esa es una información de awareness que un sistema restrictivo no tiene la necesidad de implementar, ya que todos estarían viendo la misma porción de información.

2.1.6.4. Otras variables

Hay otras variantes al momento de utilizar un sistema colaborativo. Hay un conjunto de variables que determinan cuáles deben ser las herramientas colaborativas más adecuadas. Estas variables son:

- Duración de Actividad Colaborativa
- Tamaño del Grupo
- Diversidad socio-cultural

La duración de la actividad puede indicar el tipo de herramientas a utilizar. En el caso de actividades de corta duración, se podrán utilizar herramientas sincrónicas (chat, video llamadas, edición simultanea). Por otro, si las actividades son de larga duración, las herramientas tendrán que ser asincrónicas. Con el tamaño del grupo pasa algo similar; puede ser pequeño (2 o 3 participantes), o grande donde pueden participar cientos de personas. Las herramientas colaborativas para atender a pocas personas podrán ser sincrónicas, con bastante funcionalidad de awareness. En cambio, cuando el grupo es grande, las herramientas tendrán que ser asincrónicas. En cuanto a la diversidad socio cultural, se refiere al grado de compatibilidad en la cultura y terminología con la cual se va a trabajar en la actividad. Se dirá que la compatibilidad es baja, cuando los miembros pertenezcan a distintas culturas o edades o conocimientos distintos, sobre la temática que se trate. La compatibilidad va a ser alta, cuando los integrantes de la actividad colaborativa tengan conocimientos similares, lo que requerirá menor esfuerzo para entenderse. Para los grupos de baja compatibilidad el sistema tendrá que brindar funcionalidad (generalmente mejoras en el awareness), para acercar la compatibilidad de los usuarios. Un ejemplo de este caso es el de herramientas que atienden a usuarios que hablan distintos idiomas, en este caso la herramienta tendrá que hacer traducción automática. Este es el caso del Meet de Google [5].

[5] https://meet.google.com

2.2. Paradigmas de Desarrollo de Software

La Ingeniería de software, según Boehm [24], "es la aplicación práctica del conocimiento científico al diseño, desarrollo de programas de computadora y a la documentación asociada requerida para desarrollar, operar y mantenerlos". También se conoce como desarrollo de Software o Producción de Software. Bauer [25], trata a la Ingeniería de Software, como el establecimiento de principios y métodos de la Ingeniería para obtener software rentable que sea fiable y trabaje en máquinas reales. Luego en 1993 la IEEE [26] lo sintetizó en la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación (funcionamiento) y mantenimiento del software.

Estas definiciones, surgieron a raíz de lo que se conoció como "La Crisis del Software", que se refiere a los problemas de productividad, efectividad y eficacia que, desde sus inicios, han experimentando las empresas de desarrollo de software. Una de las causas que originaron la crisis, era que se trataba de una disciplina relativamente nueva, donde no había suficiente personal capacitado y una pobre implementación de procesos involucrados en el desarrollo de software. Por otro lado, el software en sus comienzos consistía de un puñado de instrucciones estructuradas como un conjunto de hilos intrincados y anudados que dificultaban la corrección de errores y el mantenimiento de los sistemas.

Desde fines de los '60, distintos estudios de paradigmas, técnicas y herramientas relacionadas con la ingeniería de software tratan de sistematizar los procesos de desarrollo para acotar el riesgo de fracaso sobre la base de la experiencia previa. Uno de los paradigmas que incrementa la productividad, simplifica el proceso de desarrollo y promueve la comunicación entre las personas que trabajan en equipos de desarrollo es la Ingeniería de software dirigida por Modelos (MDE, por su nombre en Inglés Model Driven Engineering) también conocido como Desarrollo de software Dirigido por Modelos (MDD, por su nombre en Inglés Model Driven software Develpment), como se trató en el libro: "Desarrollo dirigido por modelos"de Pons, Giandini y Perez [10]. En esta área, la iniciativa más conocida es la Arquitectura dirigida por modelos (MDA, por sus siglas en inglés Model Driven Architecture) [11], desarrollada por el OMG (Object Management Group) [27] que provee un conjunto de estándares para la puesta en práctica de MDD.

2.2.1. Desarrollo Dirigido por Modelos (MDD)

La Ingeniería de Software establece, que el problema de construir software debe ser tratado de la misma forma en que los ingenieros construyen otros sistemas complejos, como puentes, edificios, barcos, aviones. Es decir, un proceso planificado basado en metodologías formales apoyadas por el uso de herramientas. Uno de los paradigmas que responden a esta premisa es el Desarrollo Dirigida por Modelos MDD (por sus siglas en ingles: Model Driven software Development) [10, 12, 28], que se ha convertido en un nuevo paradigma, que propone mejorar la construcción de software a través de un proceso guiado por modelos y soportado por potentes herramientas que generan código a partir de dichos modelos.

MDD promete una mejora de la productividad y de la calidad del software generado debido a que se reduce el salto semántico entre el dominio del problema y de la solución; se reducen los tiempos de desarrollo y las herramientas de generación pueden utilizar frameworks, patrones y técnicas cuyo éxito ya se ha comprobado. El paradigma MDD tiene dos ejes principales:

- Por un lado, hace énfasis en la separación entre la especificación de la funcionalidad esencial del sistema y la implementación de dicha funcionalidad, usando plataformas tecnológicas específicas. Para ello, el MDD identifica dos tipos principales de modelos: modelos con alto nivel de abstracción e independientes de cualquier tecnología de implementación y modelos que especifican el sistema en términos de construcciones de implementación disponibles en alguna tecnología específica.

- Por otro, los modelos son considerados los conductores primarios en todos los aspectos del desarrollo de software. Los modelos más abstractos son transformados en uno o más modelos más concretos, es decir que para cada plataforma tecnológica específica se genera un modelo específico. La transformación entre modelos constituye el motor del MDD y de esta manera los modelos pasan de ser entidades meramente contemplativas a ser entidades productivas, como se trató en el trabajo de tesis de Giandini [29].

El desarrollo de software dirigido por modelos permite mejorar las prácticas corrientes de desarrollo de software. Las ventajas de MDD son las siguientes:

2.2.1.1. Incremento de la Productividad

Una de las principales ventajas del paradigma MDD, es la mejora en la productividad en los proyectos de desarrollo de software. Es sabido que en el proceso de desarrollo tradicional, los modelos y diagramas que son creados en las primeras fases, pierden rápidamente su valor cuando la codificación comienza. La conexión entre los diagramas y el código se va perdiendo a medida que progresa la fase de codificación. Los cambios se hacen a menudo directamente en el código, porque no se utiliza el tiempo para actualizar los diagramas y otros documentos de alto nivel. También, el valor agregado de diagramas actualizados y documentos es cuestionable, porque cualquier nuevo cambio se encuentra en el código de todos modos.

Mientras se mantiene el mismo equipo de trabajo, en el desarrollo del software, existe bastante conocimiento de alto nivel compartido en el equipo. Los problemas comienzan cuando se cambia parte del equipo, que sucede generalmente después de que se entrega del primer lanzamiento el software. Tener solamente actualizado el código hace muy difícil la tarea de mantenimiento de un sistema. MDD minimiza este problema, ya que lo que se mantiene actualizado son los modelos abstractos y éstos son más fáciles de entender que el código fuente.

En MDD, el foco está puesto en el desarrollo de un modelo abstracto y a partir de él se obtienen los modelos más concretos que son generados a través de transformaciones. Para llegar a tener un sistema funcionando, por supuesto, es necesario que alguien defina la transformación, que es una tarea difícil y especializada. Pero esa transformación necesita ser definida sólo una vez, y puede ser aplicada en el desarrollo de muchos sistemas. En síntesis, los desarrolladores al enfocarse en el modelo abstracto, trabajan independientemente de los detalles técnicos de las plataformas donde se aplicarán estos modelos, lo que mejora la productividad de la siguiente manera:

- Los desarrolladores del modelo tienen menos trabajo, ya que los detalles específicos de la plataforma no necesitan ser diseñadas. Estos detalles técnicos se agregan automáticamente por la transformación entre modelos.

- Los desarrolladores trabajan con el modelo abtracto y con conceptos más cercanos al cliente/usuario final. Lo que mejora la comunicación dentro del equipo de desarrollo y de esta forma obtener el sistema más adecuado a lo que el usuario final requiere.

Estas mejoras en la productividad, se optimizan con el uso de herramientas que automáticamente obtengan una implementación concreta a partir de un modelo abstracto.

2.2.1.2. Adaptación a los cambios tecnológicos

La industria del software tiene una característica especial que la diferencia de las otras industrias. Todo el tiempo, aparecen nuevas tecnologías y rápidamente llegan a ser populares (por ejemplo, Java, NET, Ruby, PHP, Python, servicios de WEB, SaaS, REST, Mongo DB, XML, JSON, HTML, CSS, UML, MEAN, Node, Angular, Express, etc.).

Las nuevas tecnologías ofrecen beneficios concretos para las compañías y muchas de ellas no pueden quedarse atrás, por lo tanto, tienen que utilizarlas muy rápidamente. El software ya existente puede seguir sin cambios, pero necesitará comunicarse con los nuevos sistemas que serán construidos usando una nueva tecnología. Dentro de MDD la portabilidad se lleva a cabo enfocando en el desarrollo de modelos que son independientes de la plataforma. El mismo modelo abstracto puede ser automáticamente transformado en muchos modelos específicos para diferentes plataformas. Por lo tanto, todo lo que se especifica en el nivel del modelo abstracto es completamente portable y solo hay que crear o adaptar alguna herramienta de transformación que entienda la nueva tecnología con la que se va a trabajar.

2.2.1.3. Interoperabilidad

La mayoría de los sistemas necesitan comunicarse con otros que generalmente ya fueron desarrollados. Incluso cuando los sistemas se construyen de cero, utilizan a menudo múltiples tecnologías. Por ejemplo, un sistema WEB necesita usar una base de datos para almacenar información. Al mismo tiempo, las tecnologías utilizadas pueden tener diferentes versiones que se van actualizando a medida que pasan los años.

En MDD se pueden generar muchas implementaciones a partir del mismo modelo, y estas frecuentemente pueden estar relacionadas. Estas relaciones se llaman bridges o puentes. Como ejemplo, transformar un modelo (diagrama de clases UML) a dos modelos específicos, por ejemplo, el primero a un modelo Java y el segundo a un modelo de base de datos relacional.

Construir un puente entre ambos elementos es posible. La interoperabilidad entre plataformas

puede ser realizada por herramientas para transformación de modelos que generen, además, puentes entre ellos y posiblemente entre otras plataformas.

2.2.1.4. Mantenimiento y Documentación

En el proceso del desarrollo de software, la generación de la documentación fue siempre un punto débil. La mayoría de los desarrolladores tiene como prioridad producir código. Generar documentación durante el desarrollo cuesta tiempo y retrasa el proceso; sin embargo, ayudará en su tarea a los encargados de integrar el proyecto en etapas finales. Así pues, escribir documentación se ve como algo para el futuro, no para el presente. Por otro lado, no hay incentivo para la documentación (con excepción del líder de proyecto, que sostiene que la documentación es necesaria). Por supuesto, los desarrolladores están equivocados. Su tarea es desarrollar sistemas que puedan cambiar y que se puedan mantener en el tiempo. A pesar de la sensación de muchos desarrolladores, escribir la documentación es una de sus tareas esenciales.

Una solución a este problema, a nivel de código, es la facilidad de generar la documentación a partir del código fuente, asegurándose así de que esté siempre actualizada. La documentación es parte del código y no algo separado. Esta solución, sin embargo, resuelve únicamente el problema de la documentación en niveles inferiores. La documentación de alto nivel (texto y diagramas) todavía necesita ser mantenida a mano. Dada la complejidad de los sistemas que se construyen, la documentación en un nivel más alto de la abstracción resulta obligatoria.

Con MDD los desarrolladores pueden enfocarse solamente en el modelo, el cual tiene un nivel más abstracto que el código. El modelo abstracto es usado entonces para generar modelos más concretos, los cuáles posteriormente, serán usado para generar código. Por lo tanto, el modelo será una exacta representación del código. Así el modelo cumplirá la función de documentación de alto nivel necesaria en cualquier sistema de software. La gran diferencia es que el modelo abstracto no se deja de lado luego de ser escrito. Los cambios hechos en el sistema serán hechos sobre el modelo y regenerando los modelos concretos y el código. Esta tarea requerirá su automatización mediante herramientas adecuadas para ello.

2.2.1.5. Arquitectura Dirigida por Modelos (MDA)

Han surgido varios enfoques dentro del ámbito de MDD, pero sin duda la iniciativa más conocida y extendida es la MDA, acrónimo de Model Driven Architecture [11, 12], presentada por el consorcio OMG [30] (Object Management Group) en noviembre de 2000 con el objetivo de abordar los desafíos de integración de aplicaciones y los continuos cambios tecnológicos.

MDA es una arquitectura que apunta al diseño de software, desarrollo e implementación y provee guías que estructuran el software a partir de modelos. MDA separa la lógica del negocio de la aplicación en una plataforma tecnológica particular.

Teniendo en cuenta que MDA principalmente trabaja con dos tipos de modelos: los de alto nivel de abstracción e independientes de cualquier tecnología de implementación, llamados PIM (Platform Independent Model) y modelos que especifican el sistema en términos de construcciones de implementación disponibles en alguna tecnología específica, conocidos como PSM (Platform Specific Model).

Un PIM construído con UML u otra herramienta estándar de modelado de la OMG, se pueden implementar a través de MDA en prácticamente cualquier plataforma abierta o patentada, incluidos los servicios web, .NET, CORBA, J2EE y otros. De este modo, el desarrollador crea modelos de alto nivel e independientes de la plataforma (PIM), que posteriormente, mediante transformaciones modelo a modelo (M2M), generan de manera automática modelos dependientes de la plataforma (PSM). Y finalmente, al igual que en MDD, el código de la aplicación es derivado de los modelos específicos de la plataforma (PSM) mediante transformaciones modelo a texto (M2T).

La arquitectura dirigida por modelos (MDA) está relacionada con varios estándares. Algunos de ellos son: el Unified Modeling Language (UML) [16], el Meta-Object Facility (MOF) [31], el XML Metadata Interchange (XMI), el Enterprise Distributed Object Computing (EDOC), el Software Process Engineering Metamodel (SPEM) [32] y el Common Warehouse Metamodel (CWM) [33].

2.2.1.6. Principios de MDA

El OMG ha definido MDA y los lenguajes asociados a MDA, en base a los siguientes principios que subyacen a la visión particular del OMG sobre la Ingeniería Dirigida por Modelos o Desarrollo de software Dirigido por Modelos (MDD) [34].

- Los modelos deben ser expresados en una notación bien definida, de manera que permitan una comunicación y comprensión eficaz de la descripción de sistemas a escala empresarial.
- Las especificaciones de los sistemas deben organizarse en torno a un conjunto de modelos asociados a transformaciones que implementen las asignaciones y las relaciones entre ellos. Todo esto, permite un diseño organizado basado en un marco arquitectónico multicapa y multiperspectiva.
- Los modelos deben ser construidos en conformidad con un conjunto de meta-modelos, lo que facilita la integración y transformación entre ellos, y la automatización a través de herramientas.
- Este marco de modelado debe aumentar la aceptación y la adopción del enfoque del MDD, y fomentar la competencia entre los proveedores de herramientas de modelado.

Como se mencionó, las transformaciones modelo-modelo y modelo-código juegan un papel crucial en MDA. Se necesitan lenguajes de transformación especiales para escribir cada una de las definiciones de transformación que establecen las correspondencias (relaciones, mappings o mapeos) entre dos lenguajes de modelado (o entre un lenguaje de modelado y un lenguaje de programación).

2.2.1.7. Mapeos

Un mapeo consiste en la definición de la correspondencia entre elementos de dos modelos diferentes, como se explica en la tesis de Giandini [35]. La OMG provee un catálogo de mapeos para diferentes lenguajes, como se muestra en la siguiente página
https://www.omg.org/spec/category/language-mapping/.

El mapeo entre diferentes modelos (o niveles de modelado) puede ser implementado a través de transformaciones. Los mapeos pueden proporcionar una especificación conceptual, que permite

implementar la transformación entre los diferentes niveles de modelado propuestos por MDA (o incluso por MDD). El objetivo es, obviamente, automatizar la implementación del mapeo (es decir, de las transformaciones entre modelos) tanto como sea posible, evitando así costosas transformaciones manuales.

Muchas veces, a la hora de definir un mapeo, es necesario tener en cuenta decisiones tomadas a lo largo del desarrollo, en términos de algunos detalles en modelos de nivel inferior, como se explica en el libro de Mellor [12]. Por ejemplo, en un mapeo de PIM a PSM, deben ser tenidas en cuenta decisiones que, si bien se aplican a nivel de PIM, dependen de opciones a nivel de PSM. En estos casos, los mapeos definen anotaciones (marcas, en la jerga MDA) que se utilizarán para guiar las transformaciones entre los dos niveles de modelado.

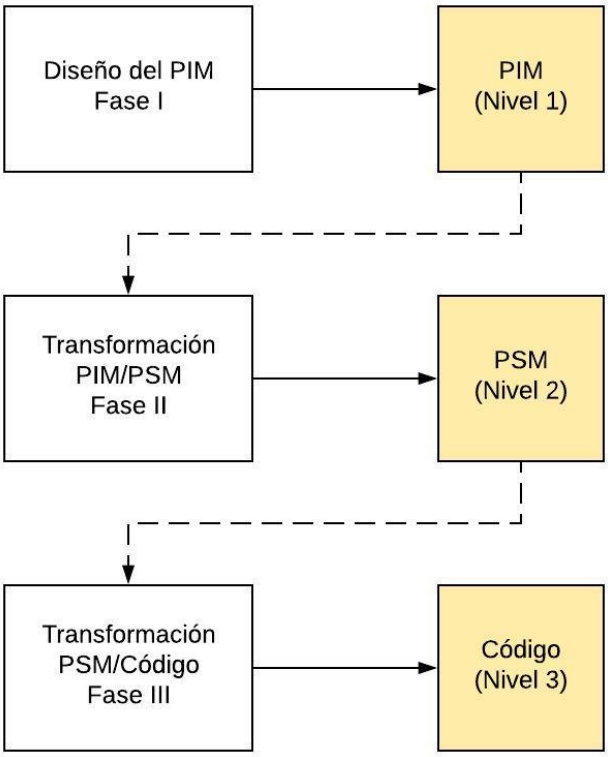


Figura 2.6: Pasos y Niveles en MDA

Para llevar a cabo todo este proceso, MDA define fases que a su vez delimitan tres niveles de abstracción del sistema a desarrollar, como muestra la Figura 2.6.

- Fase I: Creación de un Modelo Independiente de la Plataforma (Platform Independent Model o PIM). Es el modelo con el mayor nivel de abstracción del sistema y describe la

funcionalidad del mismo, pero omitiendo detalles de implementación.

- Fase II: Transformación del PIM a uno o varios Modelos Específicos de Plataforma (Platform Dependent Model o PSM). Un PSM es el mismo modelo que el PIM pero describiendo el sistema de acuerdo a una plataforma dada, como .Net, Java o incluso a entornos diferentes como Web o Mobile.

- Generación del código a partir del PSM. Consiste en hacer un mapeo o transformación del modelo PSM a la plataforma en la que está especificado. Esto se puede realizar de forma automática ya que el PSM se encuentra muy ligado a dicha plataforma.

Sin embargo, pueden observarse en la práctica, situaciones donde existan más de tres niveles de abstracción. Pueden existir transformaciones intermedias entre modelos que repitan la Fase II (PIM/PSM), donde el modelo de salida de una transformación es la entrada para la siguiente, generando así una cadena hasta llegar a la generación de código, como se explica en la página de MDA de la OMG [36].

Por ejemplo, se podría definir una transformación que vincule elementos de UML a elementos Java, la cual describiría como los elementos Java pueden ser generados a partir de los elementos UML. Esta situación se muestra en la figura 2.7.

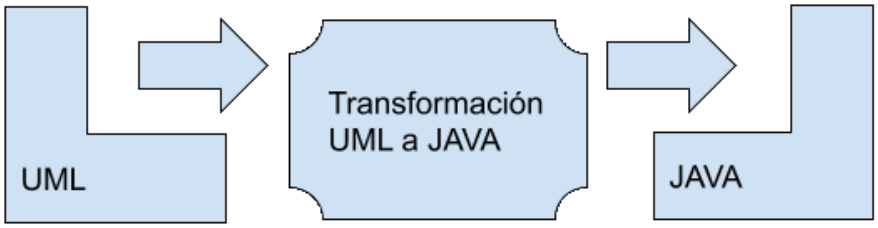


Figura 2.7: Transformación UML a JAVA

En general, una definición de transformación consiste en una colección de reglas, las cuales son especificaciones no ambiguas de las formas en que un modelo (o parte de él) puede ser usado para crear otro modelo (o parte de él). Las siguientes definiciones fueron extraídas del libro de Anneke Kepple [11]:

- Una transformación es la generación automática de un modelo destino, desde un modelo fuente, de acuerdo a una definición de transformación.

- Una definición de transformación es un conjunto de reglas de transformación que juntas, describen como un modelo en el lenguaje fuente puede ser transformado en un modelo en el lenguaje destino.

- Una regla de transformación es una descripción de cómo una o más construcciones en el lenguaje fuente pueden ser transformadas en una o más construcciones en el lenguaje destino.

2.3. Conclusiones del capítulo

En este capítulo se presentaron los conceptos teóricos que sustentan esta tesis. Se discuten las características del desarrollo dirigido por modelos MDD y los ejes principales del paradigma. Se presentan aspectos de mejora de la productividad y adaptación a los cambios tecnológicos en el desarrollo de software. Asimismo, se menciona cómo el paradigma MDD sustenta la interoperabilidad con otros sistemas. Otros aspectos que se tratan en este capítulo son las ventajas que aporta MDD al mantenimiento y la documentación de los sistemas. También se presenta la arquitectura MDA, como la iniciativa más conocida y extendida para instrumentar MDD, definiendo, cómo deben ser expresados los modelos, los mapeos y las herramientas de transformación que se utilizan para obtener los códigos fuentes de los sistemas a desarrollar.

Dentro del marco teórico de este trabajo, se presentan las características de los Sistemas Colaborativos con awareness, iniciando por las definiciones originales y puntualizando los aspectos de comunicación, colaboración y coordinación que brindan. Se focalizan los conceptos de procesos colaborativos y awareness que forman parte central de la dinámica de interacción de estos sistemas.

Las clasificaciones vistas en este capítulo permiten entender mejor qué tecnología es la más apropiada en cada caso; por ejemplo, cuando se quiere trabajar con muchos usuarios los más conveniente es promover actividades asincrónicas (distinto tiempo) donde los usuarios no necesiten trabajar conjuntamente en la misma tarea (bajo nivel de integración según la tarea). Al mismo tiempo, el sistema debería ser permisivo, para que los usuarios elijan sobre cuáles objetos quieren trabajar. Una herramienta que puede responder a estos requisitos es la Wiki, ya que permite que cientos de usuarios escriban un documento compartido.

En cambio, si se quiere trabajar con pocos usuarios, se pueden tener interacciones sincrónicas accediendo al mismo objeto del espacio compartido, incluso con posibilidad de tener edición simultánea sobre el mismo objeto.

Estas clasificaciones combinan aspectos sociales de interacción entre las personas y tecnología que soporta la participación de los usuarios. Puede observarse, que hay un amplio rango de aplicaciones colaborativas, que va desde interacciones cortas de un par de usuarios con un chat, hasta miles de usuarios escribiendo en una wiki durante varios años - como el caso de wikipedia -.

Capítulo 3

Estado del Arte

3.1. Herramientas para modelar sistemas colaborativos

Teniendo en cuenta que desarrollar aplicaciones colaborativas es una tarea compleja, ya que estos sistemas mantienen a un conjunto de actores/usuarios interactuando con un modelo compartido, en el trabajo "Sistemas Colaborativos con Awareness: Requisitos para su Modelado" [37] se buscan recursos de Ingeniería de Software (Métodos, Lenguajes, Herramientas) que faciliten la construcción de este tipos de sistemas. En ese trabajo se desarrolló una revisión sistemática de la literatura, siguiendo la guía de Kitchenham [38], donde se buscaron recursos de ingeniería de software, para la construcción de software colaborativo con awareness.

En la revisión sistemática [37], se utilizó el siguiente texto de búsqueda que contiene las palabras claves que se utilizan en las publicaciones científicas:

Texto de Búsqueda
(Groupware OR "Collaborative system") And (Model OR Design OR "Software Design" OR "Conceptual Model") And (Awareness)*

La búsqueda se realizó en Octubre de 2015[6], utilizando los servicios del Ministerio de Educación de Argentina, servicios de ACM digital library y el meta-buscador SCOPUS.

De acuerdo al proceso de revisión sistemática [38], los trabajos pasaron por un conjunto de

[6]Los resultados del análisis se puede encontrar en http://goo.gl/LPQY6k

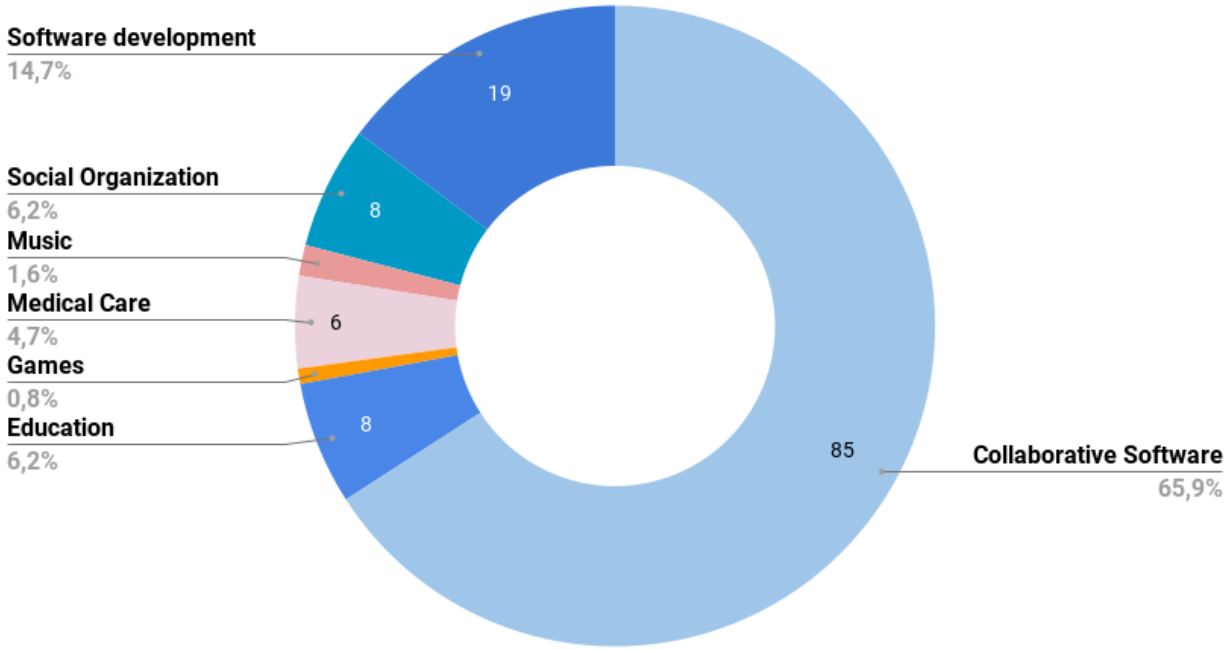


Figura 3.1: Distribución de Trabajos

preguntas, para encontrar aquellos trabajos que interesan a esta búsqueda. A continuación, se analizan las preguntas de la planificación de la revisión para encontrar los trabajos.

Pregunta 1:

¿A qué área se aplica el awareness en los sistema colaborativos propuestos?

De los 129 trabajos encontrados, se hizo una primera clasificación y 44 no trataban el desarrollo de sistemas colaborativos, sino que se referían a otras áreas como el uso de software con awareness en medicina, organización social, juegos o educación. Dentro de los 44 trabajos que se descartaron, hubo 19 en donde se mostraban herramientas para desarrollar software colaborativamente, es decir que utilizan herramientas colaborativas para desarrollar software en general, pero lo que obtienen no necesariamente es software colaborativo. En esa revisión se buscaban recursos de Ingeniería de Software, que permitieran desarrollar software colaborativo con awareness y entonces se pone foco en los 85 que sí trabajan ese tema, como se muestra en la Figura 3.1.

Pregunta 2:

¿Propone algún artefacto de ingeniería de software para modelar el Awareness en sistemas colaborativos?

A partir de esa primera pregunta, se chequea si los trabajos presentan algún recurso de ingeniería de software para modelar, diseñar, concebir o implementar el awareness en sistemas colaborativos. De los 85 trabajos que hacían referencia a Software Colaborativo con Awareness, 56 no se refieren al desarrollo del software, como se muestra en la figura 3.2, sino que proponen utilizar awareness en otras disciplinas como Juegos o KMS (Knowlege Management System), entre otras. Por otro lado, hay 29 trabajos que presentan recursos para los desarrolladores de Sistemas Colaborativos, como se muestra en la figura 3.2.

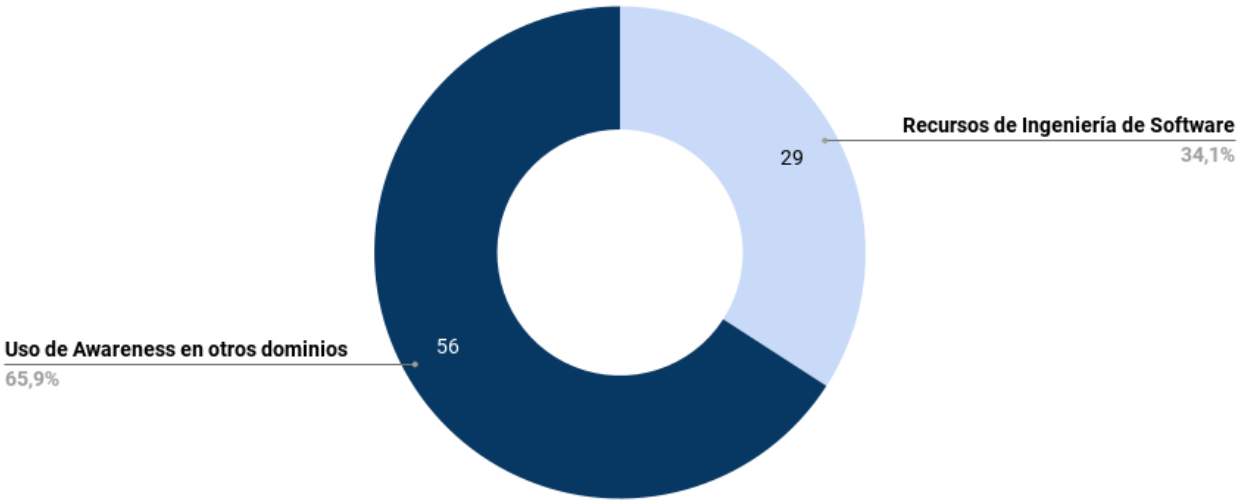


Figura 3.2: Trabajos Encontrados

Figura 3.3: Sin Recursos de Ingeniería de Software

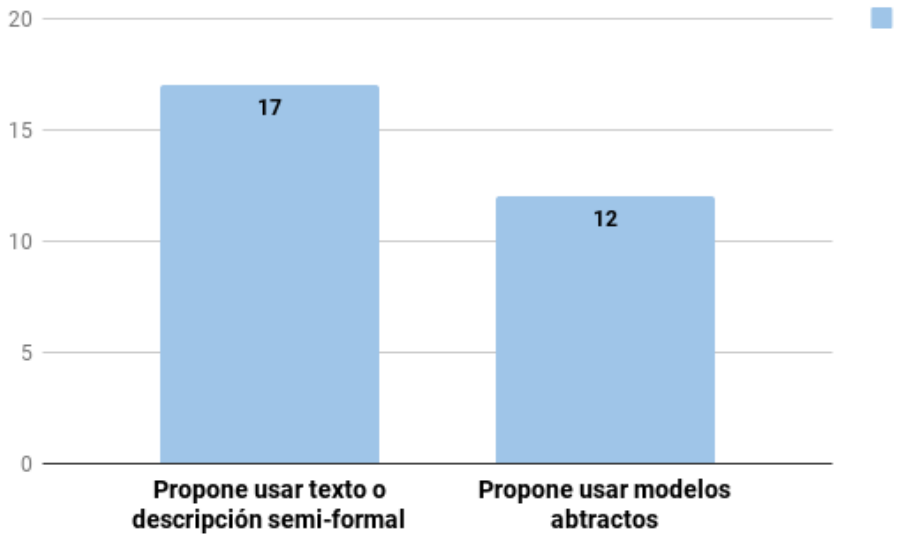


Figura 3.4: Con Recursos de Ingeniería de Software

Para los 56 trabajos que no presentan recursos en Ingeniería de Software, la clasificación es en tres grupos, como se muestra en la figura 3.3.

Se puso el foco en los 29 trabajos que presentan recursos de ingeniería de software, como se muestra en la figura 3.4, y se encontró 17 que proponen describir a los sistemas colaborativos con awareness de manera textual o de una manera informal o semi-formal. Entre ellos aparecen algunos checklist a tener en cuenta en la incorporación de awareness durante el análisis o desarrollo de los sistemas colaborativos, o guías para escribir los requerimientos. Incluso hay trabajos que presentan algoritmos para implementar la distribución de información, que involucra el awareness en los sistemas.

Finalmente se analizan los siguientes 12 trabajos que presentan algún modelo abstracto (ontología, metamodelos o modelos conceptuales) y que además incluyen awareness. Esto permite comprender la relación de este concepto, con los elementos principales de los sistemas colaborativos. También permite clasificar los distintos tipos de awareness que suelen aparecer en estos sistemas. [39] [40] [21]

Trabajos a Analizar

1. M. A. Teruel, E. Navarro, V. López-Jaquero, F. Montero, and P. González, “An extension ofi * to Model Requirements for CSCW Systems Applied to Conference Preparation Systemwith Collaborative Reviews,” in5th International i* Workshop (iStar’11), pp. 84–89, 2011.

2. M. A. Teruel, E. Navarro, V. López-Jaquero, F. Montero, J. Jaen, and P. González, “Analyzing the understandability of Requirements Engineering languages for CSCW systems: Afamily of experiments,”Information and Software Technology, 2012.

3. M. A. Teruel, E. Navarro, V. López-Jaquero, F. Montero, and P. González, “CSRMLTool: A visual studio extension for modeling CSCW requirements,” in-CEUR WorkshopProceedings, 2013.

4. M. A. Teruel, E. Navarro, V. López-Jaquero, F. Montero, and P González, “A CSCW Requirements Engineering CASE Tool: Development and usability evaluation, ”Information and Software Technology, vol. 56, no. 8, 2014.

5. M. A. Teruel, E. Navarro, V. López-Jaquero, F. Montero, and P González, “Comparing Goal-Oriented Approaches to Model Requirements for CSCW, ”in Evaluation of Novel Approaches to Software Engineering, pp. 169–184, Springer, Berlin, Heidelberg, 2013.

6. F. Belkadi, E. Bonjour, M. Camargo, N. Troussier, and B. Eynard, “A situation model tosupport awareness in collaborative design,” 2013

7. A. Kamoun, S. Tazi, and K. Drira, “FADYRCOS, a semantic interoperability framework for collaborative model-based dynamic reconfiguration of networked services, ”Computers in Industry, vol. 63, no. 8, pp. 756–765, 2012.

8. J. Gallardo, A. I. Molina, C. Bravo, M. A. Redondo, and C. A. Collazos, “An ontologicalconceptualization approach for awareness in domain-independent collaborative modelingsystems: Application to a model-driven development method,” inExpert Systems withApplications, 2011.

9. F. Gallego, A. I. Molina, J. Gallardo, and C. Bravo, “A conceptual framework for mo-deling awareness mechanisms in collaborative Systems,” inLecture Notes in ComputerScience (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes inBioinformatics), vol. 6949, pp. 454–457, 2011.

10. V. Vieira, P. Tedesco, and A. C. Salgado, “Using a metamodel to design structural andbehavioral aspects in context-sensitive groupware,” inProceedings of the 2010 14th Inter-national Conference on Computer Supported Cooperative Work in Design, CSCWD 2010,pp. 59–64, 2010.

11. J. Gallardo, C. Bravo, and M. A. Redondo, “A model-driven development method for collaborative modeling tools, ”Journal of Network and Computer Applications, vol. 35,pp. 1086–1105, 5 2012.

12. A. I. Molina, W. J. Giraldo, M. Ortega, M. A. Redondo, and C. A. Collazos, “Model-driven development of interactive groupware systems: Integration into the software development process,”Science of Computer Programming, vol. 89, pp. 320–349, 9 2014

3.1.1. Análisis de los Trabajos

Se considera primero un conjunto de trabajos (de los autores Teruel, Navarro, López-Jaquero, Montero y González), que presentan una notación para especificar los requerimientos de los sistemas colaborativos. Los trabajos comienzan con la presentación del lenguaje CSRML (Collaborative Systems Requirements Modeling Language) como una extensión de I* [41]. El Lenguaje presenta un conjunto de elementos básicos de los sistemas colaborativos (goal, role, actor, task, awareness) y relaciones entre ellos.

Los autores mencionados, continuaron durante los años 2012 y 2013 realizando diversos experimentos y comparaciones con diferentes técnicas Goal Oriented (NFR framework, i* and KAOS) de especificación de requerimientos para concluir que el CSRML era necesario [42–44], construyendo el Editor CSRML tool 2012, como una extensión de Visual Estudio [45].
En el 2014 realizaron un estudio de usabilidad del CSRML tool 2012 a través de un experimento con estudiantes, que utilizaron la herramienta para responder un cuestionario y así evaluar la

usabilidad del Editor.
La primera conclusión del análisis de los trabajos de estos autores es que CSRML no está basado en un metamodelo. Los conceptos abstractos que se describen son .elementosz relaciones". Los elementos incluidos en el lenguaje son: goal, task, role, resource y awareness cuando en general los sistemas colaborativos suelen incluir otros elementos básicos, como activity, tool y workspace.
El lenguaje CSRML propuesto, no tiene posibilidades de especificar procesos colaborativos. No es posible describir, qué tareas se pueden realizar en cada momento y tampoco se puede determinar qué sesiones o actividades colaborativas se pueden realizar antes que otras, o qué condiciones se pueden dar para que una sesión se active. Al no incluir procesos colaborativos, no se puede brindar información de awareness relacionada con ellos.
Respecto al CSRML tool 2012, se observa que la herramienta no está basada en estándares y no está integrada con el lenguaje estándar UML, lo que impide utilizar herramientas relacionadas con UML tales como plugins, editores o generadores de código.

El trabajo *A situation model to support awareness in collaborative design* [22], presenta una revisión de propuestas científicas que brindan diversas clasificaciones de awareness. Comenta distintas definiciones y concluye el análisis puntualizando las siguientes características:

- El awareness es un concepto multifacético. Distintos investigadores han propuesto clasificaciones de awareness ("social awareness", "mutual awareness", "task awareness", "workspace awareness", etc.)
- Un sistema colaborativo que provea soporte para awareness, deberá responder a un conjunto de requerimientos relacionados con una lista de preguntas que sirven para identificar distintos tipos definidos por el autor:
  - **social awareness:** Que se refiere a preguntas vinculadas con el usuario en relación a su pertenencia a distintos grupos. Entre las preguntas se encuentran: ¿Qué roles otros miembros del grupo pueden asumir? o ¿Cómo puedo interactuar con otros miembros del grupo?.
  - **task awareness:** Se refiere a preguntas relacionadas con el usuario y las tareas que realiza. Por ejemplo, el autor pregunta: ¿Qué pasos tengo que realizar para completar esta tarea? O ¿Cuánto tiempo se requiere para una tarea?.

- **workspace awareness:** Aquí el autor se refiere a qué preguntas sobre el usuario en relación a su ubicación dentro del sistema. Por ejemplo, ¿Quienes están aquí o en donde están los usuarios? O ¿Qué cambios hubo desde la última vez que estuve aquí?.

El trabajo define "situation/context awareness" como la suma de los tres tipos de awareness identificados. A partir de esta definición los autores analizaron propuestas de investigación y resumieron los siguientes conceptos claves frecuentes en los modelos de sistemas colaborativos con awareness:

- **context element:** Es modelado como un conjunto de elementos compartidos.
- **task and activity:** Esto describe qué se espera que se haga, o qué es lo que se está realizando para completar una interacción.
- **resource:** Describe un elemento del contexto. Este elemento es usado durante o para completar una interacción.
- **interaction:** Es la interacción entre sujetos y objetos usando herramientas colaborativas, o es la interacción social a través de la definición de reglas y división de tareas en la comunidad que se modela con un BPM (Business Process Modeling).
- **role:** En situaciones normales de trabajo colaborativo, el rol es entendido como el papel que le toca representar a un usuario. Por ejemplo, el "Líder" podrá participar de las reuniones de comité y podrá asignar tareas a otros roles.

Seguidamente el trabajo FADYRCOS (Framework for dynamic reconfiguration of networked collaborative systems) de Kamoun, Tazi, y Drira [46], tiene por objeto soportar colaboración en ambientes distribuidos, donde la actividad (session) puede ser implícita, y además incluir nuevos mecanismos para manejar la evolución de las sesiones y cambios de roles. Sus autores han desarrollado un framework que permite el manejo de la actividad y el desarrollo de componentes dinámicas para sistemas colaborativos, que cuenta con una arquitectura compuesta por los 3 niveles siguientes:

- **colaboration level:** El objetivo de este nivel es el de proveer un esquema de colaboración a los miembros que pertenecen a los distintos grupos, que se comunican dentro de una

sesión. Los elementos de este nivel están representados en el metamodelo genérico de colaboración (GCM). El principal concepto del metamodelo es la "Session", que representa una *actividad colaborativa*, y que cuenta con un conjunto de flujos de datos representados con el concepto Flow. Durante las actividades de colaboración los flows son intercambiados entre entidades representadas por el concepto de Node.

- **application level:** Este modelo se basa en los conceptos del dominio. En general este modelo es una especialización del GCM, donde se detallan los nodos que van a participar, los grupos que van a intervenir y las sesiones que existen en el dominio que se está modelando.

- **messaging level:** Para este nivel se cuenta con dos paradigmas de comunicación: el *Event Based Communication* y el *Peer to Peer Communication*.

El metamodelo tiene componentes abstractas como Nodes, Session (Activity) y Role/Group. Este modelo resulta interesante para el modelado de la actividad colaborativa y los mecanismos de interacción entre los diferentes roles y participantes. A través de un mecanismo de reglas, se van ejecutando transformaciones entre los niveles, utilizando reglas de refinamiento, selección y adaptación de contexto. El trabajo [46] no tiene posibilidades de especificar procesos colaborativos. Tampoco se puede determinar qué actividades colaborativas se realizan antes que otra, o qué condiciones se pueden dar para que una actividad se inicie.

Respecto al trabajo de Gallardo, Molina, Bravo, Redondo y Collazos [47], se presenta una ontología de awareness para el modelado de sistemas colaborativos. Primero introduce algunos fundamentos teóricos sobre sistemas colaborativos y awareness, a partir de una revisión de trabajos donde se destacan los trabajos previos de Gutwin y Greenberg [7, 20], y que resultan una de las contribuciones más destacadas en el campo de la teoría de awareness. Gutwin y Greenberg introducen un framework en [8], definiendo 10 elementos de awareness, donde cada uno responde a algunas preguntas que identifican el tipo que se necesita en cada caso. Las preguntas asociadas con cada tipo están basadas en la clasificación "Who, What, Where" de Gutwin y Greenberg y 6 elementos adicionales relacionados con acciones ya realizadas. Así surge la siguiente clasificación: location history, action history, presence history, event history. Previamente Gutwin y Greenberg habían definido otras clasificaciones como "informal awareness", "social awareness", "group-structural awareness" y "workspace awareness".

La ontología [47] modela los siguientes tipos de awareness:

- **group awareness:** Conciencia e información sobre diferentes aspectos del grupo y sus miembros.
- **workspace awareness:** Percepción de la interacción de los usuarios en el espacio de trabajo.

Luego presenta mecanismos para manipular el awareness, que son los siguientes:

- **awareness mechanism:** Mecanismos que informan la ocurrencia de una acción que provee el awareness.
- **access awareness mechanism:** Este mecanismo informa la acción de acceso de un usuario.
- **entity-relationship awareness mechanism:** Mecanismo que informa la acción sobre una entidad o sobre una relación.
- **communication/coordination awareness mechanism:** Mecanismo que informa la acción de comunicación o coordinación.

El trabajo [47] separa la problemática del awareness en los sistemas colaborativos en 2 aspectos. Por un lado, aparece el concepto con dos subclases "Workspace Awareness" y "Group Awareness". Por otro lado, introduce el concepto de "Mecanismo de Awareness" que representa el modelado de cómo el awareness va a ser manejado en relación a qué acción lo activa o lo dispara. Por ejemplo, cuando se quiere informar que un usuario accede a alguna sesión, se utiliza el *Access Awareness Mechanism* para informar a los usuarios involucrados.

Luego, el artículo de Gallego, Molina, Gallardo y Bravo [48] es una continuación del trabajo de Ontología anterior, y propone un metamodelo de awareness que define el *Group Awareness* y el *Workspace Awareness*. También tiene un conjunto de *Awareness Support Widgets* para soportar estos tipos de awareness.

En los trabajos [47] y [48] no se menciona cómo relacionar el awareness con los conceptos básicos de los sistemas colaborativos (Workspace, Activity, Collaborative Process, Tool, etc.). También

hay ciertos tipos de awareness que no son soportados por el metamodelo (como por ejemplo, los awareness relacionados con las actividades que realizan los usuarios, o poder mostrar el grado de avance de una tarea colaborativa.)

Finalmente, el trabajo de Vieira, Tedesco y Salgado [49], presenta un metamodelo llamado Context Metamodel. Aunque no es un trabajo específico de awareness, el metamodelo propuesto incluye algunos conceptos interesantes para especificar el awareness en los sistemas colaborativos. El metamodelo tiene como clase principal la clase ContextualElement. También tiene un conjunto de enumerativos donde se encuentra ContextType que tiene los tipos de awareness que propone Gutwin y Greenberg "who" "what" "where" "when" "why". Otro enumerativo es el UpdateType que determina cuando se actualiza el awareness y el AcquisitionType para detallar el mecanismo por el cual se adquiere la información de awareness. El metamodelo propuesto en [49] no tiene posibilidades de especificar procesos colaborativos; por lo tanto no se puede brindar información de awareness relacionada con ellos.

En el trabajo de Briggs et al, [5] se presenta la idea de thinklet como patrones de diseño para prácticas de trabajo colaborativo. Los desarrolladores/diseñadores de actividades colaborativas, utilizan thinkLets como bloques de construcción para crear diseños de procesos de colaboración reutilizables, que se transferirán a los profesionales para que los ejecuten por sí mismos sin la intervención continua de facilitadores. La colección de thinkLets forma un lenguaje de patrones para crear, documentar, comunicar y aprender diseños de procesos de grupos. Cada thinkLet debe especificar un conjunto de reglas que prescriban las acciones que las personas en diferentes roles deben realizar utilizando las capacidades que se les proporcionan, bajo algún conjunto de restricciones especificadas en los parámetros. Aunque la propuesta de utilizar thinklets (Briggs, Gert-Jan de Vreede y Kolfschoten 2007) es interesante para documentar la dinámica de la colaboración, no se especifica las herramientas que se utilizan en la actividad, los espacios donde se realizan y carece de elementos para modelar el awareness asociado a los elementos del sistema.

En el trabajo de Gallardo et al. [39] se propuso un método basado en modelos para el desarrollo de herramientas de modelado colaborativo independientes del dominio. Este método consta de un marco metodológico, un marco conceptual y un marco tecnológico. El marco metodológico define las fases a realizar en la aplicación del método, mientras que el marco conceptual está formado por los metamodelos utilizados en el método y los procesos de transformación

que se establecen entre ellos. Finalmente, el marco tecnológico consiste en la integración de algunos complementos del Eclipse Modeling Project con algunos complementos que brindan funcionalidad colaborativa. El trabajo propone un diseño de dominio dentro del marco conceptual, describiendo los elementos atómicos a editar de forma colaborativa. Las herramientas colaborativas que se utilizarán en el sistema son algunos editores específicos y algunas otras herramientas clásicas de los sistemas colaborativos (es decir, chat o noticias). Sería deseable integrar cualquier tipo de herramienta colaborativa en el diseño, describiendo qué operaciones pueden ejecutar los roles en cada momento. También, se necesita poder diseñar relaciones entre espacios de trabajo como la inclusión entre espacios, para cubrir varios dominios colaborativos (e-learning, lluvia de ideas, juegos colaborativos, etc.). También es necesario integrar un diseño específico de la información de awareness asociada a los diferentes elementos del modelo.

Finalmente, en el trabajo de Molina [50] se analizaron los principales marcos conceptuales y se especifican los conceptos que generalmente se utilizan. (Actividad, Acción/Operación, Usuario, Grupo, Rol, Herramienta, Objeto Compartido, Reglas de Awareness). Además, se menciona que la modelación de la cooperación implica la inclusión de tareas especiales de coordinación a la actividad colaborativa, para que el grupo pueda recoger sus aportes individuales en el producto final, así como la toma de decisiones o acuerdos en este proceso de producción. En este último caso, se habla de la existencia de protocolos de interacción y coordinación entre los miembros del grupo y llegan a la siguiente conclusión:

- El lenguaje debe soportar el modelado de diferentes niveles de abstracción y, en particular, la descomposición de tareas en subtareas.
- Debe poder especificar aspectos de coordinación (uno de los conceptos básicos en los sistemas CSCW).
- Los modelos que especifican el trabajo a realizar permitirán modelar el flujo de trabajo.

El trabajo de Molina presenta la sintaxis abstracta de la notación CIAN (Collaborative Interactive Application Notation) y proporciona un marco conceptual que abarca los elementos previamente identificados como relevantes para modelar aplicaciones de software colaborativo. Al igual que el trabajo en (Gallardo J et al), se apoya un modelo que describe los elementos atómicos a editar de forma colaborativa; esto puede ser útil en algunos dominios, pero restringe la variedad de herramientas potenciales que se utilizarán.

A partir de estos trabajos se puede analizar los resultados de la revisión.

3.2. Resultados Obtenidos

Los trabajos analizados en la sección anterior utilizan en general los mismos conceptos para describir los sistemas colaborativos. Estos son "Shared Object", "Tool", "Collaborative Activity", "Workspace", "User/Role", "Group" y también tienen relación con el concepto de awareness. En [22] se utiliza una de las preguntas para identificar el awareness: "What roles will the other members of the group assume?", en donde aparece el concepto de Rol asociado al awareness. También hay otros ejemplos de awareness donde se pregunta: How can I help other participants to complete the project? or What are they doing? or where are they?. También la mayoría de los trabajos incorporan el concepto de "Task", que representa las actividades colaborativas que tienen que realizar los usuarios. Sin embargo, los trabajos no modelan procesos colaborativos que incluyan esas actividades y que permitan darle un orden lógico a las mismas.

3.2.1. Definiciones y conceptos básicos: Glosario

A partir de los trabajos analizados en la sección anterior, se enumeran algunas definiciones y conceptos básicos con el objeto de formar un vocabulario común, que guíe la lectura del documento y, a la vez, contribuya a caracterizar los sistemas de groupware.

- **Sistema Colaborativo/Groupware:** son aplicaciones, para grupos u organizaciones, que surge de la unión de computadoras, de grandes bases de información y de tecnologías de comunicación. Es una tecnología diseñada para facilitar el trabajo en grupo [1,2,4]. Se puede usar para comunicar, cooperar, coordinar, resolver problemas, competir o negociar. De acuerdo a Ellis et al. [1] se enuncia la siguiente definición:

  Definición:

  Sistemas basados en computadoras que soportan grupos de personas involucradas en una tarea común (u objetivo) y que proveen de una interfaz a un ambiente compartido.

Las nociones de una “tarea común” y “ambiente compartido”, son cruciales en esta definición. A partir de ellas, se excluyen, por ejemplo, a los sistemas multiusuarios, donde los usuarios comparten algunos recursos – como espacio en disco, tiempo de procesamiento, memoria, etc. – pero no comparten en realidad una tarea común.

Esta tecnología brinda un equilibrio entre el trabajo individual de los participantes y las actividades que realizan los usuarios para lograr un objetivo grupal. En definitiva, una efectiva coordinación, mejora las posibilidades de colaboración ordenando la participación de los usuarios. Esto se refleja en procesos colaborativos que determinan en qué orden se llevan adelante las actividades del grupo y los protocolos, para definir qué acciones concretas pueden realizar los roles en cada actividad. Para lograr una efectiva colaboración, los usuarios tienen que estar informados sobre las acciones que los otros participantes realizan en el ambiente y cómo esas acciones afectan el entorno de trabajo. Esta información que brinda el sistema se llama awareness.

En un primer acercamiento a los sistemas groupware se pueden distinguir, por un lado, las aplicaciones donde los usuarios realizan actividades en forma simultánea (sincrónica) llamadas “real time groupware”. Y por otro lado están aquellas en las que los usuarios realizan sus actividades en forma asincrónica y se denominan “non real time groupware. Se analiza a continuación un glosario con los conceptos más comunes de los sistemas colaborativos.

- **Herramientas colaborativas (Tool):** La colaboración entre los usuarios se realiza a través de herramientas donde se manipulan los objetos compartidos que son los productos que se obtienen como resultado de la colaboración. Las herramientas son programas que, a diferencia de las aplicaciones monousuario, atienden a un grupo de personas. Ejemplos de herramientas colaborativas son: chat, pizarra compartida, editores colaborativos, etc. Éstas pueden ser utilizadas de diferente manera, por ejemplo, en el chat, los usuarios podrían enviar mensajes al mismo tiempo, u ordenados por un moderador, o por turnos. La forma en que se usan las herramientas se verá más adelante y se modela con el concepto de protocolo. En definitiva, una herramienta puede ser usada en con distintos protocolos.

- **Objetos colaborativos (Shared Object):** Son los elementos creados por los usuarios que se manipulan por las herramientas colaborativas. De esta manera, un usuario puede utilizar elementos creados por otros usuarios. Por ejemplo:

- En un ambiente de educación a distancia, un alumno puede editar elementos colocados por un compañero o el tutor, si el sistema lo permite.
- Cuando dos o más usuarios utilizan el chat, su conversación es un objeto colaborativo que puede ser usado por otras personas como material de discusión.
- Cuando dos usuarios editan un mismo documento, el contenido del documento es el objeto compartido

La particularidad de estos objetos es que son editados por distintos usuarios y dependiendo de la herramienta y del protocolo, la edición puede ser en forma sincrónica (al mismo tiempo) o asincrónica. Los objetos colaborativos más comunes en los ambientes groupware son documentos de texto, dibujos, páginas web o conversaciones en chats, pero no se descartan objetos más complejos como diseños CAD, o diagramas de E/R, etc.

- **Actividad colaborativa (Collaborative Avtivity / Session):** Es una actividad desarrollada por un grupo de usuarios. Puede involucrar una simple charla (usando un chat) de un par de usuarios o escribir una enciclopedia (usando una wiki) con cientos de usuarios. Analizando las actividades grupales presenciales, por ejemplo, en una clase tradicional, los participantes tienen que estar en un espacio (un aula), al mismo tiempo, donde los participantes tienen roles definidos (por ejemplo, docentes, alumnos, etc.), donde hay algunas herramientas o recursos que se utilizan (pizarra, mapas, etc.) y algunos roles tiene distintas posibilidades de comunicar cosas o escribir en la pizarra. Si se toma una clase como actividad colaborativa en un sistema colaborativo, se puede decir que habrá algunos aspectos similares a la clase tradicional. Seguramente participarán usuarios con los roles de docente y alumno. Usando el sistema, los usuarios ingresarán a algún espacio virtual que represente el aula y de acuerdo a las posibilidades de cada rol, podrán utilizar alguna herramienta colaborativa para escribir en alguna pizarra o transmitir un video o audio. Las actividades colaborativas son en definitiva lo que hacen los usuarios en el sistema. No son las herramientas ni tampoco los espacios donde se reúnen. Por ejemplo, una Clase (vista como una actividad colaborativa) se lleva a cabo en un Aula (visto como un espacio donde los usuarios se encuentran para colaborar) donde los roles de usuarios utilizan algunas herramientas colaborativas, como puede ser una pizarra, un foro, una wiki o una herramienta de videoconferencia. Las actividades colaborativas sirven para vincular estos elementos de la colaboración (espacios, herramientas y roles).

- **Rol (Role):** El rol es un concepto que separa al usuario (identidad) de lo que representa en el sistema (más conocido como el rol que juega en algún momento determinado). Un usuario puede tener distintos roles (por ejemplo, puede en algún momento ser "autor" de un documento y en otros casos puede ser "lector" de otro documento). También puede ser "docente" en algún curso y "alumno" en otro. Para definir un rol, se describen las operaciones que puede realizar con las herramientas con las que cuenta el sistema. Es importante tener en cuenta que las operaciones que ejecutan los roles pueden afectar el estado de la colaboración. En el caso de una actividad colaborativa, donde participan dos roles, un docente y un alumno, cuando el docente está exponiendo, el alumno puede levantar la mano. El alumno podrá participar cuando el docente le otorgue la palabra. Más adelante se verá que estas operaciones se utilizarán para avanzar en los procesos colaborativos y protocolos.

- **Espacio de trabajo (Workspace):** Es el lugar donde se realiza la colaboración y sirve para contextualizar el tipo de colaboración que se lleva a cabo [Using a Room Metaphor to Ease Transitions in Group-ware]. Por ejemplo, si se toma al aula virtual como un workspace, se entiende qué tipo de actividades se van a realizar, qué roles intervienen, qué tipo de herramientas se van a usar y qué protocolos de colaboración se usará. Para detallar este ejemplo se entiende que en las aulas se van a realizar actividades de clase al estilo tradicional y de consulta. En las *clases* se pueden usar herramientas de vídeo conferencia y foro y en las *consultas* se pueden usar el mail y el chat. Los roles que intervendrán podrían ser: docente, tutor y alumno. Finalmente se puede describir protocolos diferentes para la clase y para la consulta. En la clase, la maestra podría coordinar la participación de los alumnos, dándoles la palabra en el caso que ellos la soliciten y en las consultas, los tutores podrían tener las mismas atribuciones que los alumnos al momento de participar en la colaboración. En definitiva, dentro de un workspace hay herramientas que son utilizadas para comunicarse y trabajar en los objetos compartidos. Los protocolos estructuran las interacciones de los roles en el workspace y el uso de las herramientas por ellos. En general, un workspace o grupo de workspaces, no son suficientes para estructurar una aplicación colaborativa. En el ejemplo, se ve al aula como un workspace, pero probablemente haya otros espacios como la biblioteca o la cafetería, que brindarán seguramente alternativas de colaboración. Usualmente un workspace es definido dentro de un entorno más grande que lo contiene, que se llama "entorno colaborativo".

- **Protocolo (Protocol):** Dentro de las actividades colaborativas, se produce la interacción de los participantes. El protocolo es el modelado de dicha interacción y sirve para modelar, guiar y estructurar el proceso social que se lleva a cabo dentro del grupo. Los protocolos definen qué herramientas y objetos colaborativos pueden ser utilizados en cada momento de una actividad colaborativa por los roles. En este caso, el protocolo guía u ordena la participación de los usuarios y puede determinar qué operaciones los usuarios pueden ejecutar en distintos momentos de la actividad. Se destaca el protocolo básico, que es el que permite que los participantes colaboren en cualquier momento y se lo denomina "no-protocol".

- **Proceso Colaborativo (Collaborative Process):** Es un conjunto de actividades colaborativas planificadas para que los usuarios lleguen a un objetivo grupal. El proceso es especificado como un grafo de nodos interconectados por ejes. Los nodos pueden ser actividades colaborativas y nodos de control (initial, fork, join, decision, final) que permiten modelar procesos con loops y actividades que se pueden realizar en paralelo y otras que deberán esperar a que alguna otra actividad finalice.

- **Awareness:** Es la información que el sistema provee sobre el estado de la colaboración. En las reuniones presenciales estar al tanto de los otros es algo natural. Se puede percibir dónde está ubicado cada uno, cuál es su estado, qué actividad está desarrollando y con qué objeto. Por el contrario, mantener actualizada esta información en sistemas colaborativos, es bastante difícil. Es por esto que los primeros sistemas, que no mantenían esta información, resultaban confusos e ineficientes comparados con el trabajo cara a cara.

- **Entorno Colaborativo (Collaborative Environment):** Es un conjunto de workspaces. A menudo, los sistemas colaborativos complejos necesitan más de un workspace, de una manera similar a una universidad virtual que estará compuesta por distintos workspaces como el aula, la biblioteca, etc. Los entornos definen el acceso y el uso de los diferentes workspaces por parte de los distintos roles existentes. Por ejemplo, definir que algunos roles no podrán ingresar a algunos workspaces o que al cambiar de workspace el rol cambia. Los entornos colaborativos pueden contener otros entornos colaborativos de manera tal, que se puedan crear ambientes más complejos de colaboración, como por ejemplo, se podría tener un sistema colaborativo que representa a una Ciudad que contiene a una Universidad Virtual Colaborativa y a un Museo Colaborativo.

3.2.2. Requisitos para el modelado conceptual de Sistemas Colaborativos con Awareness

Teniendo en cuenta el glosario de la sección anterior y de la revisión sistemática [37] se concluye que son pocos los trabajos que presentan modelos abstractos para que los diseñadores de software incluyan el concepto de awareness en sistemas colaborativos. Una de las primeras observaciones realizadas, indica que los trabajos analizados en este capítulo utilizan los mismos conceptos principales que son los siguientes: "Shared Object", "Tool", "Actividad", "Workspace", "User/Role", "Group" lo que permite enunciar el primer requisito para un modelo conceptual para sistemas colaborativos (groupware).

Requisito 1:

Contar con un modelo conceptual que contenga los conceptos que intervienen en los sistemas colaborativos.

Continuando con el análisis, se ve que estos elementos tienen relación con el concepto de awareness. Por ejemplo, una de las preguntas que se utilizan en el trabajo de Belkadi et al. [22] para identificar un awareness específico es: "What roles will the other members of the group assume?", en donde aparece el concepto de "Role". Otro ejemplo: el awareness relacionado con el espacio de trabajo donde el autor se pregunta: How can I help other participants to complete the project? or What are they doing? or where are they?; donde vincula el espacio con el awareness. Con esta observación, se da origen al segundo requisito para el modelado de sistemas colaborativos.

Requisito 2:

Contar con un modelo que permita expresar el awareness vinculado a los conceptos que intervienen en el sistema. El modelo tiene que cubrir las distintas alternativas que presenta el awareness (workspace awareness, social awareness, group awareness, entre otras).

Las actividades colaborativas que realizan los usuarios en el sistema, según los trabajos analizados, aparecen con el concepto de "Task" o "Activity". Estas requieren algunos recursos de ingeniería de software para poder modelarlas correctamente. Se necesita algún mecanismo para darle un orden lógico a las mismas. Por ejemplo, indicar qué tarea se realiza antes que otra o

si algunas de ellas se pueden ejecutar en paralelo. Esto origina el tercer requisito.

Requisito 3:

Contar con un modelo de proceso colaborativo, como un conjunto de actividades colaborativas que desarrollan los usuarios.

A partir de este requisito es necesario contemplar el awareness relacionado con los procesos colaborativos. Por ejemplo, informar el grado de avance que un grupo de usuarios tiene en un proyecto o qué tareas están en curso y cuáles están por iniciarse, podrían ser información de awareness interesante de mostrar. Para ello, es necesario que el modelo especifique distintos tipos de awareness y los relacione con los procesos colaborativos, dando así origen al cuarto requisito.

Requisito 4:

Permitir asociar información de awareness a los procesos colaborativos.

Es sabido que las actividades colaborativas permiten que los usuarios colaboren utilizando distintas herramientas y objetos compartidos. Las actividades no son estáticas, sino que van cambiando de estado a medida que los usuarios van realizando sus acciones (por ejemplo, en una sesión donde un usuario está exponiendo un tema, otro usuario puede querer hacer preguntas.) Si el sistema permite que los usuarios pidan autorización para preguntar, entonces el expositor podrá dar la palabra a quien la haya pedido, o continuar con la exposición. En consecuencia, la actividad está en un estado cuando el expositor tiene la palabra y en otro, cuando otro usuario está preguntando. El modelado de estos diferentes estados describe cómo el sistema se va modificando a partir de la interacción de los usuarios. De esta forma, se arriba al enunciado del requisito quinto.

Requisito 5:

Permitir modelar los distintos estados por lo que pasa una actividad colaborativa (protocolos de interacción)

A partir de este requisito, es necesario modelar el awareness relacionado a los distintos estados de la actividad colaborativa. En el ejemplo anterior, cuando un usuario pide la palabra, el sistema debería informar a los otros participantes que alguien quiere preguntar. Cuando el expositor autoriza a preguntar, los otros usuarios deberán recibir información de awareness

que les indique qué usuario está preguntando. Esta dinámica de cambios de estado genera la necesidad de expresar awareness específicos asociados a los estados de la actividad. Como consecuencia de esto, se llega al requisito sexto.

Requisito 6:

Incorporar distintos tipos de awareness, asociados a los estados por lo que pasa una actividad colaborativa (session).

Finalmente, en el desarrollo de software, es importante el uso de estándares industriales. Estos facilitan tanto la comunicación entre desarrolladores como la interacción entre diferentes aplicaciones y productos basados en ellos, que propician el mejoramiento de la calidad tecnológica. Por ejemplo, si el modelo se basa en el metamodelo de UML, se podrían utilizar distintas herramientas (plugins) basados en dicho metamodelo. En consecuencia, para describir el sistema colaborativo, podrían ser útiles editores relacionados con UML; lo cual da origen al requisito séptimo.

Requisito 7:

Utilización de estándares en el desarrollo de modelos que intervienen en la construcción de sistemas colaborativos con awareness (groupware).

Contar con métodos, procesos o lenguajes que guíen la construcción de sistemas colaborativos que cuenten con funcionalidad de awareness, es fundamental para sustentar las prácticas repetibles, reusables y técnicas, que organizan el desarrollo y mejoran la calidad de los productos. En esta línea, se elaboró el metamodelo que se presenta en el siguiente capítulo.

3.3. Conclusiones del capítulo

En el capítulo se analizaron propuestas científicas que presentan recursos de ingeniería de software para modelar sistemas colaborativos con awareness. La mayoría de los trabajos muestran ejemplos de uso de aplicaciones que utilizan la información de awareness en algún dominio en particular (educación, salud, etc.). Hay pocos trabajos que abordan el modelado del awareness a través de modelos conceptuales, abstracciones o metamodelos.

Una de las carencias de los trabajos analizados es la falta de elementos que permitan modelar procesos colaborativos. Algunos permiten incorporar tareas aisladas, pero no es posible darle un orden lógico a las mismas. En consecuencia, el requisito 3 enuncia la necesidad de incluir el modelado de procesos colaborativos.

Por otro lado, se requiere modelar la interacción de los participantes dentro de las tareas colaborativas. Particularmente se necesita expresar qué acciones pueden realizar los roles en cada momento. Esto fue enunciado en el requisito 5, brindando la posibilidad al diseñador de definir diferentes estados por los que pasan las tareas colaborativas. Estos estados y el cambio entre ellos necesitan mantener a los participantes informados (awareness) sobre la dinámica de cada tarea.

El análisis realizado en este capítulo establece las bases para la definición de un modelo conceptual que cumpla con los requisitos enunciados. A partir de este modelo se definirá un lenguaje específico de dominio que permita describir sistemas colaborativos con awareness de manera precisa y efectiva.

Capítulo 4

Metamodelo de Sistemas Colaborativos

4.1. Metamodelado

En la ingeniería de software, se trabaja con diferentes modelos y de diferentes niveles de abstracción (análisis, diseño, implementación), representando diferentes aspectos del sistema (base de datos, lógica del negocio, interfaz con el usuario, etc.), diferentes requisitos (seguridad, desempeño, flexibilidad), o diferentes tareas (modelos de pruebas, modelos de emplazamiento), como se menciona en [29]. En muchos casos, es posible generar un modelo a partir de otro (por ejemplo, pasando del modelo de análisis al modelo de diseño, o del modelo de la aplicación al modelo de pruebas.)

Existen varias formas de distinguir entre tipos de modelos, cada una de ellas se basa en la respuesta a alguna pregunta acerca del modelo, por ejemplo:

- ¿En qué parte del proceso de desarrollo de software es usado el modelo?
- ¿El modelo es abstracto o es detallado?
- ¿Qué sistema es descrito por el modelo? ¿Es un modelo del negocio o es un modelo de software?
- ¿Qué aspectos del sistema son descritos por el modelo? ¿Es un modelo de la estructura o es un modelo del comportamiento?

- ¿Es un modelo orientado a una tecnología específica o es un modelo independiente de la tecnología?

El uso de modelos es cada vez más recomendado en proyectos de ingeniería de software. Esto debe ser contrastado con las técnicas de desarrollo de software clásicas, donde los desarrolladores principalmente trabajan escribiendo código en lenguajes de programación. Por un lado, existen modelos informales, que son expresados utilizando lenguaje natural, figuras, tablas u otras notaciones. Por otro, modelos formales, cuando la notación empleada es un formalismo, es decir posee una sintaxis y semántica (significado) precisamente definidos.

Los modelos se definen utilizando lenguajes de modelado. Dichos lenguajes de modelado a su vez están definidos mediante otro modelo que se denomina metamodelo. La construcción de un metamodelo se realiza utilizando un lenguaje de modelado que tiene las siguientes características:

- El lenguaje de modelado debe permitir especificar la sintaxis abstracta del lenguaje, es decir que permita definir su metamodelo, indicando cuáles son los elementos del lenguaje y sus relaciones, así como también las propiedades para cada elemento.
- Debe permitir definir restricciones para que las instancias de este nuevo lenguaje estén correctamente formadas. Por lo tanto, es preciso que el lenguaje de modelado permita especificar reglas básicas como, por ejemplo, qué objetos se pueden conectar a través de cuál relación.
- Para que este lenguaje sea más amigable al usuario, sería deseable que se pudiera instanciar de manera gráfica. Para esto, debe permitirse que en su definición sea posible especificar símbolos gráficos para los elementos, de manera gráfica, declarativa o en código. Debe ofrecer al menos la funcionalidad básica esperada de cualquier herramienta de edición. Estas funciones incluyen almacenar y recuperar un modelo del disco, deshacer y rehacer de muchos niveles, cortar, copiar, pegar, borrar. También deberían permitir manipular directamente los elementos a editar, imprimir y exportar en varios formatos, distribuir los elementos en el diagrama de manera automática, visualizar en distinta escala (zoom).
- En algunos casos, las herramientas que soporta el lenguaje requieren que el usuario que crea el metamodelo (el metamodelador) especifique los iconos, el tipo de paleta y a veces

los menús en forma manual (incluso en algunas herramientas se pedía al metamodelador, que cree iconos de diferentes tamaños para diferentes usos, plataformas o configuraciones de la pantalla). Para incrementar la productividad del metamodelador es necesario automatizar estas tareas, por ejemplo, brindarle la posibilidad de que un icono escale automáticamente, a una variedad de tamaños. Generar todos estos elementos automáticamente asegura que todas las partes de la herramienta de modelado permanezcan sincronizadas con la definición del lenguaje de modelado. Por lo tanto, sería deseable que la herramienta proveyera una definición automática de los iconos, la paleta, los menús y los diálogos.

- Actualmente la probabilidad de trabajar con una sola herramienta en el desarrollo de un software es baja. Por lo tanto, debe ser posible el intercambio de metamodelos y modelos, es decir la importación y exportación de modelos y metamodelos, para el intercambio de información entre otras instancias de la herramienta y de otras herramientas.

- Es muy común que a medida que se use un lenguaje, se noten deficiencias que hacen necesario cambiar su definición. Por lo tanto, es muy útil que la herramienta permita modificar los metamodelos cuando existen instancias de estos, actualizando sus instancias automáticamente. Esto permite que el metamodelador trabaje para perfeccionar la definición de su metamodelo, mientras construye el lenguaje, y que aún antes de haber terminado los modeladores ya puedan utilizarlo.

Un modelo formal se ajusta a un metamodelo único. El enfoque MDA de la OMG (Object Management Group), descrito previamente, utiliza un lenguaje para escribir metamodelos llamado Meta Object Facility o MOF [27]. Este metalenguaje se ha usado para definir formalmente los metamodelos de lenguajes como UML [51], SysML [52], SPEM [32] o CWM.

Un metamodelo define qué elementos pueden existir en un modelo. Por ejemplo, el metamodelo de UML define que se puede usar los conceptos de “Class”, “State”, “Package”, etc. en un modelo UML. Los Metamodelos son también modelos gráficos, por ello un metamodelo en sí mismo tiene que ser escrito en un lenguaje bien definido. Este lenguaje se lo denomina metalenguaje. En teoría, se puede encontrar un metamodelo de un metamodelo, que se denomina meta-metamodelo y así en forma infinita. Pero la OMG usa cuatro niveles M0, M1, M2, M3 que están representados en la Figura 4.1. El éxito de los lenguajes gráficos de modelado, tales

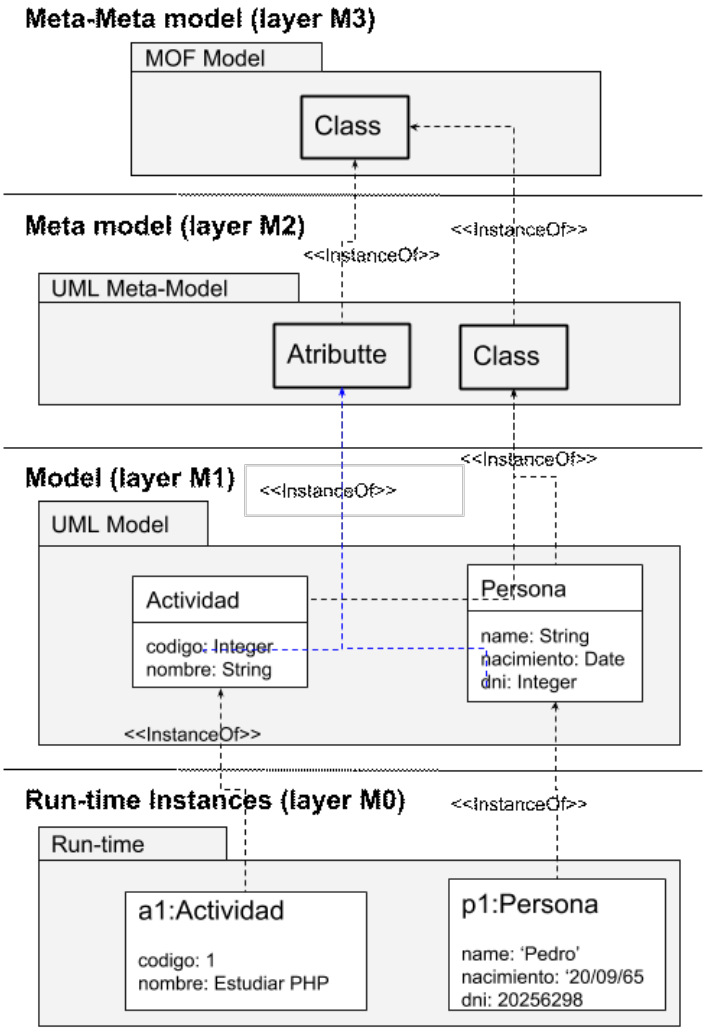


Figura 4.1: Niveles de modelado definidos por la OMG

como el Unified Modeling Language (UML) [51] se basa principalmente en el uso de construcciones gráficas que transmiten un significado intuitivo (por ejemplo, un cuadrado representa un objeto, una línea uniendo dos cuadrados representa una relación entre ambos objetos). Estos lenguajes resultan atractivos para los usuarios, ya que aparentemente son fáciles de entender y aplicar. Sin embargo, la falta de precisión en la definición de su semántica puede originar malas interpretaciones de los modelos, inconsistencia entre los diferentes sub-modelos del sistema y discusiones acerca del significado del lenguaje.

Por otro lado, en el capítulo anterior se concluyó en la necesidad de contar con modelos que permitan expresar los conceptos que aparecen en los sistemas colaborativos, que permitan modelar procesos, protocolos y awareness. Por todo esto, se pensó en extender UML para cubrir con estos requisitos planteados. En el trabajo de Lidia Fuentes y Antonio Vallecillo [53]

se enumeran algunas razones por las que un diseñador puede querer extender y adaptar un metamodelo existente UML 2.0, por ejemplo:

- Disponer de una terminología y vocabulario propio de un dominio de aplicación o de una plataforma de implementación concreta.
- Definir una nueva notación para símbolos ya existentes, más acorde con el dominio de la aplicación objetivo.
- Añadir cierta semántica que no aparece determinada de forma precisa en el metamodelo.
- Añadir restricciones a las existentes en el metamodelo, guiando su forma de utilización.
- Añadir información que puede ser útil a la hora de transformar el modelo a otros modelos, o a código.

UML incluye mecanismos de extensión en el propio lenguaje, que permite definir lenguajes de modelado que son derivados de UML. Uno es la creación de un Perfil, como se explicó en [53], que se define en un paquete UML, llamado «profile», que permite extender a un metamodelo o a otro Perfil. Tres son los mecanismos que se utilizan para definir Perfiles: estereotipos (stereotypes), restricciones (constraints), y valores etiquetados (tagged values). Es importante señalar que estos tres mecanismos de extensión no son de primer nivel, es decir, no permiten modificar metamodelos existentes, sólo añadirles elementos y restricciones, pero respetando su sintaxis y semántica original. Sin embargo, sí que son muy adecuados para particularizar un metamodelo para uno o varios dominios o plataformas existentes. Cada una de estas particularizaciones o adaptaciones viene definida por un Perfil, que agrupa los estereotipos, restricciones, y valores etiquetados propios de tal adaptación. Se puede decir que: utilizar perfiles para extender un metamodelo implica aplicar un mecanismo de extensión ligera.

Otro mecanismo es el Metamodelado, que se lo recomienda cuando se quiere desarrollar una sintaxis particular para un dominio, por ejemplo, Los Sistemas Colaborativos con Awareness. Asimismo se busca elaborar un vocabulario y una terminología propia y una sintaxis específica. También se puede definir una nueva notación para símbolos de UML ya existentes, por ejemplo, procesos y máquinas de estado más acorde con el dominio de la aplicación. Se puede añadir, de esta forma, cierta semántica que no aparece determinada de forma precisa en el metamodelo original, por ejemplo, obtener versiones ejecutables de los modelos que se elaboren.

Es por ello, que se tomó la decisión de utilizar el mecanismo de metamodelado para elaborar un metamodelo y construir un lenguaje específico (DSL - Domain Specific Language) que permita expresar una sintaxis amigable que describa los modelos de sistemas colaborativos, y que se presenta en la próxima sección.

4.2. CSSL - Collaborative Software System Language

Un modelo de un sistema colaborativo con awareness, es una descripción del sistema escrito en un lenguaje bien formado, generalmente gráfico, el cual está preparado para ser interpretado automáticamente por un programa. En este trabajo, se presenta el metamodelo de CSSL (Collaborative Software System Language) que es una abstracción que permite expresar los conceptos que intervienen en los sistemas colaborativos.

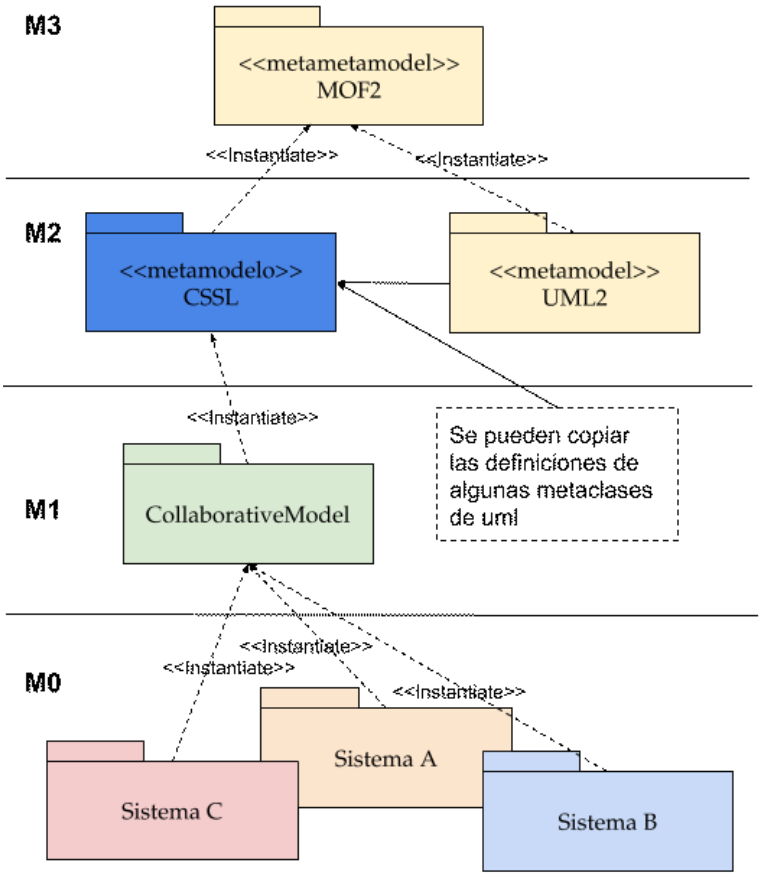


Figura 4.2: Los 4 niveles de modelado de la OMG

4.2.1. Características del Metamodelo

Las siguientes son las ventajas de contar con un metamodelo para el lenguaje de modelado.

- Permite definir la sintaxis abstracta y semántica de los elementos del dominio, facilitando el entendimiento y comunicación de los modelos.
- Permite el intercambio de modelos entre las herramientas de modelado.
- Permite la representación de elementos del dominio específicos.

Para la construcción del metamodelo, usando la técnica de metamodelado, existen dos alternativas. Se podrían crear metaclases a partir de MOF2, de la misma forma como fueron creadas las metaclases de UML (se muestra en la figura 4.2). También se podría imitar la creación de algunas metaclases, similares a las de UML, que podrían ser útiles en los diseños. Por ejemplo, se podría crear en el metamodelo las metaclases Class y Association, y se podría instanciar clases y asociarlas en los modelos de sistemas colaborativos.

Sin embargo, se analizó que se necesitaban imitar numerosas metaclases de UML (*Activity, Event, Operation, Class, StateMachine, Association, etc.*) ya que eran necesarias para cubrir los requerimientos de los Sistemas Colaborativos [37]. En consecuencia, se decidió construir el metamodelo CSSL, como una extensión de UML. Es decir que el lenguaje CSSL contiene completamente a UML, como se muestra en la figura 4.3. Esto permite construir modelos colaborativos usando tanto metaclases del lenguaje CSSL como cualquier metaclase de UML. También se pueden utilizar los diagramas de UML y todas las herramientas desarrolladas para UML, por ejemplo, Editores de UML (tanto de gráficos o texto), Mapeadores de UML a código, etc. Por otro lado, hay que tener en cuenta que UML cuenta con un conjunto de diagramas que también se podrían utilizar, como se muestra en la figura 4.4, que permiten describir tanto la estructura del sistema, como el comportamiento del mismo. Dentro de los diagramas de estructura, se destacan los diagramas de clase, de paquetes y de despliegue (deploy), que suelen ser los más utilizados. Dentro de los diagramas de comportamiento, se destacan entre otros: el diagrama de casos de uso, diagramas de actividad, diagramas de interacción y máquinas de estado.

[7] https://www.uml-diagrams.org/uml-25-diagrams.html

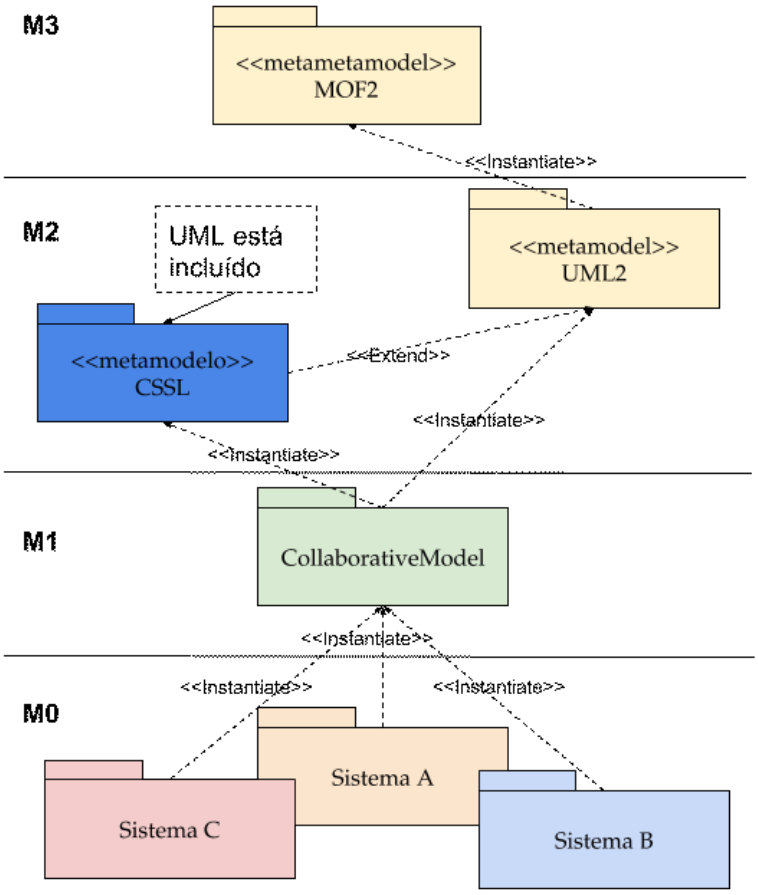


Figura 4.3: Extendiendo UML

En el caso del diagrama de casos de uso, se puede usar directamente, ya que un caso de uso puede tener varios actores. Los casos de uso se pueden asociar a posibles actividades colaborativas del sistema y a los actores con los roles que intervienen en el mismo.

4.3. Análisis del CSSL

Para realizar el análisis del lenguaje CSSL, se divide al metamodelo en cuatro sub-modelos (conjunto de metaclases vinculadas que cubren algún aspecto del sistema): el modelo conceptual, el modelo de operaciones, el modelo dinámico y el modelo de awareness. En los diagramas aparecen metaclases con tres colores distintos. En gris oscuro están las metaclases de UML, en gris claro las metaclases abstractas y en amarillo claro están las metaclases propias del metamodelo de sistemas colaborativos.

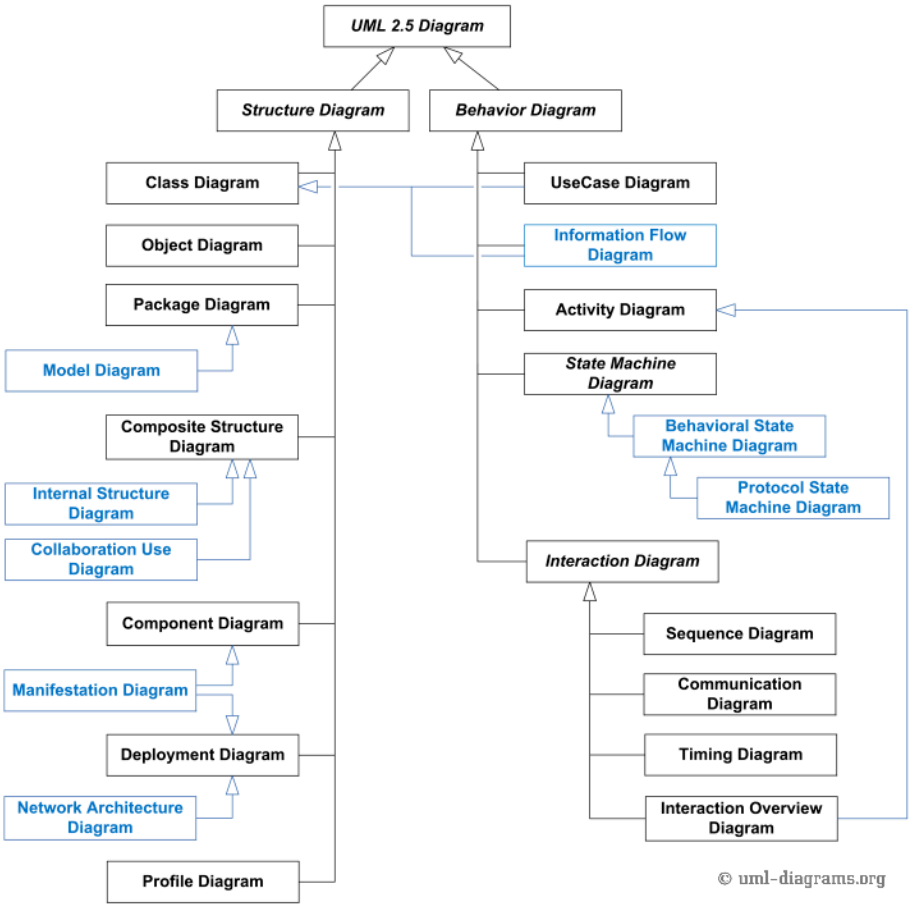


Figura 4.4: Diagramas UML [7]

4.3.1. Sub-Modelo Conceptual

El modelo conceptual cuenta con un conjunto de metaclases, que representan los conceptos estructurales de los sistemas colaborativos y que se mencionan en la mayoría de los trabajos analizados en la revisión [37]. Sus clases principales son *CollaborativeElement* y *CollaborativeAssociation* como se muestra en la figura 4.5.

Estas metaclases son subclase de Class y Association de Uml, entonces se pueden asociar los elementos colaborativos de acuerdo a las asociaciones creadas. Por ejemplo, se puede modelar que una CollaborativeActivity se desarrolla en algún Workspace o que utiliza alguna Tool. En las fichas que se describen a continuación, puede verse los detalles de cada metaclase.

Meta-Clase CollaborativeElement:UML::Class
Descripción: CollaborativeElement es una metaclase abstracta, que define las características generales de los elementos que aparecen en los sistemas colaborativos.

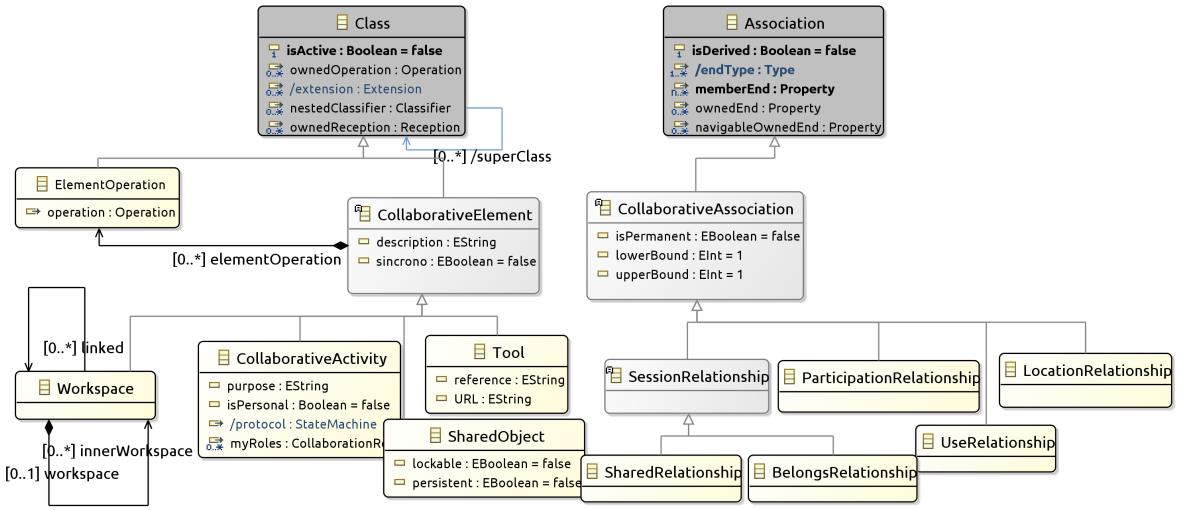


Figura 4.5: CSSL: Modelo Conceptual

Estos elementos pueden asociarse utilizando las subclases de CollaborativeAssociation.

SuperClass: Uml::Class

Sub-clases: Workspace, Tool, SharedObject, CollaborativeActivity

Atributos: -

- **description**: String[1..1] Permite describir el elemento colaborativo.
- **sincrono**: Boolean[1..1] Se utiliza para determinar si el elemento colaborativo se sincroniza o no.

Association Ends: -

- **elementOperation**: ElementOperation [0..*]. Esta colección permite asociar al elemento colaborativo con las operaciones que se le asignan a los roles que intervienen en el sistema.

Meta-Clase Workspace:CollaborativeElement

Descripción: El Workspace es una metaclase, que representa el lugar en el que la colaboración se lleva a cabo. Dentro de un workspace se realizan las actividades colaborativas y es donde se encuentran las herramientas colaborativas que son utilizadas

para comunicarse y trabajar con los objetos compartidos. Desde el punto de vista del diseñador, los espacios sirven para contextualizar las actividades que se desarrollan en ese espacio. Por ejemplo, si se quiere modelar un aula virtual como un espacio, se sabe que en el aula se realizan actividades de clase, o consulta y que los roles suelen ser docente, tutor y alumno. Los espacios de trabajo pueden estar compuestos por sub-espacios, que a su vez tendrán actividades colaborativas, herramientas y roles que permiten componer espacios de colaboración más complejos.

SuperClass: CollaborativeElement

Association Ends: -

- **innerWorkspace**: Workspace[0..*] Aquí se encuentran los espacios que se encuentran dentro de él.
- **workspace**: Workspace[0..1] Indica el espacio en el que está contenido.
- **linked**: Workspace[0..*] Aquí se encuentran los espacios con los que están vinculados y se utilizan para modelar la navegación entre los espacios.

Meta-Clase CollaborativeActivity:CollaborativeElement

Descripción: Esta metaclase representa las actividades que realizan los usuarios en el sistema. Con las actividades colaborativas se define qué roles interactúan, con cuáles herramientas y cuál es el protocolo de interacción. Estas actividades se relacionan con otros elementos colaborativos a través de las asociaciones, por ejemplo, se puede modelar las Tool que se pueden usar (useRelationship) en la actividad.

SuperClass: CollaborativeElement

Atributos: -

- **purpose**: String[1..1] Describe el propósito de la actividad colaborativa.
- **isPersonal**: Boolean[1..1]= false. Define si la actividad es personal o colaborativa.

Association Ends: -

- **/myRoles**: CollaborationRole[0..*]. Aquí se encuentran los roles que participan en la actividad colaborativa.
- **/protocol**: StateMachine [1..1]. Se modela el comportamiento de la actividad colaborativa. El protocolo es una propiedad derivada y representa el ownedBehavior que se hereda de UML::Class.

Meta-Clase Tool:CollaborativeElement

Descripción: Esta metaclase representa las herramientas colaborativas que se utilizan en las actividades colaborativas. Con ellas los usuarios editan los objetos compartidos. Las herramientas más comunes son el Chat o Pizarra Compartida, Editores colaborativos o repositorios.

SuperClass: CollaborativeElement

Atributos: -

- **reference**: String[0..1] Es una referencia al producto que se utiliza.
- **URL**: String[0..1] Ubicación de la descripción del producto.

Meta-Clase CollaborativeAssociation:UML::Association

Descripción: CollaborativeAssociation es una metaclase abstracta, que permite relacionar a dos o más elementos colaborativos (CollaborativeElement). La relación entre elementos colaborativos se realiza con alguna semántica particular. Utilizando alguna de las subclases se podrá por ejemplo, decir que una Tool se puede usar (useRelationship) en alguna CollaborativeActivity o que un SharedObject está en (locationRelationship) algún workspace.

SuperClass: UML::Association

Sub-clases: SessionRelationship, ParticipationRelationship, LocationRelationship, UseRelationship

Atributos: -

- **isPermanent**: Boolean[1..1] = false. Determina si la asociación es permanente o se puede cambiar. Por ejemplo, sirve para que algunos objetos puedan

cambiar de lugar. Estarán asociados temporalmente a un espacio y luego se moverán a otro. En otros casos, si tenemos que la asociación es permanente el objeto no podrá salir del espacio en el que está.

- **lowerBound**: Eint[0..1]= 1. Especifica la cardinalidad mínima que tendrá esa asociación.
- **upperBound**: Eint[0..1]= 1. Especifica la cardinalidad máxima que tendrá esa asociación.

Meta-Clase SessionRelationship:CollaborativeAssociation

Descripción: SessionRelationship es una metaclase abstracta, que permite relacionar a dos o más elementos colaborativos (CollaborativeElement). Esta relación, se utiliza para describir cómo se relacionan las actividades colaborativas con los espacios. Vinculan las instancias de Workspace con CollaborativeActivity.

SuperClass: CollaborativeAssociation

Sub-clases: BelongRelationship, SharedRelationship

Meta-Clase BelongRelationship:SessionRelationship

Descripción: BelongRelationship es una metaclase que indica que una CollaborativeActivity pertenece a un Workspace. Esto permite modelar por ejemplo, que una Clase (instancia de CollaborativeActivity) se desarrolla en un Aula (instancia de Workspace).

SuperClass: SessionRelationship

Meta-Clase SharedRelationship:SessionRelationship

Descripción: SharedRelationship es una metaclase que indica que una CollaborativeActivity se desarrolla en distintos Workspace. Esto permite modelar por ejemplo, que una conversación (instancia de CollaborativeActivity) se desarrolla con usuarios que pueden estar en distintos espacios que son instancias de Workspace.

SuperClass: SessionRelationship

Meta-Clase UseRelationship:CollaborativeAssociation

Descripción: UseRelationship es una metaclase que se utiliza para modelar qué instancias de Tool se utilizan en una instancia de CollaborativeActivity. Esto permite modelar, por ejemplo, que una Pizarra (instancia de Tool) se utiliza en el Aula (instancia de CollaborativeActivity).

SuperClass: CollaborativeAssociation

Meta-Clase LocationRelationship:CollaborativeAssociation

Descripción:

SuperClass: CollaborativeAssociation

Meta-Clase ParticipationRelationship:CollaborativeAssociation

Descripción:

SuperClass: CollaborativeAssociation

4.3.2. Modelo de Operaciones del Sistema

Este sub-modelo permite modelar la relación que hay entre los roles y las operaciones del sistema. Los roles tienen asignadas un conjunto de RoleElementOperation, que es una metaclase que modela una operación de un elemento colaborativo que se le asigna a un rol. De esta forma, podemos describir que operaciones de cada elemento colaborativo pueden utilizar cada uno de los roles en el sistema.

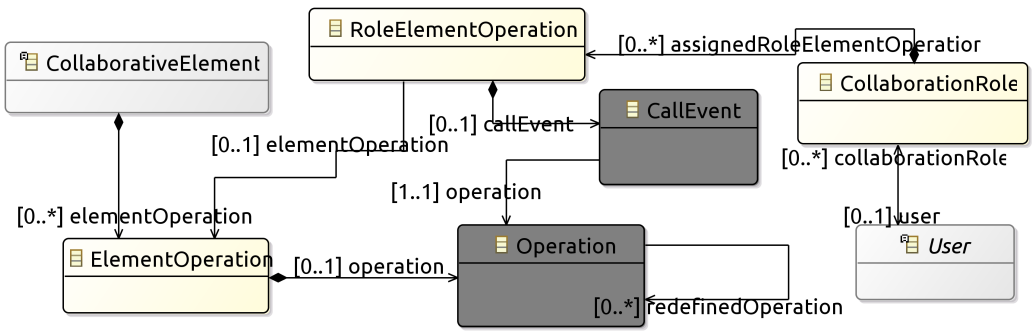


Figura 4.6: CSSL: Modelo de Operaciones

En la figura 4.6 se nuestran las metaclases de este sub-modelo y se detalla en las fichas cada una de ellas.

Meta-Clase CollaborationRole:UML::Class

Descripción:

SuperClass: Uml::Class

Association Ends: -

- **user**: User[1..1]
- **assignedRoleOperation**: RoleOperation[0..*]

Meta-Clase User:UML::Class

Descripción:

SuperClass: Uml::Class

Association Ends: -

- **collaborationRole**: CollaborationRole[1..*]

Meta-Clase ElementOperation:UML::Class

Descripción: Esta metaclase representa las operaciones que los elementos colaborativos tienen disponibles. Sirve para describir, por ejemplo, que los workspace tendrán operaciones para entrar o salir del espacio. Las herramientas por otro lado tendrán operaciones relacionadas con la colaboración por ejemplo: un repositorio tendrá operaciones para subir y/o editar un archivo. Más adelante, se verá cómo se asignan esas operaciones a los distintos roles y qué efecto tendrá que se ejecute alguna de ellas.

SuperClass: Uml::Class

Association Ends: -

- **operation**: Operation[1..1] Aquí se encuentran la operación concreta que se ejecutará en el sistema.

Meta-Clase RoleElementOperation:UML::Class

Descripción:

SuperClass: Uml::Class

Association Ends: -

- **elementOperation**: ElementOperation[0..1]
- **callEvent**: Uml::CallEvent[0..1]

4.3.3. Modelo Dinámico

En esta sección, se discute el submodelo que permite trabajar con aspectos dinámicos de los sistemas colaborativos. En la revisión [37], se definió que los sistemas colaborativos tienen aspectos dinámicos que tienen que ser modelados para describir el comportamiento de los sistemas colaborativos con awareness. Con estas herramientas de diseño, se podrá especificar el orden en que las actividades colaborativas se llevan a cabo, y cómo las operaciones que realizan los usuarios afectan la dinámica de la colaboración. En el metamodelo CSSL hay dos conceptos que permiten modelar estos aspectos dinámicos. Por un lado, están los procesos colaborativos, y por otro, los protocolos.

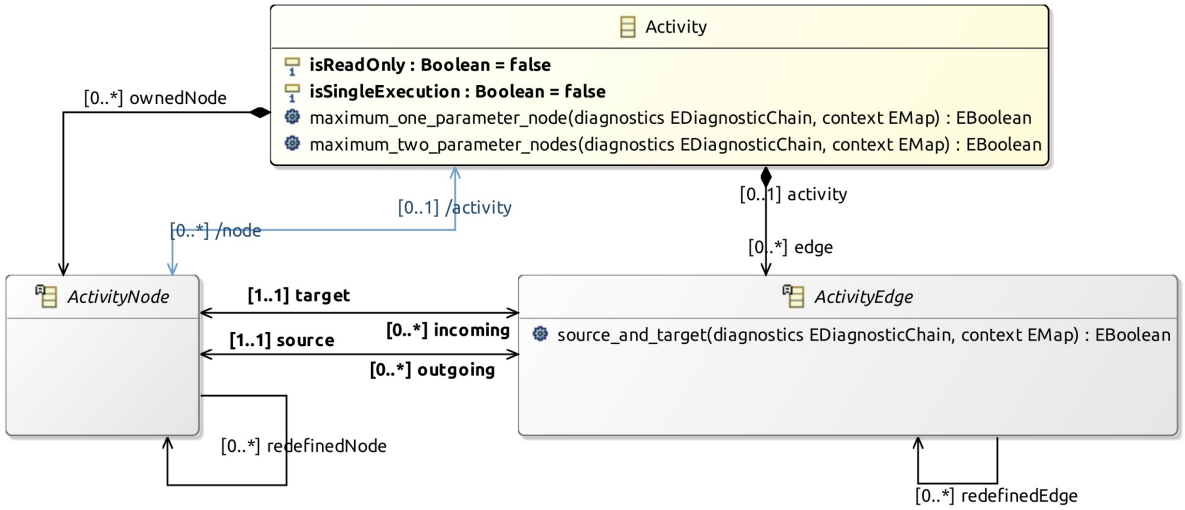


Figura 4.7: Procesos en UML

4.3.3.1. Procesos Colaborativos

Los procesos colaborativos se entienden como procesos "tradicionales", donde los nodos pueden ser actividades colaborativas. Se sabe que en UML los procesos se modelan usando la clase Activity, que contiene un conjunto de Nodos (ActivityNode) y Ejes (ActivyEdge) como se muestra en la figura 4.7.

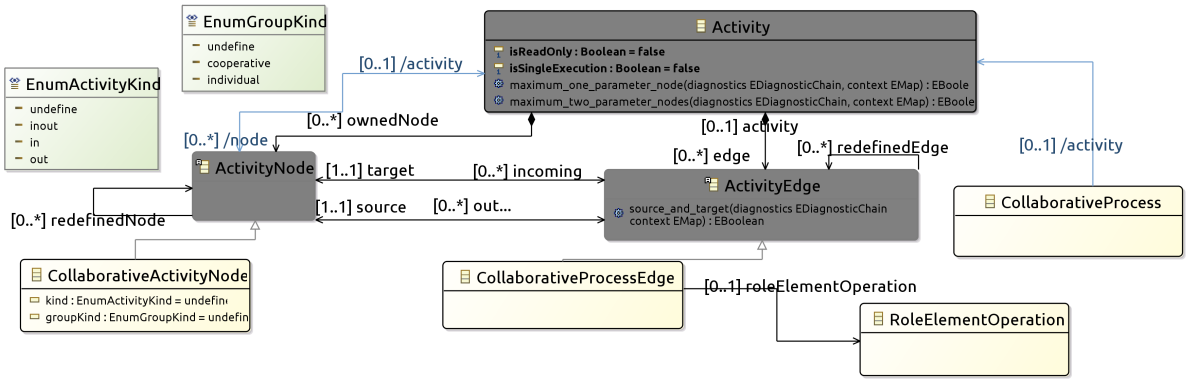


Figura 4.8: Procesos Colaborativos en CSSL

Teniendo en cuenta que se pretende modelar procesos donde los nodos sean Actividades Colaborativas, es que se extiende UML creando una sub-clase de ActivityNode que se llama CollaborativeActivityNode. También se extiende la clase ActivityEdge con la metaclase CollaborativeProcessEdge para tener ejes que se activen, cuando algún rol ejecuta alguna operación del sistema como se muestra en la figura 4.8. Con este sub-modelo, se puede instanciar procesos donde las actividades colaborativas sean los nodos, y algunas operaciones que ejecutan los roles sean los ejes.

Como en los diagramas de actividad de UML, se cuenta con otros nodos (ControlNodes) que se usan para manejar el flujo de los tokens entre los nodos, y representan un conjunto de nodos concretos como: InitialNode, FinalNode, ForkNode, MergeNode, etc. El InitialNode es el primero en estar disponible al iniciar la actividad. El ForkNode recibe un token de un ActivityEdge de entrada y envía un token a cada ActivityEdge de salida. En la figura 4.9, se muestran las metaclases que se utilizan para modelar los procesos colaborativos y se incluyen los nodos de control (NontrolNode) de UML.

Las siguientes fichas detallan las clases que componen este sub-modelo.

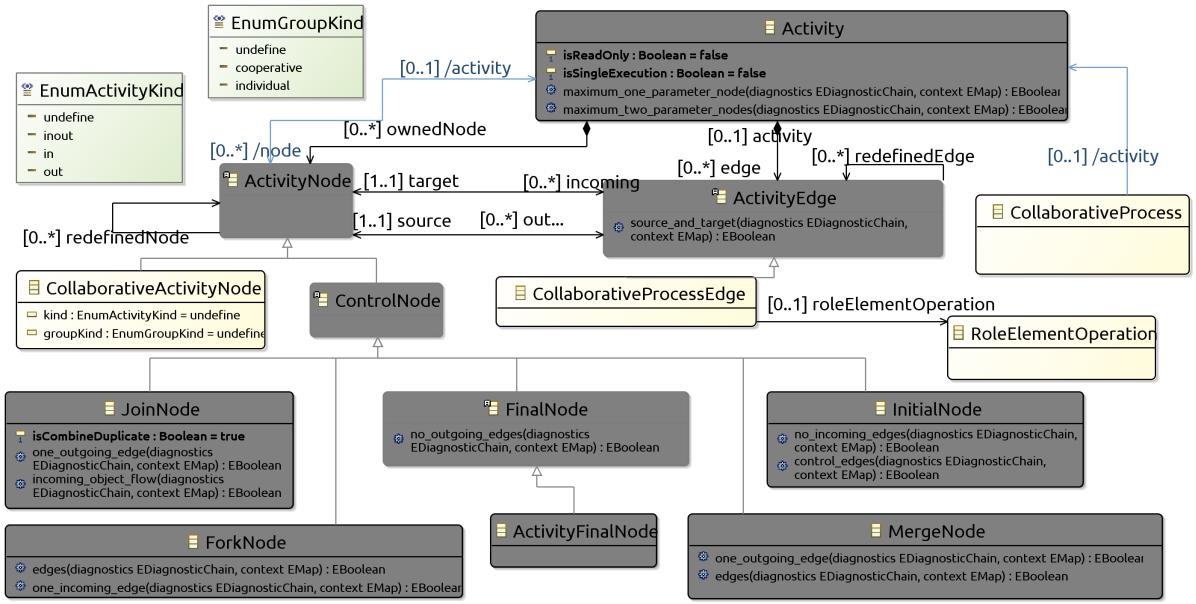


Figura 4.9: Procesos Colaborativos con ControlNodes

Enum EnumGroupKind:Enumeration

Descripción: Es una enumeración usada para especificar las distintas conformaciones de grupos.

Literales: -

- **undefine**: No se determina si la actividad produce o recibe información.
- **cooperative**: Especifica que la actividad es cooperativa.
- **individual**: Especifica que la actividad es individual.

Enum EnumActivityKind:Enumeration

Descripción: Es una enumeración usada para especificar características de las actividades colaborativas, en relación a lo que producen o reciben las tareas.

Literales: -

- **undefine**: No se determina si la actividad produce o recibe información.
- **in**: Especifica que la actividad recibe alguna información.
- **out**: Especifica que la actividad produce alguna información.
- **inout**: Especifica que la actividad recibe y produce información.

Meta-Clase CollaborativeActivityNode:UML::ActivityNode

Descripción: Representa a una actividad colaborativa dentro de un proceso colaborativo. Cuando el diseñador define que una actividad colaborativa, se crea una instancia de "CollaborativeActivityNode". Así entonces los procesos colaborativos se podrán construir con nodos colaborativos y nodos de control, que permiten diagramar los procesos.

SuperClass: UML::ActivityNode

Atributos: -

- **kind**: EnumActivityKind [1..1] = undefine. Determina si la actividad produce algún resultado, recibe información o ambos. Ver el enumerativo EnumActivityKind.
- **groupKind**: EnunGroupKind [1..1] = undefine. Especifica cómo es la composición del grupo. Ver el enumerativo EnunGroupKind.

Meta-Clase CollaborativeActivityNode:UML::ActivityNode

Descripción: Representa a una actividad colaborativa dentro de un proceso colaborativo. Cuando el diseñador define que una actividad colaborativa, se crea una instancia de "CollaborativeActivityNode". Así entonces los procesos colaborativos se podrán construir con nodos colaborativos y nodos de control, que permiten diagramar los procesos.

SuperClass: UML::ActivityNode

Atributos: -

- **kind**: EnumActivityKind [1..1] = undefine. Determina si la actividad produce algún resultado, recibe información o ambos. Ver el enumerativo EnumActivityKind.
- **groupKind**: EnunGroupKind [1..1] = undefine. Especifica cómo es la composición del grupo. Ver el enumerativo EnunGroupKind.

Meta-Clase CollaborativeProcessEdge:UML::ActivityEdge

Descripción: Representa a una transición entre nodos de un proceso colaborativo. La diferencia con los "ActivityEdge" es que la transición se activa cuando un rol ejecuta alguna operación con algún elemento colaborativo. Por ejemplo: cuando un rol cierra un documento, puede provocar que se termine una actividad colaborativa y pasar a otra actividad.

SuperClass: UML::ActivityEdge

Hasta aquí se han descripto a los procesos colaborativos que definen como se organizan las actividades colaborativas dentro del sistema. En qué orden se realiza cada una de ellas y qué operaciones activan los cambios de actividad. En la siguiente sección se trata sobre el modelado de la dinámica dentro de una actividad.

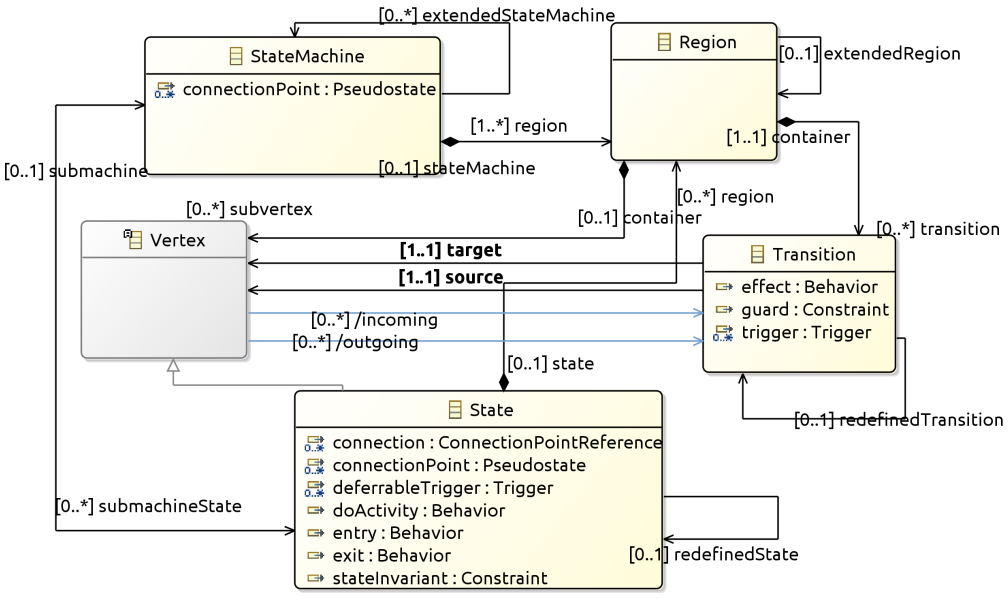


Figura 4.10: Maquinas de Estado - UML State Machine

4.3.3.2. Protocolos de Colaboración

Los protocolos en términos generales son un conjunto de reglas que define de qué manera debe realizarse una actividad. En el caso de los sistemas colaborativos, los protocolos establecen las reglas que determinan cuándo y cómo pueden participar los usuarios. Las máquinas de estado

se utilizan para modelar los protocolos. En UML existen un conjunto de clases que permiten instanciar máquinas de estado y se muestran en la siguiente figura 4.10, donde se destacan las clases que modelan los estados “State” y las transiciones “Transition”.

En el metamodelo CSSL, se define al comportamiento de una actividad colaborativa, como una máquina de estado donde los estados representan las operaciones que pueden ejecutar los distintos roles que intervienen en la actividad. Por otro lado, se define que las transiciones, pueden ser disparadas cuando se ejecuta alguna operación que realiza alguno de los roles, con los elementos colaborativos.

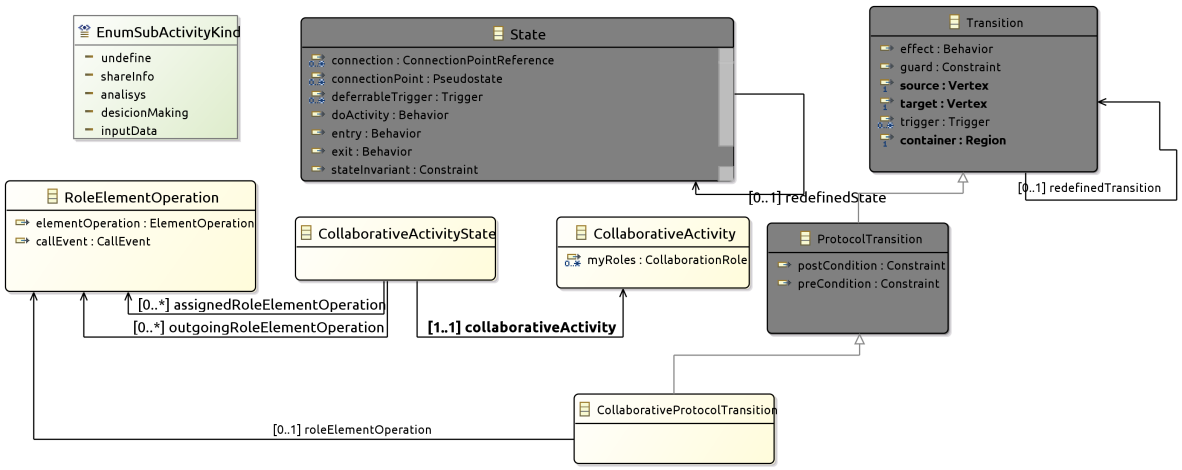


Figura 4.11: Protocolos en CSSL

Para modelar esto, CSSL extiende las máquinas de estado de UML, definiendo una subclase de “State” que represente un estado de una actividad Colaborativa particular usando la metaclase “CollaborativeActivityState”. Por otro lado, se cuenta con una sub-clase de transition “CollaborativeProtocolTransition” que permite identificar la operación que ejecutó algún rol para activar la transición, como se muestra en la figura 4.11.

Las siguientes fichas describen las metaclases de este sub-modelo.

Meta-Clase CollaborativeActivityState:UML::State

Descripción: Representa a los estados de una actividad colaborativa. El principal objetivo de esta metaclase, es el de mantener las operaciones de los elementos colaborativos que los roles pueden ejecutar y las operaciones que cambian de estado.

SuperClass: UML::State

Association Ends: -

- **assignedRoleElementOperation**: RoleElementOperation [0..*]. Aquí se encuentran las operaciones de los elementos colaborativos, que pueden ejecutar los roles que participan en la actividad colaborativa.
- **outgoingRoleElementOperation**: RoleElementOperation [0..*]. Aquí se encuentran las operaciones de los elementos colaborativos, que pueden ejecutar los roles que cambian de estado en la máquina de estado.

Meta-Clase CollaborativeProtocolTransition:UML::ProtocolTransition

Descripción: Representa a las transiciones de una máquina de estado de una actividad colaborativa. El principal objetivo de esta metaclase, es el de modelar las transiciones entre estados de una actividad colaborativa. Estas transiciones se disparan a partir de las operaciones de los elementos colaborativos que ejecutan los roles de la actividad colaborativa.

SuperClass: UML::ProtocolTransition

Association Ends: -

- **roleElementOperation**: RoleElementOperation [0..1]. Aquí se encuentran las operación del elemento colaborativo que al ser ejecutado por un rol se activa la transición.

Enum EnumSubActivityKind:Enumeration

Descripción: Esta enumeración sirve para especificar distintos tipos de los estados de las actividades colaborativas. Se pueden definir estados donde se comparta información, se realiza un análisis o se toma una decisión.

Literales: -

- **undefine**:No se determina qué tipo de sub-actividad se realiza.
- **shareInfo**: Especifica que la sub-actividad comparte información.
- **analisys**: Especifica que la sub-actividad analiza alguna información.
- **desicionMaking**: Especifica que la sub-actividad debe tomar alguna decisión.

- **inputData**: Especifica que la sub-actividad introduce algún dato.

El lenguaje CSSL cuenta con recursos para modelar la dinámica de las aplicaciones colaborativas. Se puede describir procesos que involucran distintas actividades colaborativas o modelar la intaracción de los usuarios dentro de una actividad usando los protocolos.

4.3.4. Modelo de Awareness

4.3.4.1. Awareness del Modelo Conceptual

A partir de la revisión sistemática [37], se destacan siete requisitos que deben cumplir los modelos que representan los sistemas colaborativos. Tres de ellos hacen referencia al modelado de la funcionalidad de awareness.

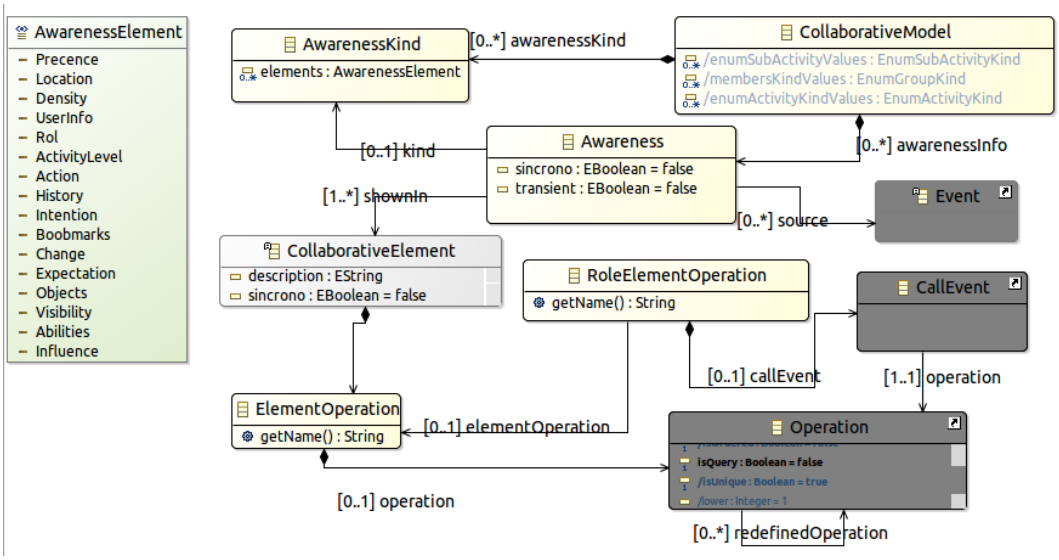


Figura 4.12: Modelo de Awareness en CSSL

Para cumplir con los requisitos expresados en la sección 3.2.2, el metamodelo CSSL, cuenta con metaclases que permiten asociar el awareness a distintos elementos colaborativos del sistema que se muestran en la figura 4.12.

Este sub-modelo responde al siguiente requisito:

***Requisito 2**: Contar con un modelo que permita expresar el awareness vinculado a los conceptos que intervienen en el sistema. El modelo tiene que cubrir las distintas alternativas*

que presenta el awareness (workspace awareness, social awareness, group awareness, entre otras).

Para expresar las distintas alternativas de awareness, se cuenta con la metaclase AwarenessKind que se usa para instanciar los tipos de awareness que tendrán los modelos colaborativos que están creando. Luego, se podrán crear los awareness concretos que se utilizaran en el sistema - que forman la colección awarenessInfo - y asignarle el tipo de awareness. Finalmente, estos awareness concretos están vinculados a los conceptos que intervienen en el sistema con la relación shownIn y source. Con shownIn[1..*], se expresa donde será mostrado el awareness concreto, donde pueden ser varios elementos colaborativos y con source[0..*] se señala, cuáles eventos van a actualizar el awareness concreto.

En las fichas que siguen, se detallan las clases del metamodelo CSSL, que permiten modelar Awareness.

Enum AwarenessElement:UML::Enumeration

Descripción: Este enumerativo tiene los elementos de awareness básicos que se describieron inicialmente en el trabajo de Gutwin y Greenberg [7, 8] que se listaron en la tabla 2.1 y posteriormente estudiados en otros trabajos, entre ellos el de Dourish y Bellotti [6].

Literales: -

- Precence, Location, Density, UserInfo, Rol, ActivityLevel, Action, History, Intention, Boobmarks, Change, Expectation, Objects, Visibility, Abilities, Influence.
- Estos elementos básicos se componen para obtener información de Awareness más compleja; por ejemplo: si se quiere mostrar la ubicación de los usuarios se combina el elemento Location+UserInfo. Si se quiere mostrar que han hecho los roles y en qué lugar del sistema se combinan Action+History+Location+Rol.

Meta-Clase AwarenessKind:UML::Class

Descripción: Representa a los distintos tipos de awareness que soporta el sistema. Los tipos de awareness se construyen combinando los elementos de awareness que lo componen, por ejemplo, ubicación, presencia, acción, identidad, etc. El modelo del

sistema colaborativo “CollaborativeModel”, mantiene un conjunto de tipos de awareness que se usará en el sistema. Luego, cuando se instancian los awareness concretos que se utilizarán en el sistema, se define de qué tipo serán.

SuperClass: UML::Class

Association Ends: -

- **elements**: AwarenessElement [0..*]. Aquí se componen los elementos de Awareness que se combinan en este tipo de awareness.

Meta-Clase Awareness:UML::Class

Descripción: Con esta metaclase se modela los awareness concretos que se utilizarán en el sistema colaborativo. Esta metaclase, será de alguno de los tipos de awareness definidos para el modelo que se está describiendo, se mostrará en uno o varios elementos colaborativos y tendrá un evento que originará la presentación del mismo.

SuperClass: UML::Class

Association Ends: -

- **kind**: AwarenessKind [0..1]. Representa alguno de los tipos de Awareness definidos en el sistema.
- **shownIn**: CollaboratieElement [0..*]. Aquí se encuentran los elementos colaborativos donde se mostrará la información de Awareness.
- **source**: Event [0..1]. Hace referencia al evento que origina la actualización de la información de awareness.

Con estas metaclases, se puede modelar la información de awareness que se muestra en algún espacio/actividad/herramienta. Por ejemplo, el awareness de presencia que se actualiza cuando ingresa algún usuario a algún elemento del sistema.

4.3.4.2. Awareness de los Procesos Colaborativos

Relacionado al sub-modelo, de la figura 4.8, que permite trabajar con procesos colaborativos, en la figura 4.13, se muestra cómo se vincula el awareness con las metaclases del lenguaje CSSL que permiten modelar los procesos colaborativos.

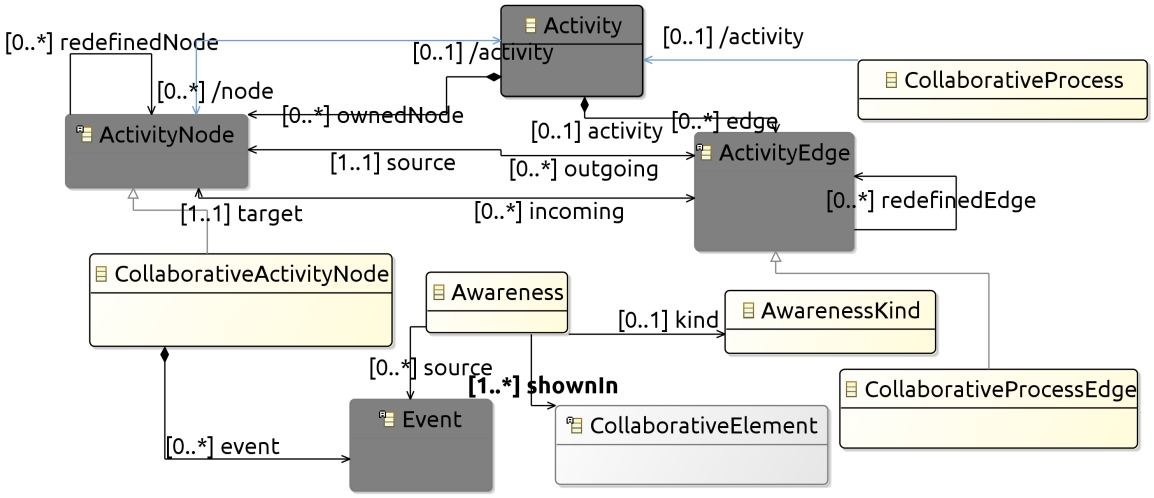


Figura 4.13: Modelo de Awareness de los procesos colaborativos

En la figura 4.13, puede verse que los nodos de las Actividades Colaborativas (CollaborativeActivityNode) conocen a un conjunto de eventos que pueden originar ciertos awareness. En UML la clase Event es una clase abstracta cuyas subclases son "Time-Event", "Change-Event", "Call-Event" entre otras. De esta forma, se puede modelar que ante algunos eventos se active algún awareness y que se mostrará en algún elemento colaborativo. Por ejemplo, se puede definir que cuando se inicia o finaliza una actividad, se active un evento que dispare la actualización de algún awareness. También, pueden ser eventos relacionados con las operaciones que ejecutan los roles en alguna actividad colaborativa.

Con el modelo de awareness asociado a los procesos colaborativos, se puede agregar distintos eventos a los nodos de los procesos colaborativos. De esta forma, se está cumpliendo con el requisito que permite asociar el awareness a los procesos colaborativos.

***Requisito 4**: Permitir asociar información de awareness a los procesos colaborativos.*

4.3.4.3. Awareness de los Protocolos

En el caso de los protocolos, puede verse en el sub-modelo de clases de la figura 4.14, cómo se relaciona el awareness con los estados de los protocolos. La clase CollaborativeActivityState es sub-clase de UML::State y forma parte de un ProtocolStateMachine, que es la forma en que se especifica el comportamiento de las actividades colaborativas, como se muestra en la figura 4.14.

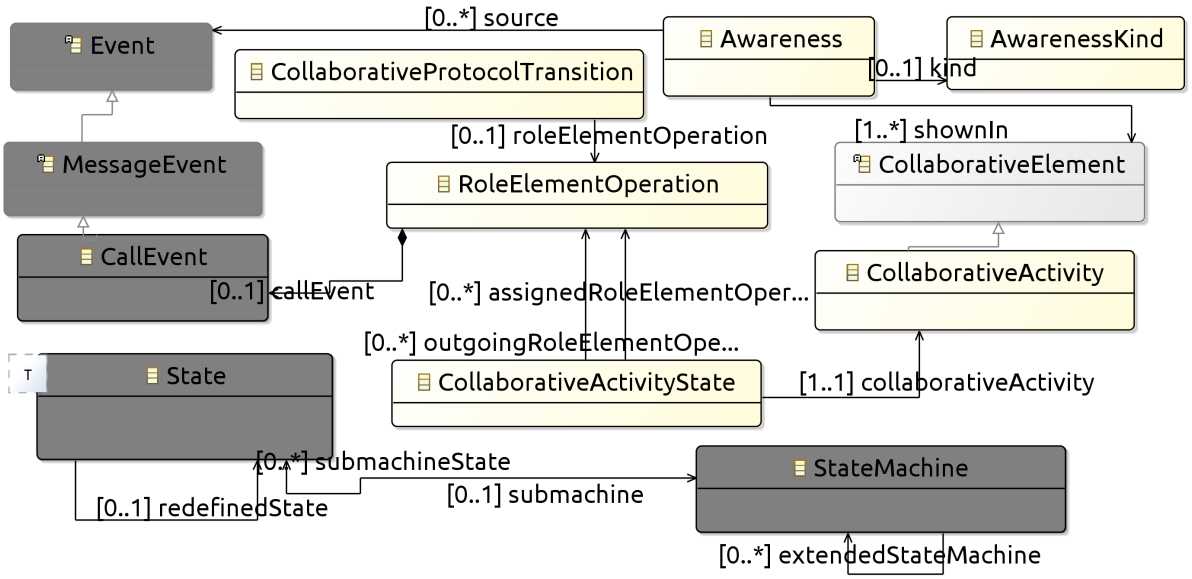


Figura 4.14: Modelo de Awareness de los protocolos

Por otro lado, la clase CollaborativeActivityState conoce a un conjunto de operaciones asignadas a los roles a través de RoleElementOperation en la colección assignedRoleElementOperation. Alguna de esas operaciones, permiten cambiar de estado y se agrupan en el contenedor outgoingRoleElementOperation, usando instancias de RoleElementOperation. El cambio de estado está modelado con la clase CollaborativeProtocolTransition que tiene el estado origen y destino, y la operación que acciona el rol con algún elemento colaborativo.

Se ve en la figura 4.14, cómo se vincula la información de awareness con los distintos estados de las actividades colaborativas. Los estados, se entienden como un conjunto de operaciones que los roles pueden realizar. A su vez, las operaciones a partir de la clase “RoleElementOperation”, tienen un evento asociado. Al mismo tiempo, los awareness controlan un conjunto de Eventos con la agrupación source. De esta forma se puede modelar que se active alguna información

de awareness cuando una operación se ejecute en algún estado de una actividad colaborativa, cumpliendo de esta forma el requisito número 6.

***Requisito 6**: Incorporar distintos tipos de awareness, asociados a los estados por lo que pasa una actividad colaborativa.*

4.4. Conclusiones del capítulo

En el capítulo se presentó un lenguaje especifico de dominio (CSSL) construido como una extensión de UML, utilizando la técnica de metamodelado. CSSL permite definir en forma precisa, concisa y amigable los conceptos abstractos de los sistemas colaborativos, incluyendo el concepto de awareness, procesos colaborativos y protocolos.

Se analizó el mecanismo de inclusión de UML al lenguaje CSSL. Esto permite que, en los modelos construidos con éste último, se utilicen metaclases, diagramas de UML y todas las herramientas desarrolladas para UML - Editores de UML (tanto de gráficos o texto), Mapeadores de UML a código, etc.-

El lenguaje CSSL se detalló explicando los sub-modelos que describen distintos aspectos de los sistemas colaborativos, como el modelo conceptual, el modelo de operaciones, el modelo dinámico y el modelo de awareness. Para cada uno de ellos, se detalló cada una de las metaclases y sus propiedades.

Capítulo 5

CSSL Tool

El lenguaje CSSL, brinda una sintaxis abstracta de los sistemas colaborativos. Como fue construido a partir de UML, se puede utilizar cualquier editor de UML y crear instancias de las metaclases del lenguaje.

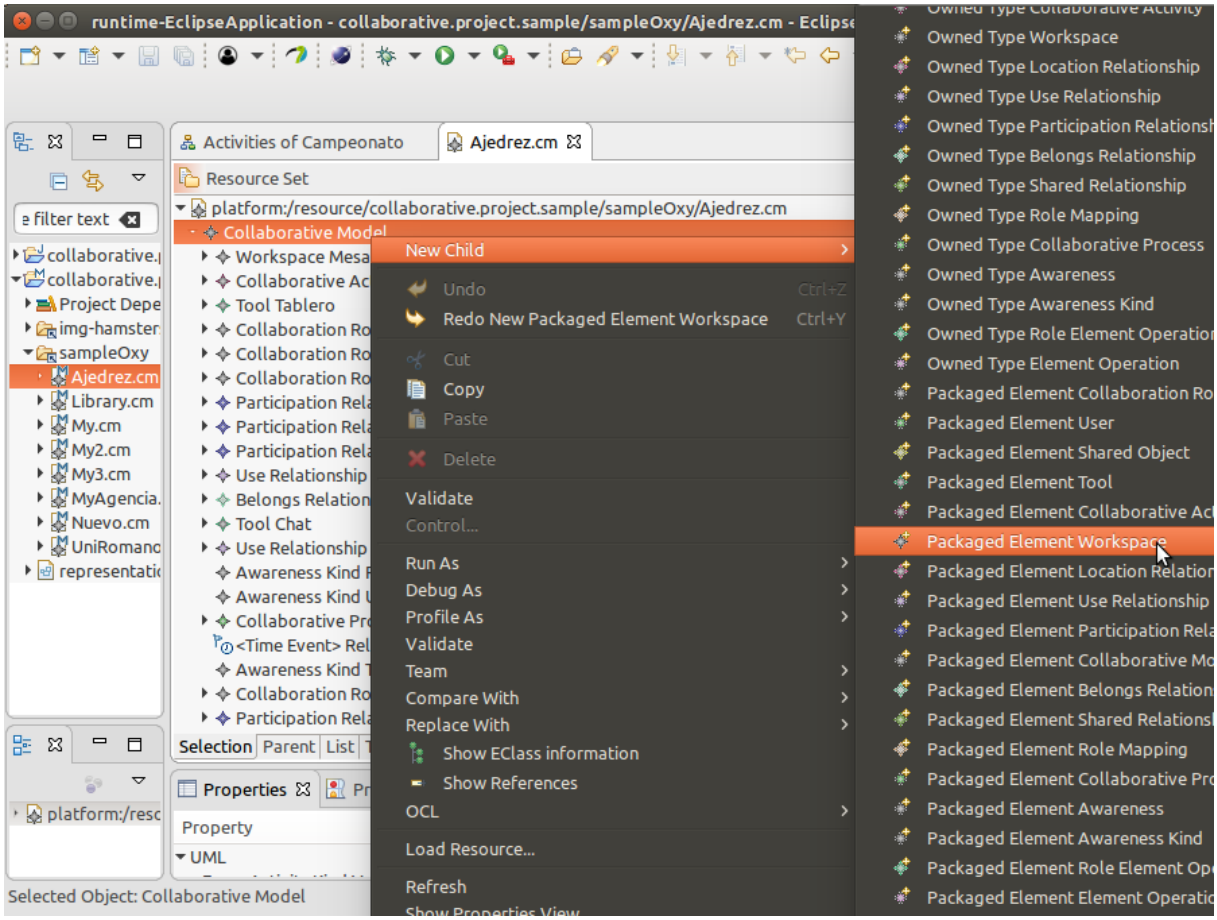


Figura 5.1: Editor de UML

En la figura 5.1 puede verse cómo se instancia un modelo colaborativo y usando el editor estándar de UML.

Primero, hay que crear una instancia de CollaborativeModel y usando un editor standard de UML, se crean sus elementos como hijos. Luego hay que editar las clases creadas para asignarles nombre y asociarlas entre sí. En la figura 5.1 se muestra un ejemplo creando una instancia de CollaborativeActivity y agrupándola a la colección PackageElement del CollaborativeModel.

Hay que tener en cuenta que crear modelos usando este editor, resulta un trabajo arduo y poco amigable. Usando esta forma de instanciar el metamodelo, se crea un listado en forma de árbol con todas las clases y asociaciones que se definieron en el metamodelo.

Para brindar mayor flexibilidad y legibilidad al DSL, se pueden crear distintas sintaxis concretas y representaciones gráficas, que suelen ser más expresivas para los diseñadores.

Sirius[8] es un proyecto de Eclipse que permite construir representaciones gráficas para arquitecturas basadas en modelos. Basado en el mecanismo de Viewpoint, Sirius permite a los diseñadores trabajar con representaciones de los elementos del dominio que se adaptan a las necesidades del equipo de trabajo. Para el lenguaje CSSL, se desarrolló un conjunto de representaciones (views) para describir distintos aspectos del sistema.

5.1. Proyecto de Especificación de Vistas en Sirius

El proyecto *Sirius* permite especificar vistas asociadas a una metaclase de un metamodelo. El conjunto de vistas se agrupa en un proyecto

a partir de clases de un Metamodelo y se lo llama: “Viewpoint Specification Project”. Este proyecto contiene la definición de un conjunto de herramientas de modelado que permiten ver y crear instancias de un metamodelo.

La estructura de la especificación se inicia de esta forma y tiene las siguientes propiedades como se muestra en la figura 5.2.

[8] https://www.eclipse.org/sirius/doc/

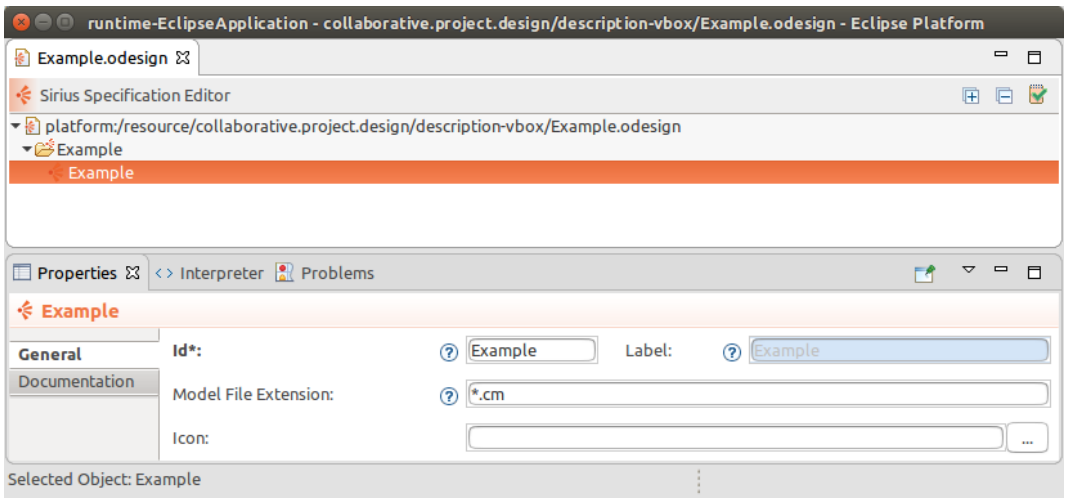


Figura 5.2: Propiedades del Viewpoint

- **id**: Es un identificador único, que luego es usado para trabajar con los modelos. En el ejemplo se llama "Example"
- **label**: Texto que aparece en el sistema cuando se muestra este objeto. Por default es igual que el id.
- **Model File Extension**: Permite asociar este viewpoint con los modelos que tengan esta extensión. En este caso *.cm. "Sirius" permite asociar un viewpoint a varias extensiones que se indican separadas por coma (por ejemplo, para asociarlo a UML y ecore hay que indicar UML, ecore)

5.1.1. Representaciones del Modelo

A partir de acá, se pueden definir distintos diagramas para los elementos del metamodelo. Para agregar un diagrama al viewpoint, se crea una nueva representación como se muestra en la Figura 5.3.

Para cada viewpoint, se pueden agregar distintas representaciones/diagramas. Los diagramas tienen una clase raíz "Domain Class", desde la cual se representan los elementos relacionados con ella. La forma de representación puede ser diferente, por ejemplo:

- **Diagram Description**: Tiene una representación gráfica donde aparecen nodos y relaciones y decoraciones.

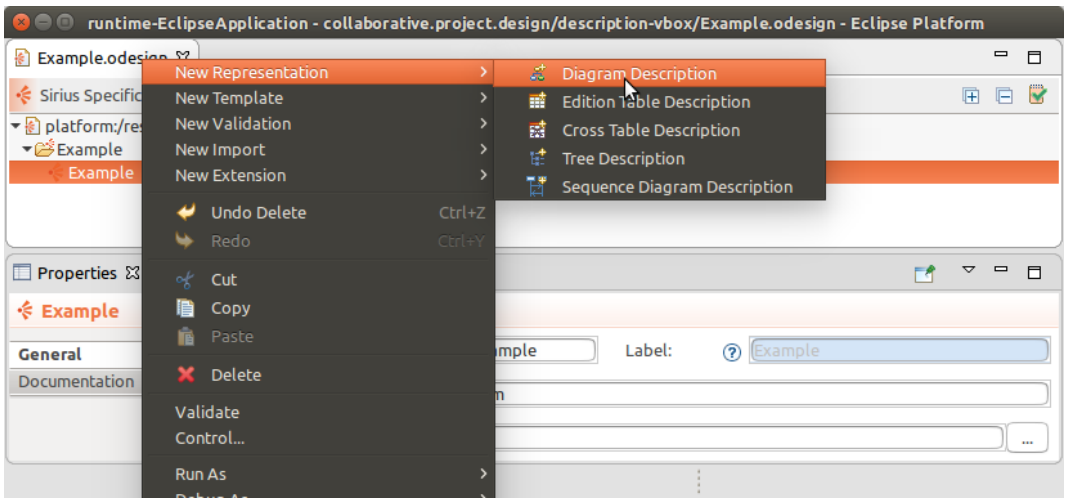


Figura 5.3: Creación de un Diagrama para un Modelo

- **Edition Table Description**: Representación en forma de tabla que permite editar los campos de la tabla.
- **Cross Table Description**: Otra representación de tabla, preparada para mostrar relaciones entre elementos en un formato de matriz.
- **Tree Description**: Permite mostrar los elementos en formato de árbol.
- **Sequence Diagram Description**: Como su nombre lo indica, esta representación permite mostrar una secuencia ordenada de elementos. Típicamente, representa eventos enviados y recibidos por los elementos a través del tiempo.

En la Figura 5.4, se muestra la creación de una representación de tipo Diagrama, que toma como origen la clase cm::CollaborativeModel. Esto significa que en el diagrama pueden aparecer todas las clases incluidas en un CollaborativeModel.

Las propiedades del Diagrama son:

- **id**: Identificador del elemento.
- **Label**: Nombre que aparece en el editor.
- **Domain Class**: El tipo del elemento desde el cual se podrán crear elementos en el diagrama.

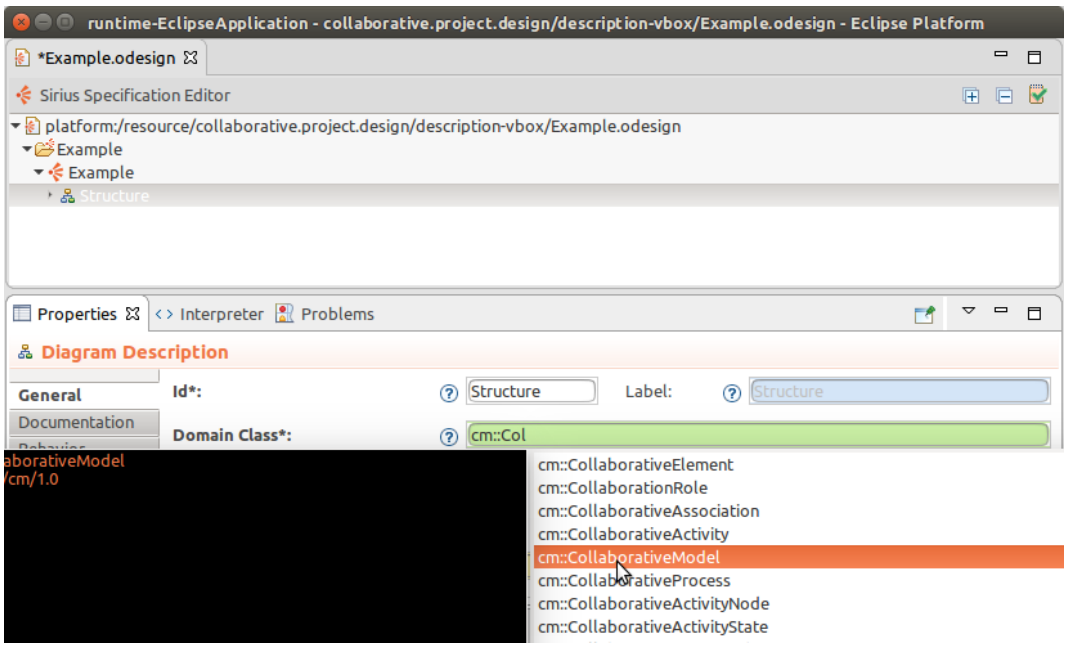


Figura 5.4: Propiedades de un Diagrama

- **Initialization**: Si es verdadero, la representación se crea automáticamente cuando se ingresa al sistema

- **Enable Popup Bars**: Si es verdadero, la herramienta de creación será visible en la paleta y disponible en la barra de popup del diagrama.

- **Precondition Expression**: Se espera un boolean. Es una expresión (código breve en aql, ocl, etc.) que determina si el diagrama puede ser abierto. Puede ser que alguna propiedad de la instancia con la que se está trabajando indique que el diagrama no puede ser creado porque está seteado en "false".

- **Show on Startup**: Si es verdadero, la representación de este tipo será abierta automáticamente cuando la sesión es abierta.

5.1.2. Nodos en un Diagrama

En los diagramas creados con Sirius se pueden agregar distintos componentes que representan alguno de los elementos del modelo. En la figura 5.5, se muestra cómo se agrega un nodo al diagrama, que son usados para representar gráficamente los elementos modelo. Los posibles elementos que se pueden agregar son:

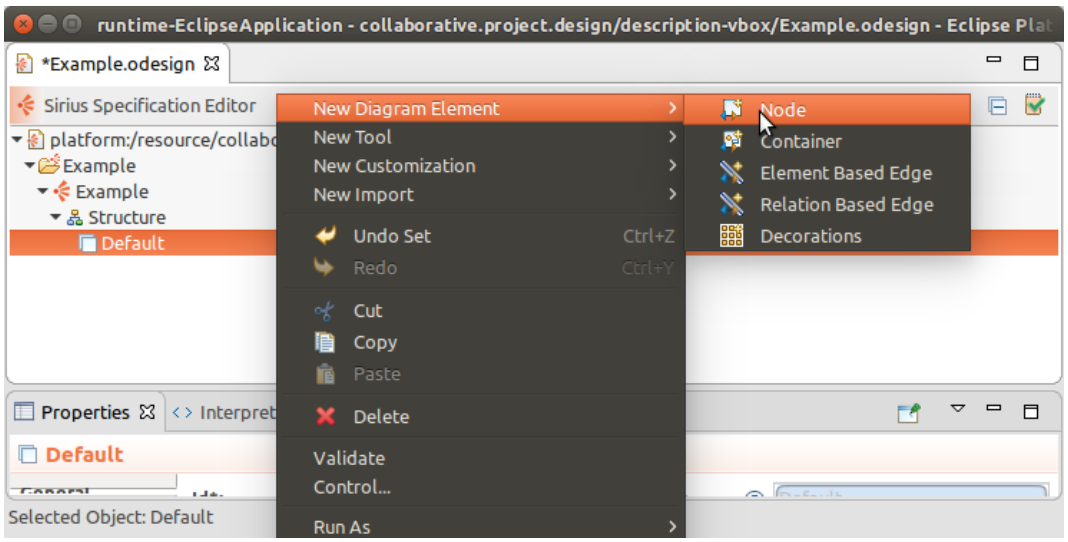


Figura 5.5: Nuevo nodo en el Diagrama

- **Nodes**: Los nodos son representaciones atómicas de una clase del modelo, ya que no pueden contener otro elemento. Sin embargo, en los bordes pueden contener elementos (Bordered Nodes) relacionados con el nodo. Los nodos pueden tener distintas representaciones y otras propiedades que lo describen. Pueden aparecer directamente dentro un diagrama o dentro de un elemento contenedor (container).

- **Containers**: Son usados para representar una clase del modelo, que pueden contener otros elementos (inclusive otros containers, recursivamente) y también pueden tener Bordered Nodes. Los containers, pueden aparecer en los diagramas o dentro de algún contenedor (igual que los nodos).

- **Element-Based Edge**: Los ejes son usados para definir conexiones entre los elementos del diagrama. Estos ejes son configurados como los nodos o containers basados en una clase del dominio. Además, hay que especificar las propiedades “Source Mapping” y “Target Mapping”, que se obtienen a partir de las expresiones “Source Finder Expression” y “Target Finder Expression”.

- **Relation-Based Edge**: Los ejes son usados para definir conexiones entre los elementos del diagrama. En este caso, se especifica los elementos origen y destino de la relación, usando las propiedades “Source Mapping” y “Target Mapping”.

- **Decorations**: Son usados para agregar anotaciones gráficas a los elementos del diagrama.

En la figura 5.6, se muestra cómo se configuran las siguientes propiedades de los nodos:

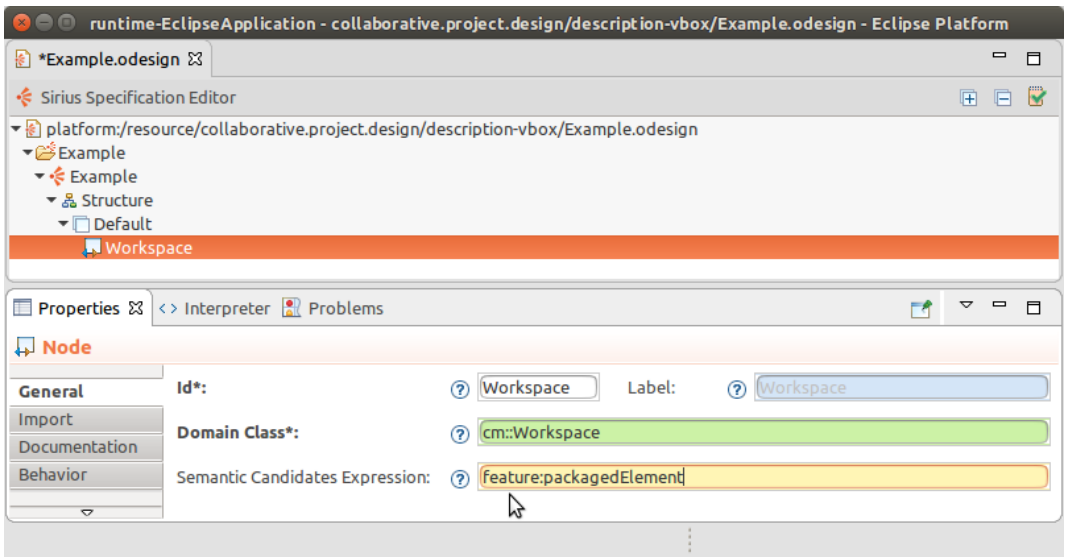


Figura 5.6: Propiedades de los Nodos

- **id**: Identificador único.
- **Label**: Texto usado para mostrar el nodo al usuario final.
- **Domain Class**: El tipo del elemento del modelo a mostrar. En el ejemplo de la imagen 5.6, se indica que diagrama mostrará las instancias de "cm::Workspace".
- **Semantic Candidates Expression**: Restringe la lista de los elementos a mostrar. En el ejemplo de la figura 5.6, se muestran los "Workspace" que están en la colección PackagedElement, en este caso, de CollaborativeModel que es la clase donde se incorporan estos workspaces. Si esta expresión queda vacía, "Sirius" usa el método eAllContents() para recuperar los elementos a mostrar, pero esto puede provocar problemas de performance.

Los nodos tienen un aspecto que se define por lo que en Sirius se llama *Style*. En la figura 5.7, se muestra cómo se le agrega un Style a un nodo. Hay varias posibilidades para definir los estilos, lo que brinda muchas posibilidades para representar los elementos y son las siguientes:

- **Basic Shape**: Es un formato simple que es un rectángulo.
- **Workspace Image**: Esta opción permite vincular una imagen a un elemento del diagrama. En los lenguajes específicos de dominio se suelen usar representaciones gráficas para los elementos del dominio.
- **Otras formas**: (Square, Diamond, Dot, Elipse, Note, Gauge, Custom Style)

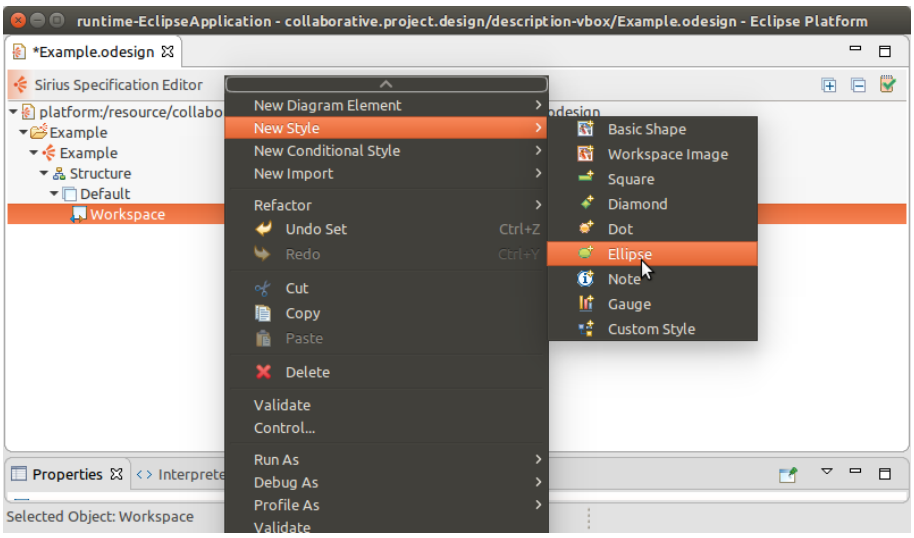


Figura 5.7: Estilo de un Nodo

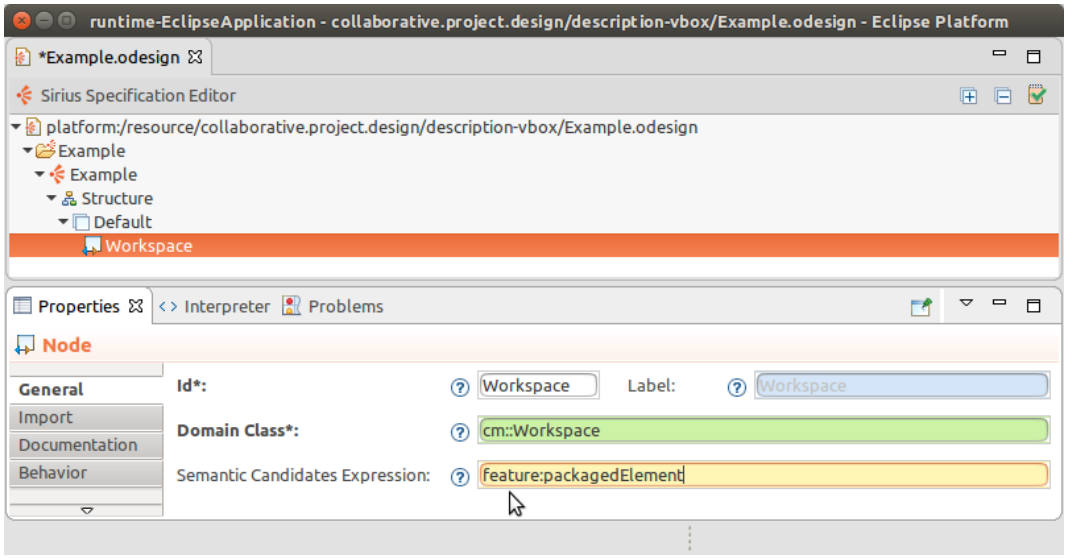


Figura 5.8: Propiedades de un Nodo

La figura 5.8, muestra cómo se configura el estilo de un nodo. Esto permite que los nodos se presenten con distintos tamaños, colores y bordes. También puede configurarse el label de los nodos indicando el tamaño de la letra, formato, ubicación del label y una expresión que es en definitiva lo que se va a mostrar. Esta expresión permite presentar distintos aspectos relacionados con la instancia de la clase, que se está mostrando con el nodo.

Para probar esta representación en los modelos colaborativos, hay que asociar la vista "Example" (el viewpoint "Example") a un proyecto que contenga los modelos creados, como se muestra en las figuras 5.9 y 5.10. En este caso, se muestra un proyecto "Modelos Ejemplos" donde se van a crear los modelos, allí aparece un modelo de sistema colaborativo llamado "Primer.cm".

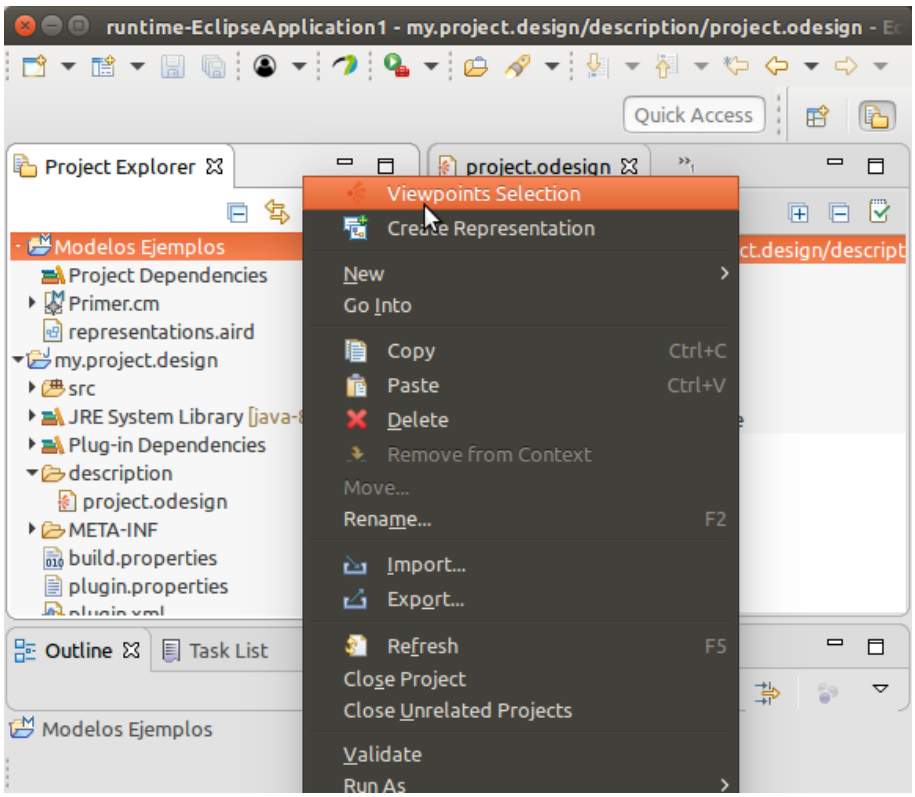


Figura 5.9: Selección de las Vistas de los Modelos

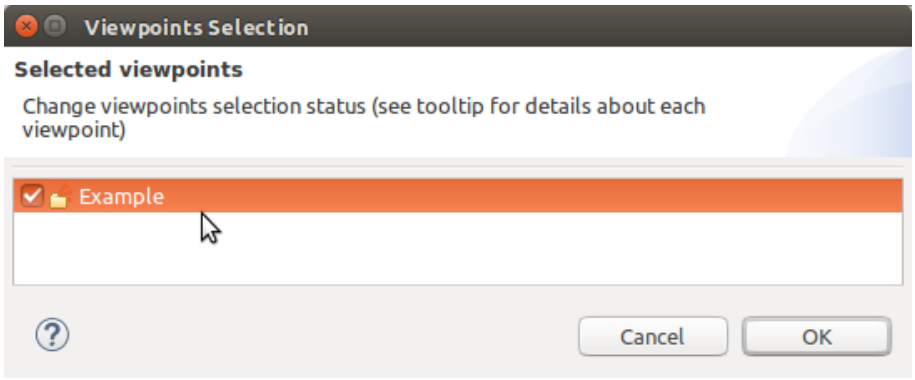


Figura 5.10: Selección de la Vista

En la figura 5.10, se muestra una ventana popup con los viewpoints creados en el workspace. En este caso solamente aparece el viewpoint “Example”.

Una vez elegido el viewpoint para un proyecto, se pueden usar los diagramas creados en ese viewpoint (por ejemplo, “Structure”). Todas las instancias de “Collaborative Model” del proyecto (en este caso: Primer.cm) pueden usar los diagramas. En la figura 5.11, se muestra cómo se crea la representación “Structure” para el modelo colaborativo “Primer.cm”

En la figura 5.12, se puede ver la representación para las instancias de workspace. En el ejemplo

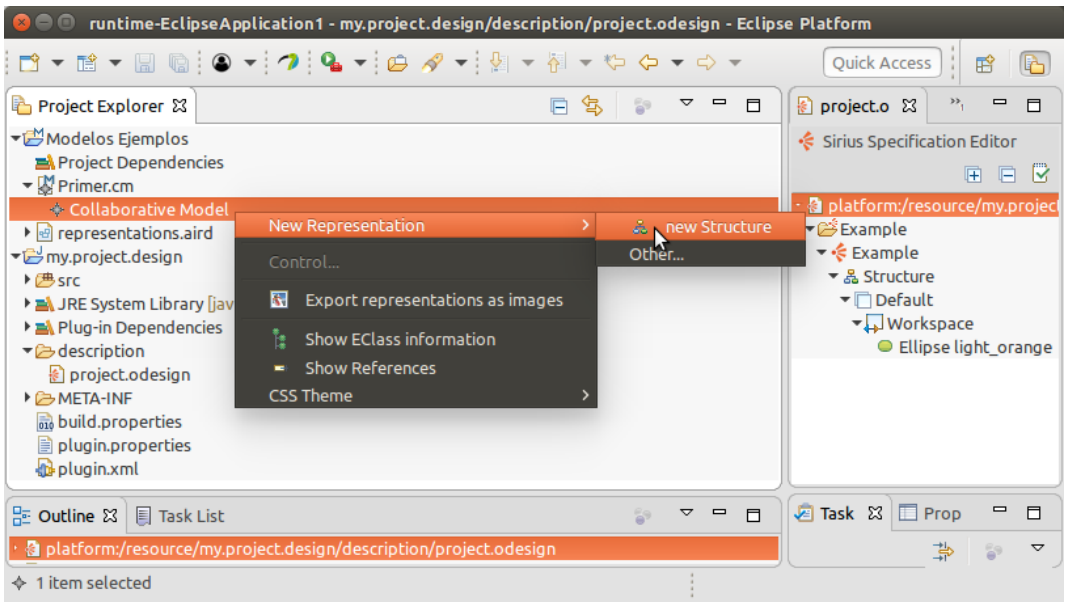


Figura 5.11: Aplicar una representación a un Modelo

se ve que hay tres instancias de Workspace que se representan en el diagrama con la forma de elipse naranja, como se especificó en la figura 5.6, 5.7 y 5.8.

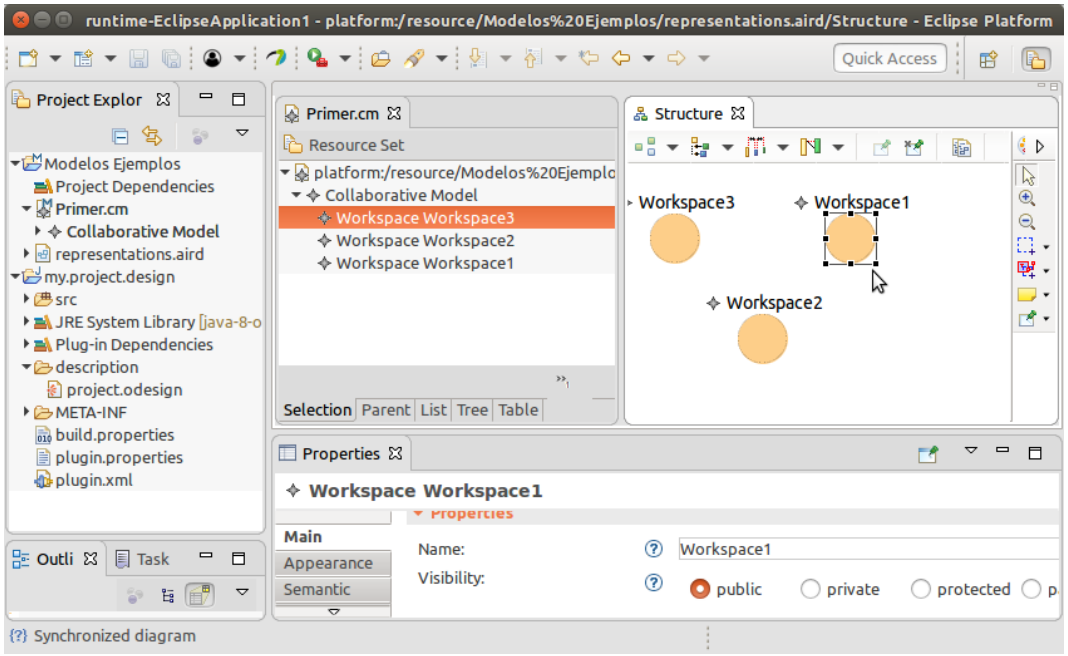


Figura 5.12: Representación de Instancias de Workspace

5.1.3. Ejes en un Diagrama

Además de los nodos, con "Sirius"se pueden agregar ejes a los diagramas, que permiten asociar a los nodos. En la figura 5.13, se muestra cómo se agrega un eje.

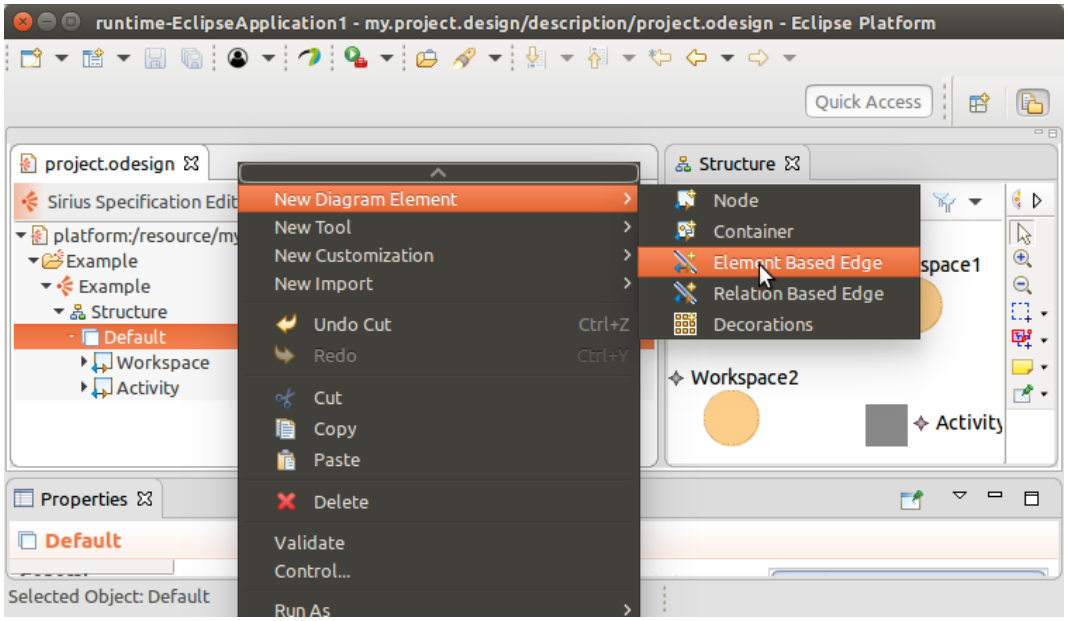


Figura 5.13: Nuevo Eje en el Diagrama

Para especificar un eje, hay que definir los nodos "origen" y "destino", entre otras propiedades. Partiendo del ejemplo de la asociación BelongRelationship expresado en la imagen 5.14, se dibuja el eje como se muestra en la parte superior de la imagen 5.15

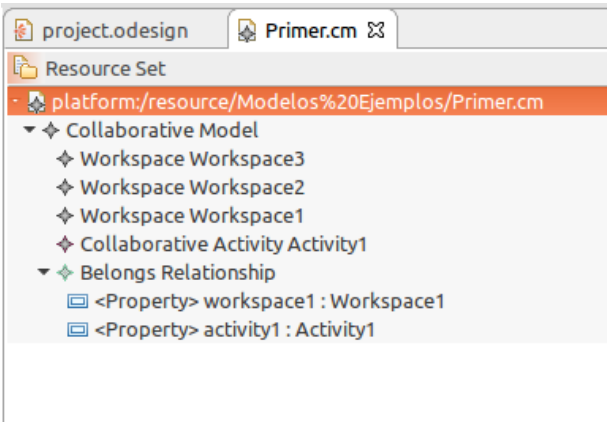


Figura 5.14: Ejemplo de BelongRelationship

En la figura 5.15, se muestran las propiedades de un eje a partir de la clase (BelongRelationship) que representa la relación que hay entre una actividad a un workspace. Las propiedades del eje

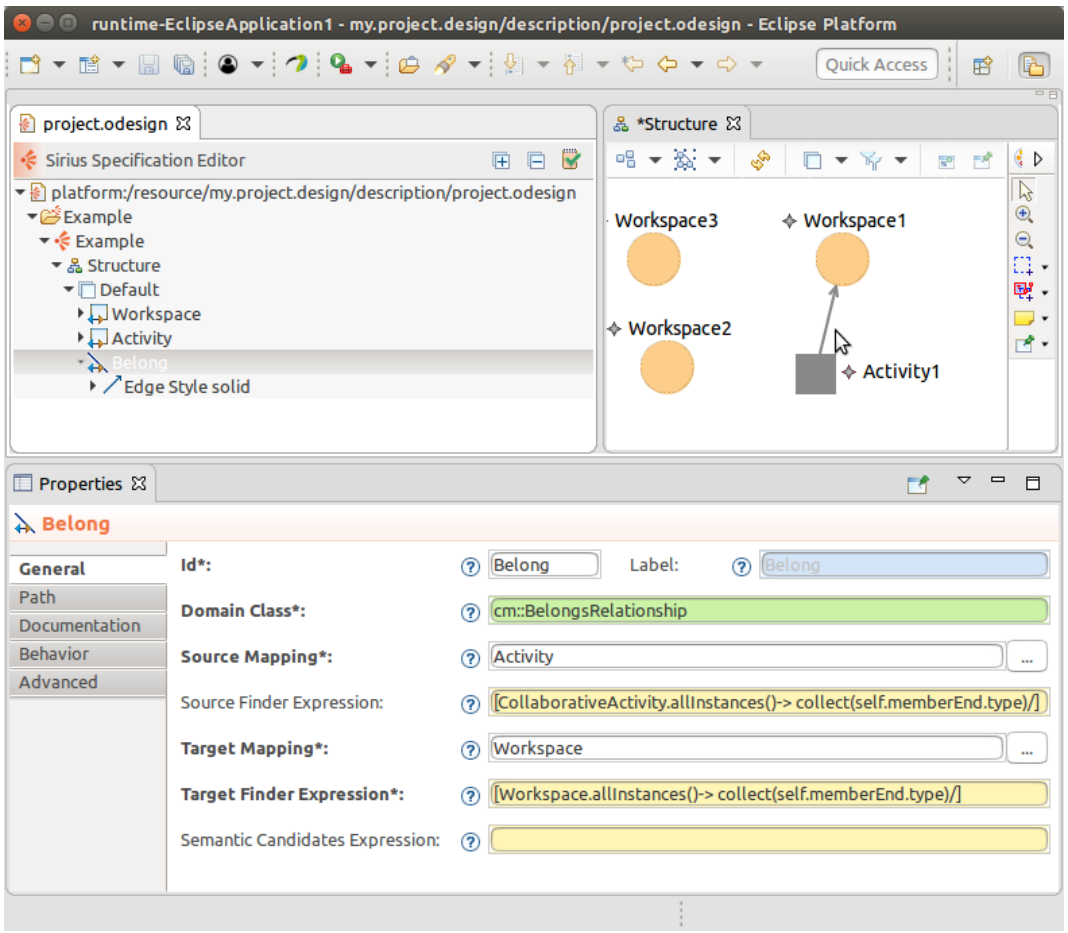


Figura 5.15: Propiedades de un Eje

Belong son:

- **Id**: Belong (El identificador tiene que ser único)
- **Domain Class**: cm::BelongsRelationship (Muestra un eje para cada instancia de BelongsRelationship. Es decir que el eje se basa en un elemento del modelo -Element Based Edge-)
- **Source Mapping**: Activity (Nombre del nodo origen)
- **Source Finder Expression**: [CollaborativeActivity.allInstances()
  -> collect(self.memberEnd.type)/]. (Para encontrar el origen de la asociación busca dentro de todas las instancias de CollaborativeActivity aquellas que aparezca dentro de sus extremos -memberEnd.type-)
- **Target Mapping**: Workspace (Nombre del nodo destino)

- **Target Finder Expression**: [Workspace.allInstances()
  -> collect(self.memberEnd.type)/]
  (Busca el destino dentro de todos los espacios -workspace- que se encuentran en la asociación)

En el ejemplo 5.15, se muestra un detalle de la representación de un eje. Sirius usa el modelo para ver cuál relación tiene que dibujar y lo hace utilizando expresiones OCL o AQL que permiten definir cuál origen y cuál destino se tienen que dibujar, como se muestra en las propiedades Source y Target Finder Expresión.

5.1.4. Paleta de Edición

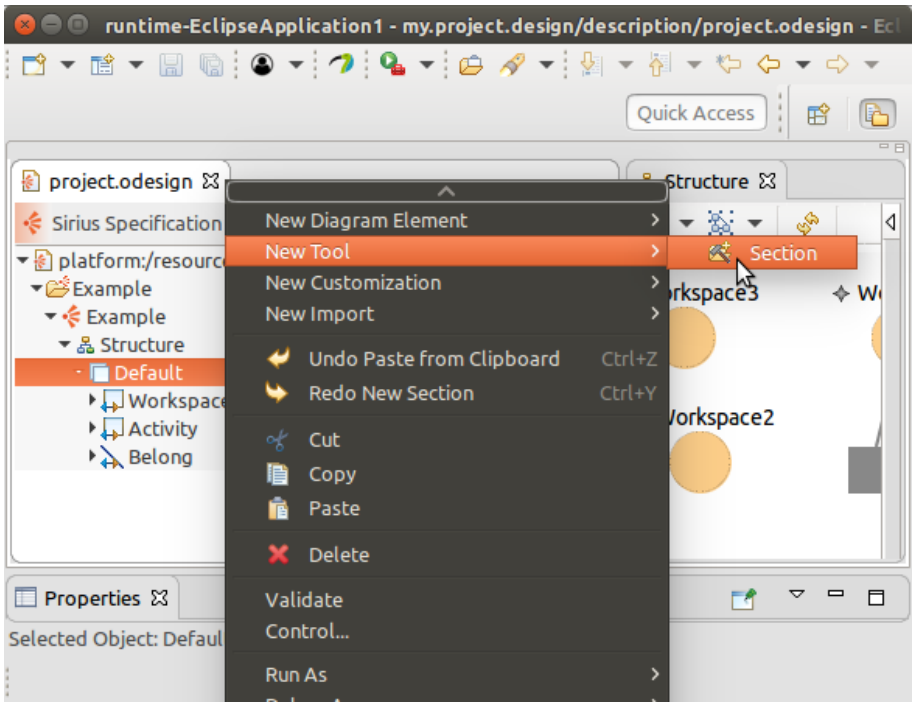


Figura 5.16: Nueva Paleta de Edición

En esta sección se comenta cómo agregar una paleta de herramientas con distintos operadores que permiten incorporar y/o editar nodos y ejes. Para esto, "Sirius" prepara variables y acciones que se ejecutarán cuando el usuario elija algunas de las opciones de la paleta. En la siguiente sección, se explica cómo se agregan nodos y ejes al modelo.

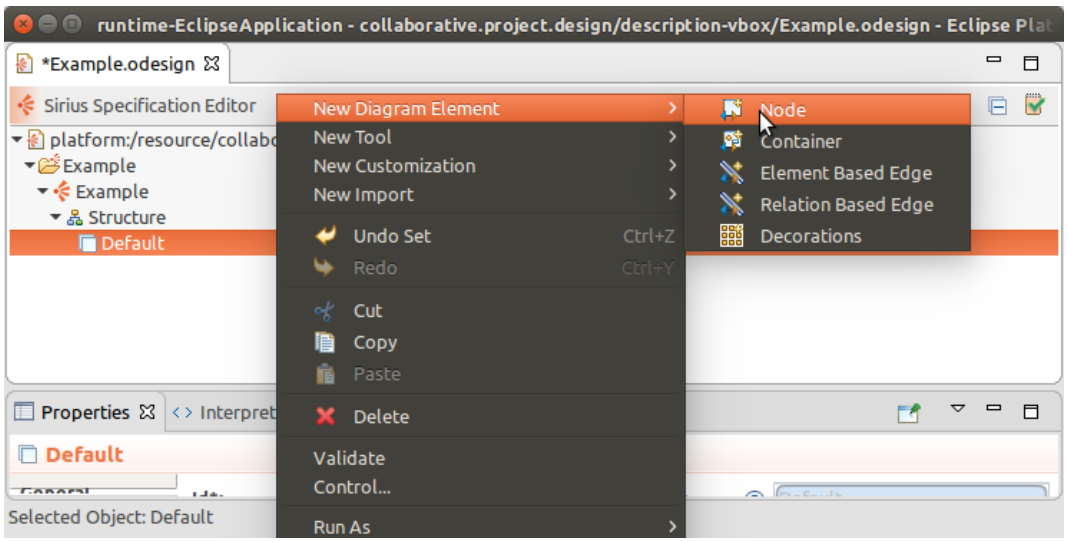


Figura 5.17: Creación de un Nodo

5.1.5. Creación de un Nodo

Para que aparezca un nodo nuevo en el diagrama, hay que crear una instancia asociada al nodo en el modelo. Por ejemplo, para crear un nodo actividad, representado con un cuadrado gris como se especificó en Sirius, hay que crear una instancia de CollaborativeActivity en el modelo. En las figuras 5.17 y 5.18, puede verse cómo se preparan las acciones que se van a realizar para crear un nodo. En el ejemplo, se prepara las acciones para crear un nodo Actividad.

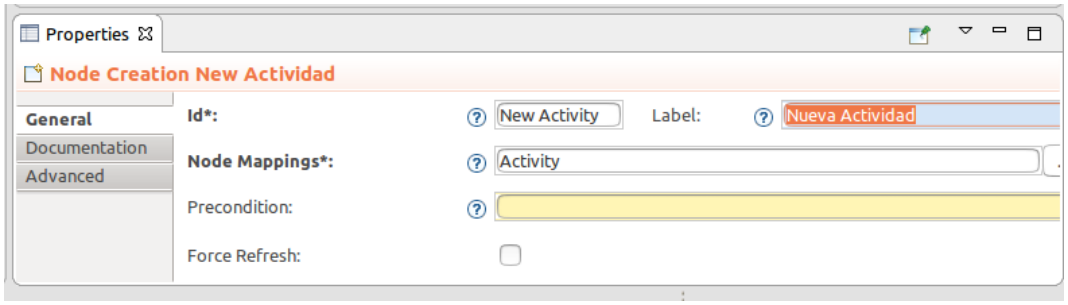


Figura 5.18: Propiedades de la Creación de un Nodo

Propiedades de la herramienta de creación de nodos:

- **Id**: Nombre que identifica a la herramienta de creación de elementos (nodos). Este nombre tiene que ser único.
- **Label**: Nombre usado en la interfaz.
- **Node Mappings**: Nodo que se va a crear. (Activity, en este ejemplo)

- **Precondition**: Expresión que habilita la creación del objeto. La expresión tiene que devolver **true** o **false**.
- **Available variables**:
  - **container**: Objeto que contiene los objetos que se van a crear (ecore.EObject).

Al momento de la creación de un Nodo, se puede ejecutar un conjunto de comandos que especifican en la sección *Begin* de las propiedades. Aquí se definen las operaciones, y en qué contexto se ejecutan. En general, se comienza fijando el contexto, como se muestra en la figura 5.19 (generalmente con la variable "container"), que hace referencia a la instancia donde el usuario haya hecho click. En el ejemplo, donde se crea una instancia de activity, el container es una instancia de CollaborativeModel.

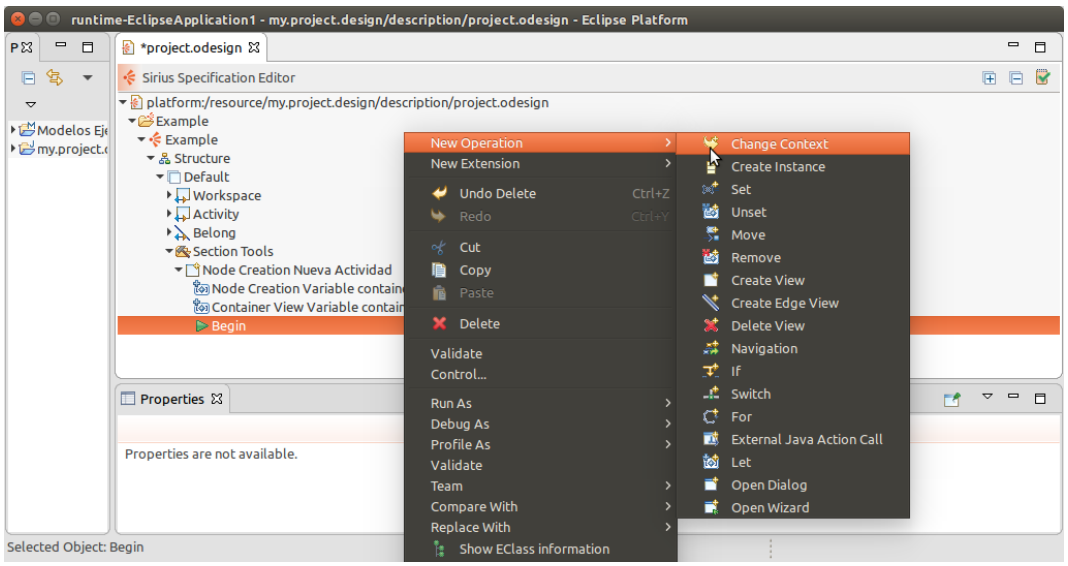


Figura 5.19: Fija el contexto para la creación instancias

- **Reference Name**: Nombre de la propiedad donde se almacenará la nueva instancia creada. En el ejemplo, "packgedElement" es la propiedad de CollaborativeModel que contiene los elementos del modelo.
- **Type Name**: El nombre de la Clase que se va a crear.
- **Variable Name**: Una vez que la instancia es creada, se le va dar el nombre de variable que se especifique aquí.

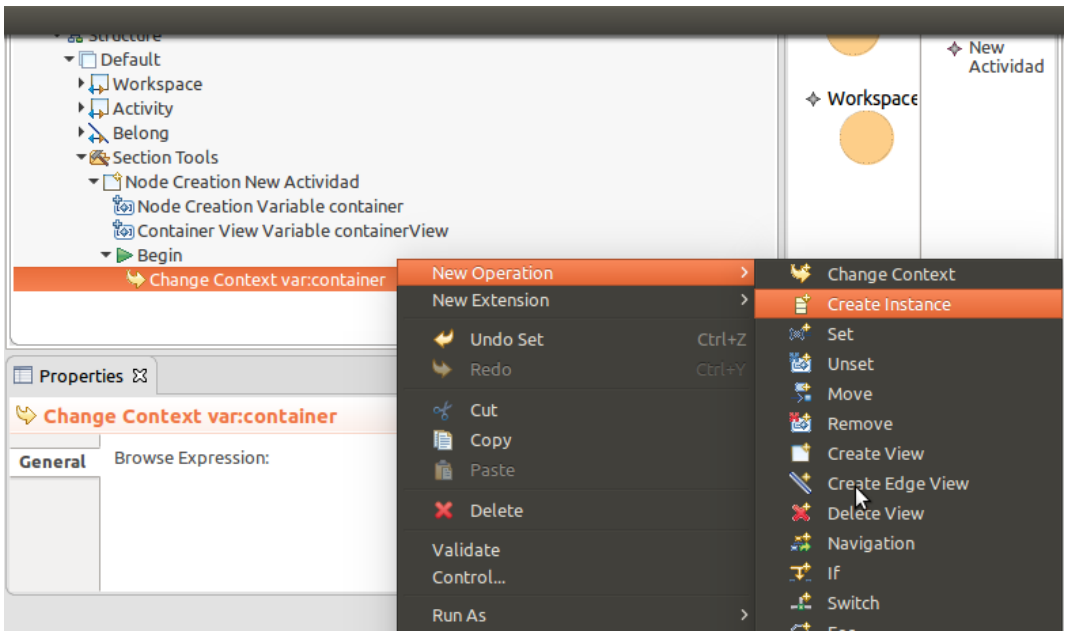


Figura 5.20: Creación de una instancia de CollaborativeActivity

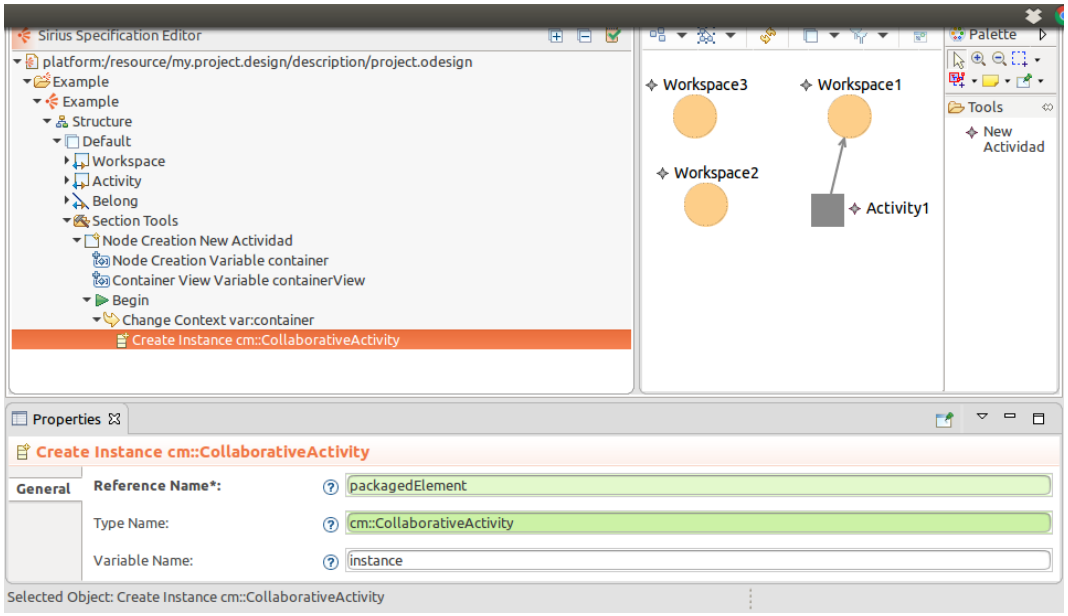


Figura 5.21: Propiedades de la creación de una instancia

Una vez que se crea un elemento, figuras 5.20 y 5.21, se le puede aplicar otras operaciones. Por ejemplo, se le puede dar un valor a su nombre más adecuado, con una expresión AQL que calcula cuántos elementos hay y se le asigna el nombre, como se muestra en la figura 5.22, con la expresión:

['Actividad'.concat(container.packagedElement->
selectByType(cm::CollaborativeActivity)->size().toString())/]

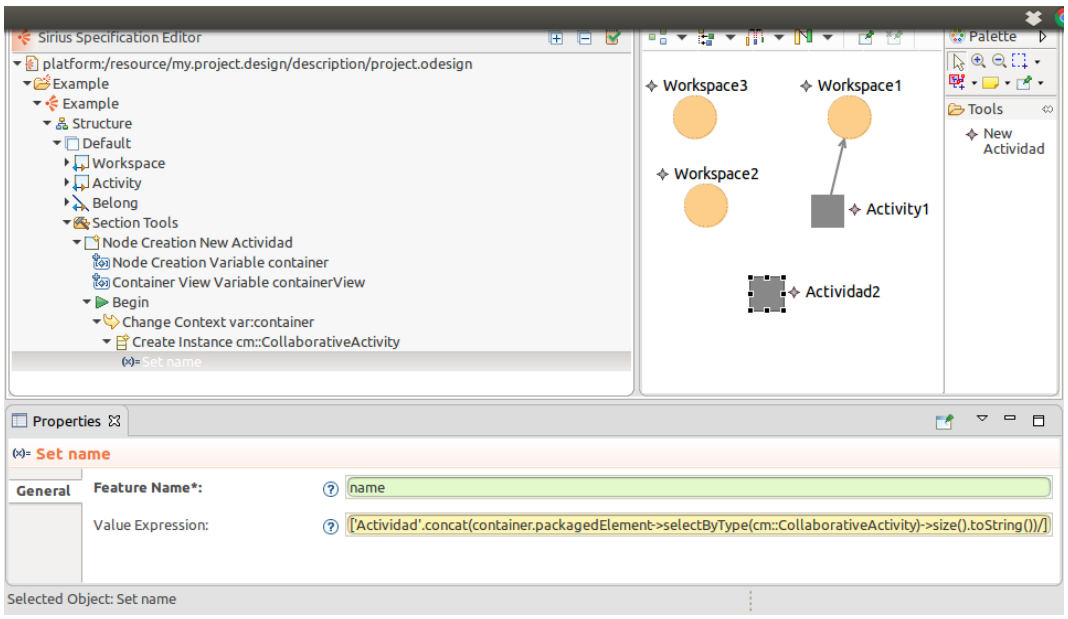


Figura 5.22: Instancia creada con el nombre asignado

5.2. Editores del Lenguaje CSSL

Con las herramientas de Sirius mencionadas, se crearon distintos editores que permiten utilizar el lenguaje CSSL, que sirven para instanciar las metaclases presentadas en el capítulo anterior. Los desarrolladores de sistemas colaborativos pueden crear instancias de CollaborativeModel y con los editores agregarles los elementos colaborativos (Espacios, Roles, Herramientas y Actividades), asociarlos y especificar todas las características del sistema; por ejemplo, incorporar procesos colaborativos, protocolos e información de awareness, que se utilizaran en el sistema.

Los editores, como mencionamos en la sección 5.1, muestran distintos aspectos del modelo y con las paletas se puede crear y/o modificar los elementos mostrados. En la figura 5.23, aparecen los Roles, Procesos y Tipos de Awareness que tiene un Modelo Colaborativo (una instancia de CollaborativeModel).

5.2.1. Dashboard del Sistema

Con el editor que se muestra en la figura 5.23, se pueden crear y editar algunos elementos en el sistema. Puede observarse, que la creación de estos elementos se realiza sencillamente clickeando en el título de los elementos listados. En todos estos casos, se agrega un elemento al listado y luego se le puede cambiar el nombre.

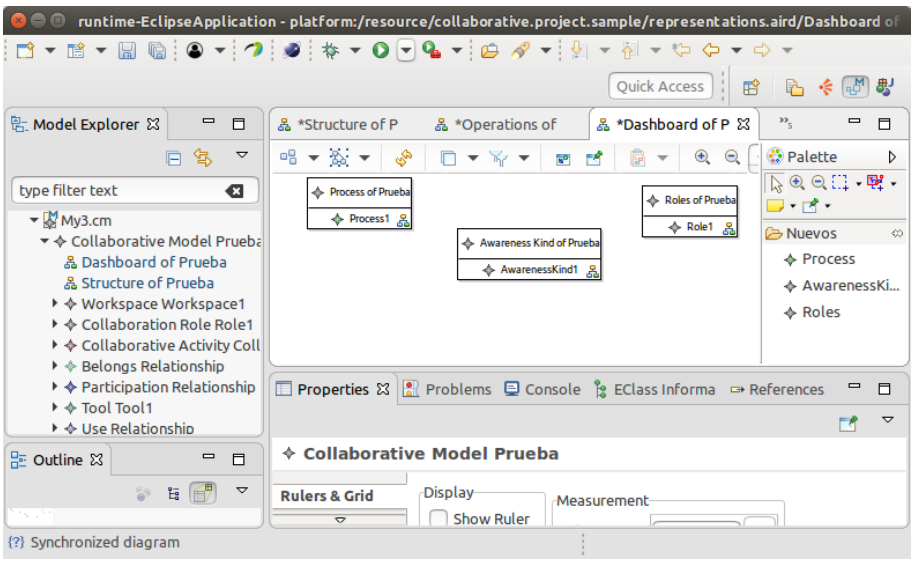


Figura 5.23: Dashboard del Sistema Colaborativo

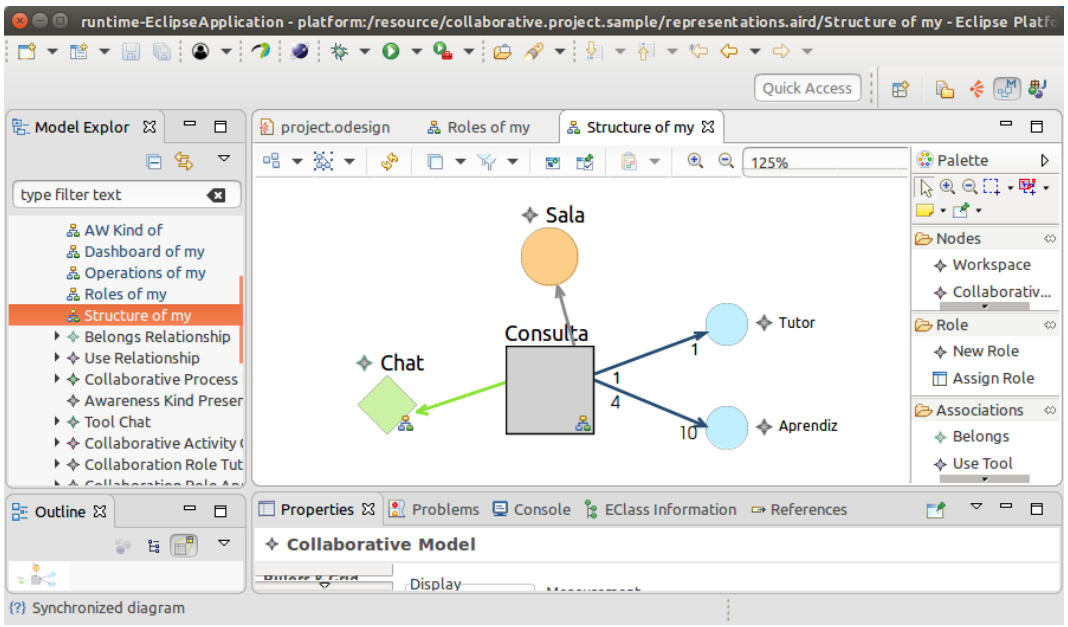


Figura 5.24: Especificación de Entorno Colaborativos (Espacios, Herramientas, Roles y Actividades)

5.2.2. Estructura del sistema

El editor de la estructura muestra una representación para los elementos que definen la estructura del sistema colaborativo (Workspace, CollaborativeActivity, Tool, Role). También, se representan las relaciones entre esos elementos (BelongRelationship, ParticipationRelationship, UseRelationship,). El editor permite crear elementos y asociaciones con el panel que tiene al costado. Para identificar a simple vista a los elementos y asociaciones del sistema, se utilizan

distintas formas y colores como se muestra en la figura 5.24.

La representación de uno de los elementos de la estructura del sistema, el workspace, se muestra en el código xml 5.1. En la primera fila puede verse que la representación "Structure" se ubica en la Metaclase "CollaborativeModel". A partir de allí se definen las capas y los elementos que se muestran en el editor. En la descripción del editor "Structure", se muestra en la capa Default la definición de los nodos de la Metaclase "Workspace" que se encuentran en la colección packageElement del CollaborativeModel. Dicho de forma más sencilla, se muestran todos los workspace del modelo.

XML Code 5.1: Definición de Worksapce

```
<ownedRepresentations xsi:type="description_1:DiagramDescription" name="Structure"
    titleExpression="aql:'Structure of ' + self.name" domainClass="cm.CollaborativeModel">
      <defaultLayer name="Default">
<nodeMappings name="Workspace" semanticCandidatesExpression="feature:packagedElement"
    doubleClickDescription="WorkspaceDoubleClick" domainClass="cm.Workspace">
                      <style xsi:type="style:EllipseNodeDescription" labelSize="12"
                          sizeComputationExpression="4" >
                              <borderColor xsi:type="description:SystemColor" href="
                                  systemColors/@entries[name='black']"/>
                              <labelColor xsi:type="description:SystemColor" href="
                                  systemColors/@entries[name='black']"/>
                              <color xsi:type="description:SystemColor" href="systemColors/
                                  @entries[name='light_orange']"/>
                      </style>
</nodeMappings>
```

En el código xml 5.1 *Definición de Workspace*, se muestra que la representación del nodo para los workspace tiene una forma, tamaño, tamaño de la descripción, color del borde, color de la descripción y el color del fondo, como puede verse en la figura 5.24 para el workspace "Sala".

Además de los nodos que se muestran en los editores y que representan los elementos colaborativos, aparecen ejes o flechas que representan las asociaciones entre los elementos. En la figura 5.25, se muestra la flecha entre una CollaborativeActivity (la Consulta) y un Workspace (la Sala), que representa una asociación BelongRelationship del lenguaje CSSL. En la imagen, se destaca cómo se representa en el modelo la relación mencionada que se está creando con el editor, donde aparece una instancia de BelongRelationship, que tiene dos propiedades que

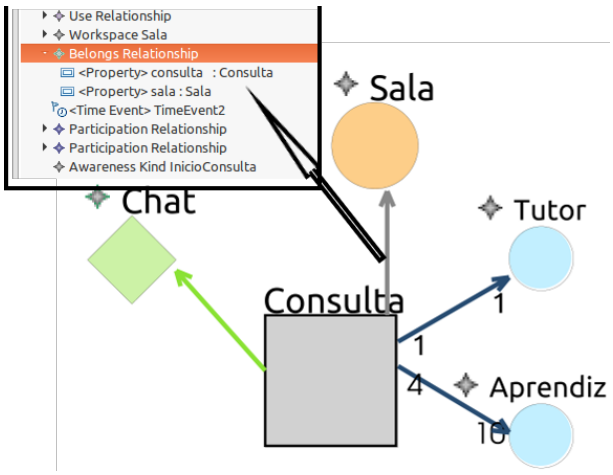


Figura 5.25: Especificación de la Asociación BelongRelationship

indica la relación entre la Sala y la Consulta.

XML Code 5.2: Definición del Eje Belong

```
<edgeMappings name="Belongs" sourceMapping="nodeMappings[name='CollaborativeActivity']"
    targetMapping="nodeMappings[name='Workspace']" targetFinderExpression="[Workspace.
    allInstances()-> collect(self.memberEnd.type)/]" sourceFinderExpression="[
    CollaborativeActivity.allInstances()-> collect(self.memberEnd.type)/]" domainClass="cm.
    BelongsRelationship" useDomainElement="true">
        <style sizeComputationExpression="2">
          <strokeColor xsi:type="description:SystemColor" href="systemColors/@entries[name
              ='gray']"/>
          <centerLabelStyleDescription labelSize="12">
            <labelColor xsi:type="description:SystemColor" href="systemColors/@entries[name
                ='black']"/>
          </centerLabelStyleDescription>
        </style>
</edgeMappings>
```

En el código 5.2, se muestra como hace **Sirius** para dibujar una flecha a partir de las instancias del modelo que se está construyendo. En este ejemplo, se representa un eje con un edgeMappings a partir de la clase BelongRelationship. La construcción de un eje a partir de una instancia de BelongRelationship, que tiene un source a una instancia de CollaborativeActivity y el target una instancia de Workspace. Las instancias las encuentra realizando una consulta al modelo, buscando dentro de las todas las instancias de Workspace, aquellas que formen parte de uno de los extremos de la asociación BelongRelationship (lo mismo hace con la CollaborativeActivity).

Esto lo realiza con la expresión:

targetFinderExpression="[Workspace.allInstances()-> collect(self.memberEnd.type)/]"

Los distintos editores, son una vista (ViewPoint) diferente del mismo modelo. En consecuencia, las acciones que modifiquen el modelo, se visualizarán en todos los editores. Por ejemplo, si se agrega una clase en el editor de la estructura, aparecerá también en los listados del dashboard y viceversa.

5.2.3. Operaciones del sistema

Los modelos colaborativos son instancias de la metaclase CollaborativeModel, que es sub-clase de Model (de UML). Los modelos colaborativos contienen instancias de diferentes clases, entre ellas los elementos colaborativos CollaborativeElement. Estas, tienen un conjunto de operaciones asociadas con la metaclase ElementOperation, como se muestra en la figura 4.5.

5.2.3.1. Operaciones del los Elementos - ElementOperation

Los elementos colaborativos tienen un conjunto de operaciones que luego se le van a asignar a los roles. Los elementos colaborativos pueden ser Workspace, Tool, CollaborativeActivity y SharedObject. Cada uno de ellos cuenta con un conjunto de operaciones que se modelan con la metaclase ElementOperation. Por ejemplo, se puede decir que un Chat (Tool) tiene operaciones para enviar mensajes o archivos. Por otro lado, un Aula (Workspace) puede tener operaciones para entrar o salir de la misma.

Con el editor que se muestra en la figura 5.26, se muestra cómo se puede agregar fácilmente operaciones a los elementos colaborativos (una Actividad en este caso), utilizando la barra de herramientas o clickeando en la estrellita cerca de los nombres de los elementos colaborativos.

5.2.3.2. Operaciones de los Roles - RoleElementOperation

Las operaciones que tienen los elementos colaborativos pueden ser asignadas a los roles que intervienen en el sistema. El diseñador de un sistema colaborativo tiene que definir qué operaciones de los elementos colaborativos pueden ejecutar los roles.

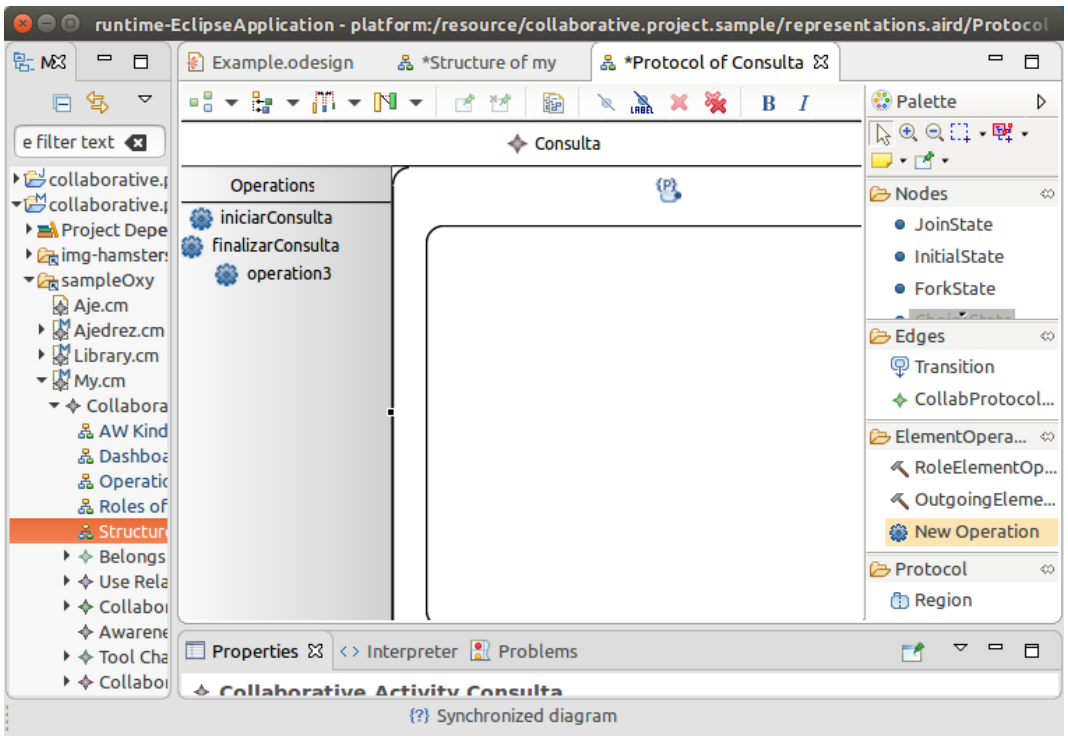


Figura 5.26: Nueva Operación para la Actividad “Consulta”

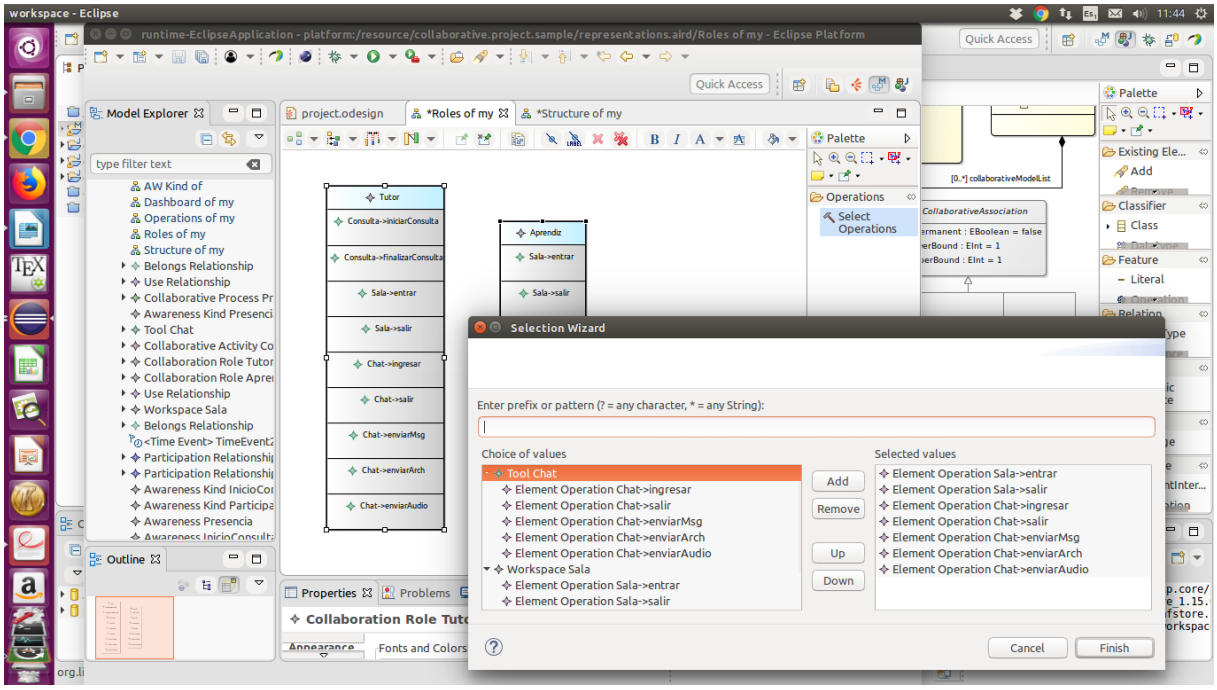


Figura 5.27: Operaciones de los roles

En la figura 5.27, se muestran las operaciones de los roles que participan en modelo, donde también se muestra cómo se asignan a los roles las operaciones inicialmente creadas para los elementos colaborativos. Cómo se especificó en el modelo conceptual del CSSL en el capítulo

XML Code 5.3: Diagrama de Roles

```
<ownedRepresentations xsi:type="description_1:DiagramDescription" name="RoleDiag"
    titleExpression="aql:'Roles of ' + self.name" domainClass="cm.CollaborativeModel">
    <defaultLayer name="Default">
        <containerMappings name="NodeContainer" labelDirectEdit="ownedTools[name='editName
            ']" semanticCandidatesExpression="[self.packagedElement/]" synchronizationLock="
            true" domainClass="cm.CollaborationRole" childrenPresentation="VerticalStack">
          <subContainerMappings name="RoleElementOperation" semanticCandidatesExpression="[
              self.assignedRoleElementOperation/]" domainClass="cm::RoleElementOperation">
            <style xsi:type="style:FlatContainerStyleDescription"
                borderSizeComputationExpression="1" labelSize="10" labelExpression="[self.
                elementOperation.name/]">
              <borderColor xsi:type="description:SystemColor" href="systemColors/@entries[
                  name='black']"/>
              <labelColor xsi:type="description:SystemColor" href="systemColors/@entries[name
                  ='black']"/>
              <backgroundColor xsi:type="description:SystemColor" href="systemColors/@entries
                  [name='white']"/>
              <foregroundColor xsi:type="description:SystemColor" href="systemColors/@entries
                  [name='light_gray']"/>
            </style>
          </subContainerMappings>
```

anterior (figura: 4.5), los roles tienen una colección de operaciones asignadas llamada roleElementOperation).

Para mostrar las operaciones de los roles, se creó en Sirius una vista donde se muestra los roles con sus operaciones. La representación "RoleDiag", como se muestra en código 5.3, está en la capa default y muestra a los roles con un nodo contenedor (NodeContainer) que tiene un conjunto de nodos interiores, las operaciones asignadas. El resultado se muestra en la figura 5.27.

En el caso de contar con una actividad colaborativa "Consulta", que tiene una operación "*iniciarConsulta*", el diseñador tendrá que indicar qué rol podrá iniciar la consulta (por ejemplo: seleccionar al rol "Tutor" para esa operación y eso se realiza con el editor que se muestra en la figura 5.27). Puede verse que en el panel de la izquierda están las operaciones de todos los

elementos colaborativos del sistema, y en el panel de la derecha, están las operaciones que se le asignan al rol, como se muestra en la figura 5.28.

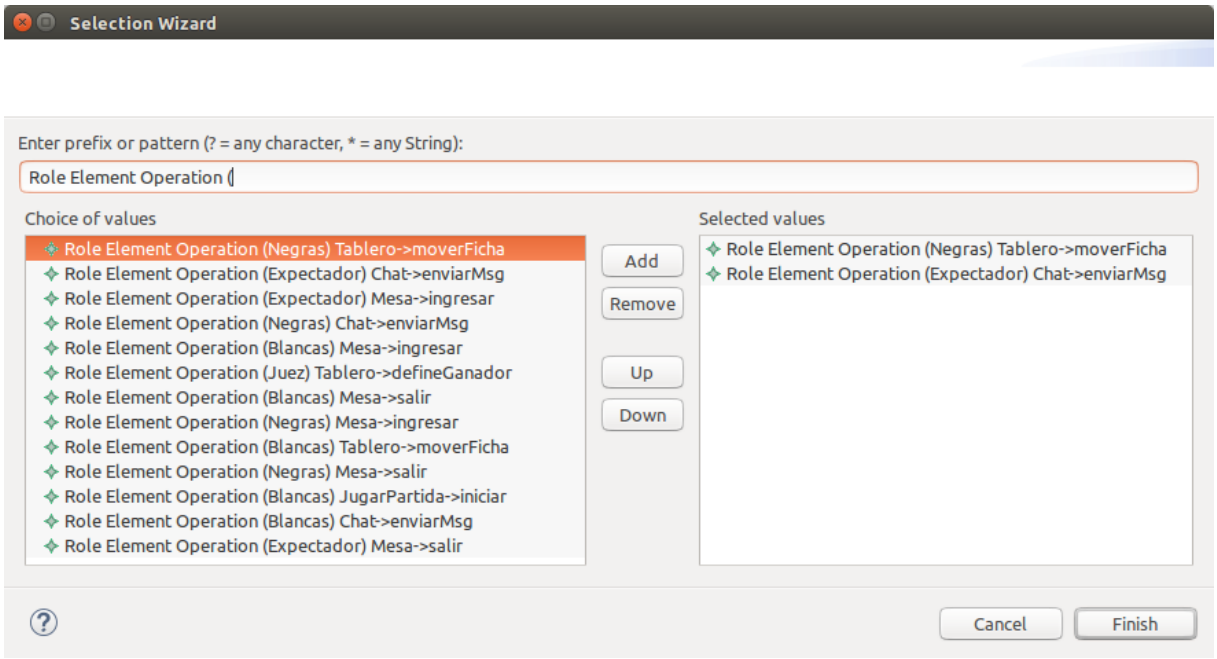


Figura 5.28: Asignación de Operaciones

Sirius tiene paneles que se pueden programar para mostrar elementos del modelo. Este es el caso del panel que se muestra en la figura 5.28, que responde al código xml 5.4, que se utiliza para los Roles del sistema con la expresión container.oclIsTypeOf(CollaborationRole). Esto significa que a este panel de selección de "Operaciones", se les asigna a los roles del sistema.

XML Code 5.4: Panel de Asignación de Operaciones a los Roles

```
<ownedTools xsi:type="tool_1:PaneBasedSelectionWizardDescription" name="Select Operations"
    precondition="[container.oclIsTypeOf(CollaborationRole)/]" forceRefresh="true" iconPath
    ="" candidatesExpression="[uml::Class.allInstances()/]" tree="true" rootExpression="[
    CollaborativeElement.allInstances()-> select(ce:CollaborativeElement | ce.
    elementOperation->size()>0)/]" childrenExpression="aql:self.elementOperation"
    preSelectedCandidatesExpression="[self.oclAsType(CollaborationRole).
    assignedRoleElementOperation->collect(elementOperation)/]">
```

El panel de la izquierda de la figura 5.28, aparece una estructura de árbol y en el primer nivel del árbol, se obtienen todas las instancias de CollaborativeElement, que tengan operaciones asignadas con la expresión:

rootExpression="[CollaborativeElement.allInstances()-> select(ce:CollaborativeElement | ce.elementOperation->size()>0)]".

En el segundo nivel del árbol, se muestran las operaciones de cada uno de los elementos con la expresión:

childrenExpression="[aql:self.elementOperation]"

En el panel de la derecha, aparecen las operaciones asignadas al rol con el que se está trabajando con la expresión:

preSelectedCandidatesExpression="[self.oclAsType(CollaborationRole) .assignedRoleElementOperation->collect(elementOperation)/]"

Más adelante se verá qué efecto tiene que se ejecute alguna de estas operaciones por los roles, por ejemplo, que se actualice alguna información de awareness.

5.2.3.3. Awareness relacionado al entorno

En los capítulos anteriores, se definió el awareness como información que muestra el sistema sobre el estado de la colaboración. Por ejemplo, si hay otros usuarios conectados, dónde se encuentran o qué están haciendo. Para especificar la información de awareness que maneja el sistema, el diseñador tiene que modelar dos aspectos del awareness. Por un lado, se define la propiedad del awareness *"shownIn"* que determina donde se muestra la información de awareness. Por otro lado, con la propiedad *"source"*, el diseñador indica qué eventos activan la actualización de esa información.

La propiedad *"shownIn"* refiere a los elementos colaborativos que mostrarán la información de Awareness, por ejemplo, los que se especifican en la figura 5.24, donde Espacios (Workspace), Herramientas (Tool) y Actividades Colaborativas (CollaborativeActivity) son componentes visibles en el sistema. Es decir, son los elementos que aparecen en la interfaz del sistema; por ejemplo, en ventanas, cuadros o widgets en la pantalla. Por ejemplo, en la figura 5.29 se puede muestra la "**Presencia**" (el awareness de presencia) en la **Sala**.

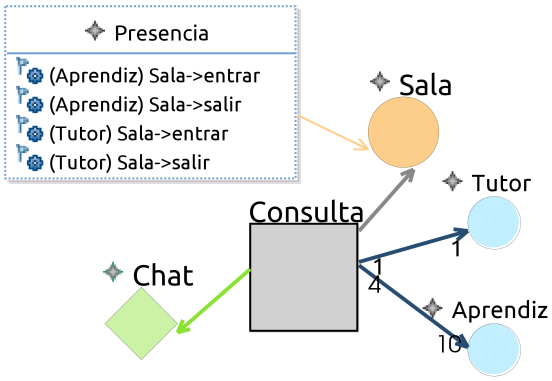


Figura 5.29: Ejemplo de Awareness

Usando la propiedad *"source"*, el diseñador del sistema define los eventos que disparan la actualización de la información de awareness. En la figura 5.29 se modeló que la información se actualizará cuando los roles ejecuten las operaciones de *"entrar"* o *"salir"* del espacio Sala.

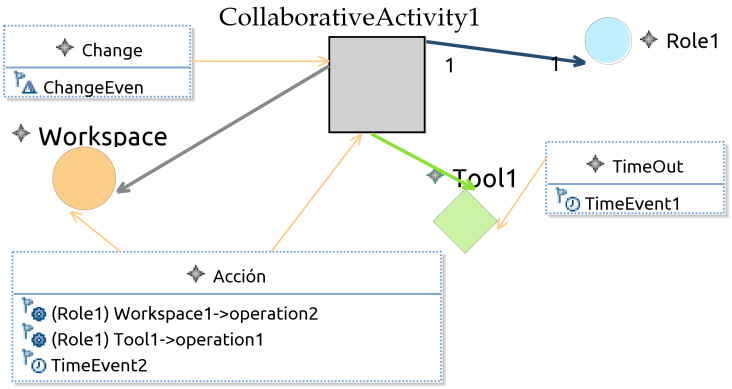


Figura 5.30: Especificación del "source" y "showIn"

En la figura 5.30, se muestran distintas alternativas en especificación del awareness. Por un lado, en el caso del awareness *"Change"*, se especificó que se mostrará en la actividad colaborativa cuando se dispare el evento de cambio *"ChangeEvent"*. Otro caso similar es el awareness *"TimeOut"*, que se actualiza en la herramienta *"Tool1"* y se actualiza a través de un evento de tiempo *"TimeEvent1"*. Por último, en el caso del awareness de *"Acción"*, se muestra que se puede mostrar esta información en más de un elemento colaborativo (en el Workspace1 y en la CollaborativeActivity1) y hay una variedad de eventos que lo mantienen actualizado (en el ejemplo se especificó que se actualizará cuando el *"Role1"* ejecute la operación2 del *"Workspace"*, o la operación1 de la herramienta *"Tool1"* o se dispare el evento *"TimeEvent2"*).

Los autores que introdujeron el concepto de **Awareness**, en los trabajos de Gutwin [7, 8] y de Dourish [6, 9], lo vinculan a ciertas preguntas que responden a las necesidades del sistema colaborativo que se está diseñando. En la tabla 5.1 se listan los elementos de awareness y las preguntas que pueden sugerir los mismos, en base a la tabla definida por Carl Gutwin [7].

Awareness	**Pregunta Relevante**
Presence	¿Quiénes están en el entorno, el workspace, la actividad colaborativa o el artefacto?
UserLocation	¿En qué lugar (espacio, actividad colaborativa, artefacto, etc.) se encuentra un usuario determinado?
Density	¿Qué densidad de usuarios en cada lugar del entorno?
UserInformation	¿Qué información conocemos del usuario?
Rol	¿Qué rol tiene el usuario en cada momento?
ActivityLevel	¿Qué nivel de actividad tienen los usuarios?
UserAction	¿Qué acción está haciendo el usuario?
PlaceAction	¿Qué acción se está haciendo en un lugar del entorno (espacio, actividad colaborativa, artefacto, etc.)?
UserHistory	¿Cuál es la historia de acciones de un usuario?
Intentions	¿Qué harán los usuarios?
UserBookmarks	¿Cuáles son los bookmarks de un usuario?
Expectations	¿Qué acción se considera que hará luego un usuario considerando el estado actual?
Changes	¿Qué objetos han cambiado?
Objects	¿Cuál es el estado de algún objeto?
Visibility	¿Cuál es la visibilidad de algún usuario? ¿Qué objetos puede ver?
Abilities	¿Qué sabe hacer el usuario?
Influence	¿Cuál es el campo de influencia del usuario?
ObjectsUsers	¿Qué objetos están siendo usados y por quién?
ObjectHistory	¿Qué cambios se han realizado sobre el objeto y quién lo realizó?

Tabla 5.1: Elementos de Awareness

5.2.4. Modelado de la Dinámica del Sistema

Una de las características destacables de los sistemas colaborativos, es que atiende a las necesidades de un grupo de usuarios. Esto se refleja en que las acciones que realizan los usuarios, modificando el estado del sistema y eso se muestra en las interfaces de los participantes de la colaboración. En este sentido, es necesario contar con diseños para describir la participación de los usuarios. El lenguaje CSSL cuenta con dos recursos para modelar la dinámica del sistema; por un lado, los procesos colaborativos y por otro, los protocolos. Ambos pueden tener awareness asociado, donde los diseñadores pueden describir qué eventos y operaciones actualizan esa información.

5.2.4.1. Procesos Colaborativos

El proceso colaborativo está compuesto por un conjunto de actividades colaborativas, que son las tareas que un grupo de roles de usuarios realiza con una o más herramienta colaborativa, para llegar a un objetivo determinado. Los procesos colaborativos definen el orden en que las actividades se llevan a cabo. Como puede verse en la figura 5.31 en la especificación de un proceso, pueden aparecer nodos de control -por ejemplo, decision, join y fork- que permiten armar procesos con caminos en paralelo, loops o sincronización de actividades. En la industria en general, se utilizan procesos de producción (workflow) para organizar las tareas de los empleados. Los procesos colaborativos, incluyen la posibilidad de que las tareas intermedias sean actividades colaborativas.

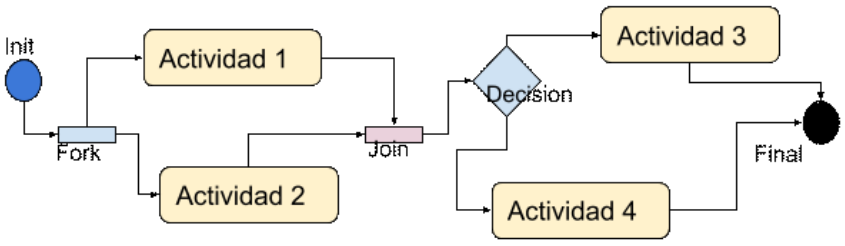


Figura 5.31: Notación de Procesos en UML

Con el editor de procesos que se muestra en la figura 5.32, se pueden agregar actividades y conectarlas con ejes que pueden involucrar operaciones que ejecutan los roles o eventos que se disparan en el sistema.

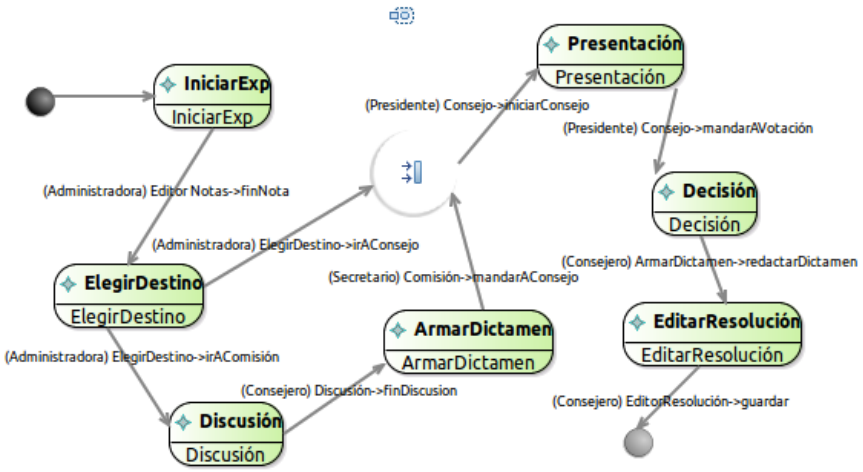


Figura 5.32: Especificación de Procesos Colaborativos

Los procesos colaborativos se representan de forma similar a los diagramas de actividad de UML, donde los nodos son actividades colaborativas o nodos de control que permiten sincronizar actividades. Los ejes (edges) pasan el control de un nodo a otro y se activan cuando algún rol ejecuta alguna operación.

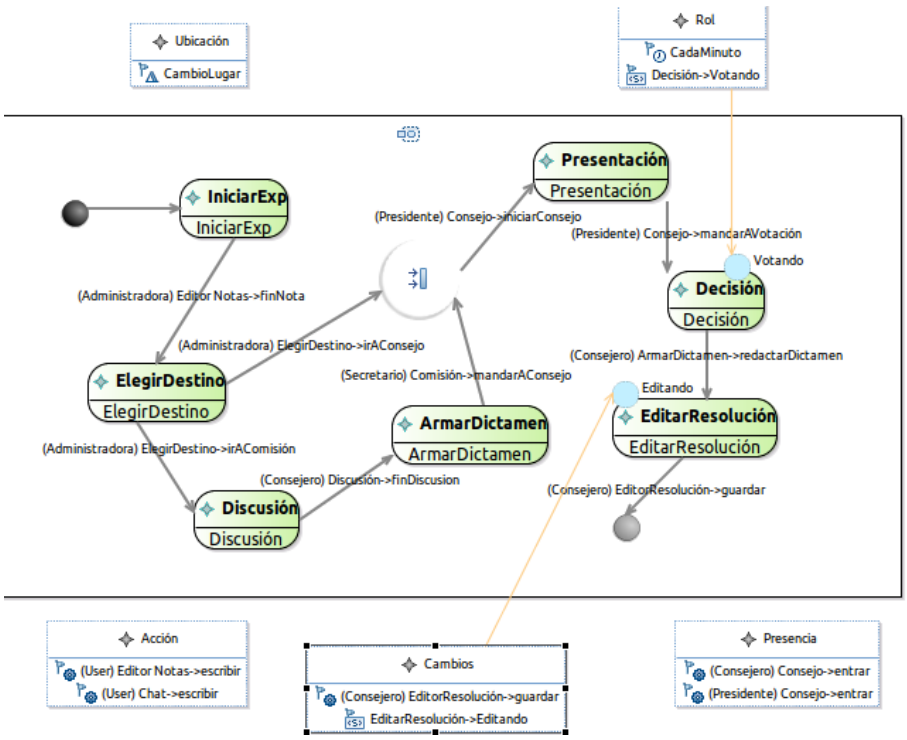


Figura 5.33: Especificación de Procesos Colaborativos con Awareness

5.2.4.2. Awareness relacionado con el proceso colaborativo

En la sección anterior, se mencionó que el awareness tiene la propiedad "shownIn" que indica donde se muestra el awareness. También tiene la propiedad "source" que especifica qué eventos originan el awareness (y la actualización del mismo). Una acción/operación de un usuario puede provocar que se muestre cierta información de awareness o cuando se inicia o finaliza una actividad colaborativa puede disparar que se muestre alguna información de awareness. Por ello es que los procesos suelen tener eventos que originan el awareness. En la figura 5.33, se muestra cómo se incluye el awareness en los procesos colaborativos.

5.2.4.3. Protocolo

Las actividades colaborativas, por ser subclase de Class de UML, pueden tener un comportamiento asociado (behavioral feature). Para modelar el comportamiento, se utilizan las máquinas de estados de UML. Con ellas se define que una actividad colaborativa puede tener distintos estados, y durante el desarrollo de la colaboración se puede cambiar de estado a partir de lo que vaya pasando dentro de la actividad colaborativa. Los elementos principales de las máquinas de estado son: vértices y transiciones. Dentro de los vértices están los estados y pseudoestados, que permiten modelar situaciones especiales dentro de la actividad. Dentro de los pseudoestados, están init, fork, join, choice, terminate. La notación de máquinas de estado poder verse en las figuras 5.34.

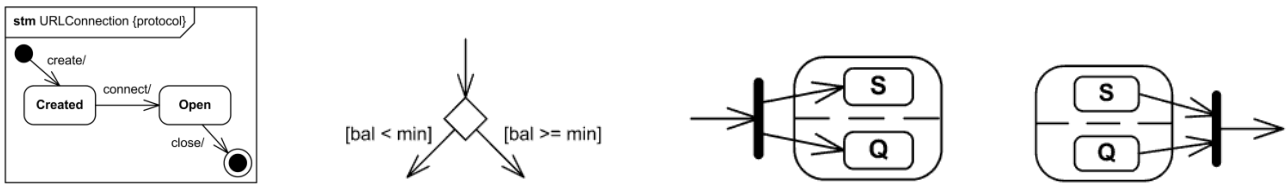


Figura 5.34: Notación de StateMachine en UML

En el metamodelo, se entiende al comportamiento de una actividad colaborativa como una máquina de estado, en particular una ProtocolStateMachine. Un protocolo es un conjunto de reglas establecidas que modelan la interacción de los usuarios/roles dentro de una actividad colaborativa.

Los nombres de los roles en este nivel deben ser descriptivos en función del comportamiento específico dentro de la actividad colaborativa. Esto significa que los roles son específicos de la

actividad, ya que un usuario puede tener un rol fuera de la actividad y otro cuando ingresa a la actividad. Por ejemplo, el rol docente (fuera de la actividad) puede en alguna actividad concreta puede ser observador -por ejemplo, cuando un alumno está presentando algún tema-.

En definitiva, la actividad colaborativa podrá tener distintos estados durante su ejecución y dichos estados están definidos por las operaciones que los usuarios/roles pueden ejecutar en cada estado. Hay que tener en cuenta que una acción que realice un usuario/rol, puede provocar un cambio de estado, como se detalla en la figura 5.35.

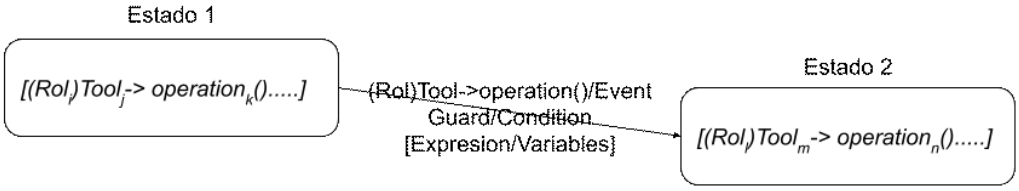


Figura 5.35: Notación de Estados y Transiciones

Por ejemplo: cuando el rol docente le otorga la palabra al alumno, cambia el estado de la actividad y entonces el alumno podrá enviar su mensaje. Las transiciones también pueden ser originadas por otros eventos como el paso del tiempo o algún cambio en algunos valores como se muestra en la figura 5.35.

La transición también tiene una guarda o condición y una expresión donde se pueden ejecutar pequeñas porciones de código. Una guarda representa un conjunto de condiciones, en base al estado de las variables que representan el estado actual. La expresión es una secuencia de comandos que se ejecutan cuando una transición se lleva a cabo. Esta secuencia de comandos en general son asignaciones, que permite actualizar el estado de las variables que se podrán utilizar en los próximos estados.

5.2.4.4. Ejemplo de protocolo usando variables

El protocolo de ejemplo que se llama “OneTwo” se basa en el protocolo debate donde dos roles pueden hablar por turnos. El objetivo del protocolo OneTwo establece que los participantes identificados con los roles One y Two alternan los turnos en un Chat cuya única operación es “send”. El rol One debe esperar a que el rol Two ejecute send dos veces en el Chat, para poder

contribuir nuevamente. En el estado inicial, cualquier participante de cualquiera de los grupos puede enviar un mensaje (Chat.send). La primera transición permite inicializar la variable S, que representa la cantidad de veces que contribuyeron los participantes del rol Two antes que participe uno del rol One. Se puede observar en la figura 5.36, que el protocolo comienza en un estado "Ambos" en el que tanto los roles One como Two, pueden ejecutar la operación send con el Chat. En el protocolo, se va a ir contando con la variable S la cantidad de participaciones seguidas que tuvo el rol Two. A partir de ese momento existen dos transiciones posibles para el protocolo:

- **Estado 1:** Un participante del rol One ejecuta la operación send.
- **Estado 2:** Un participante del rol Two ejecuta la operación send.

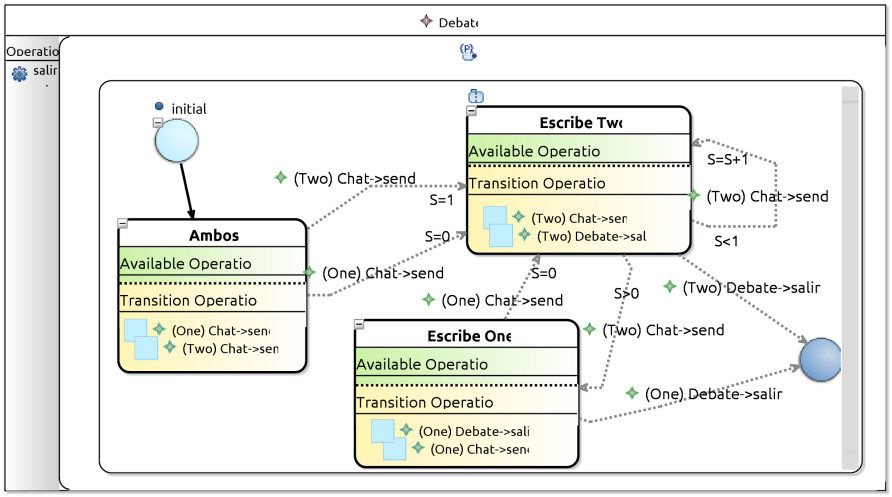


Figura 5.36: Ejemplo de Protocolo OneTwoChat

En ambos casos, en el próximo estado, "Escribe Two", sólo algún participante del rol Two podrá ejecutar la operación send con la herramienta Chat. En el caso que la transición se haya originado por un participante del rol One, la variable continúa con su valor, 0. En cambio, el valor será 1 en caso que la transición haya sido motivada por el send de algún participante del rol Two. Cuando el rol Two mande un mensaje existen dos posibilidades. La primera, permite que se vuelva al mismo estado cuando la variable $S < 1$. En ese caso a S se le suma 1. La segunda, cuando $S > 0$ se pasa al estado "Escribe One" donde el rol One podrá participar.

En el estado "Escribe One", el rol One puede enviar mensaje y cuando lo hace, se pasa al estado "Escribe Two" y se pone en 0 la variable S para que se inicie la cuenta de las participaciones

del rol Two.

5.2.4.5. Awareness y los Protocolos

Como el caso de los procesos colaborativos, los protocolos también tienen asociados muchos eventos que pueden disparar alguna información de awareness. Una acción/operación de un usuario puede provocar que se muestre cierta información de awareness, o cuando se cambia de estado o se inicia/finaliza un protocolo, puede disparar que se muestra alguna información de awareness. En los protocolos hay otros eventos que pueden relacionar los valores que tienen las variables o eventos asociados al paso del tiempo. En la figura 5.37 se muestra el caso de los protocolos con los awareness donde algunas operaciones originan su actualización.

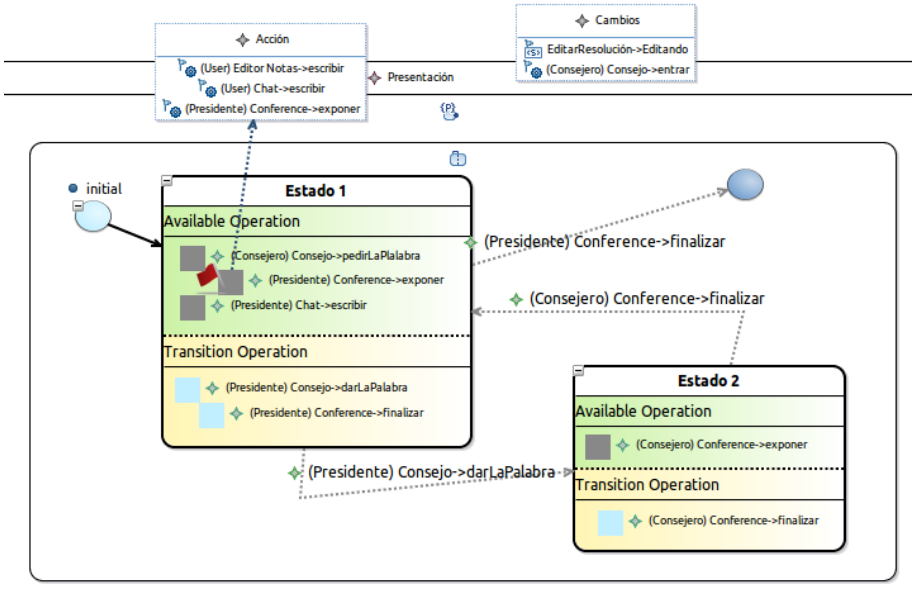


Figura 5.37: Ejemplo de Protocolo con Awareness

5.3. Casos de Modelado

Para entender los beneficios de contar con un Metamodelo que permite instanciar Modelos de sistemas colaborativos, se presenta en esta sección un conjunto de casos que habitualmente existen en los sistemas colaborativos.

5.3.1. Caso 1: Estructura del Sistema

5.3.1.1. Herramienta con Operaciones:

Se empieza por una herramienta colaborativa que tiene un conjunto de operaciones. En este caso es el "Chat" que se encuentra habitualmente en los sistemas colaborativos. Las operaciones que están disponibles en la mayoría de los Chat son: ingresar, salir, enviar mensaje, enviar archivo, enviar audio, etc. y se especifican de forma similar a las operaciones de las clases de UML, como se muestra en la figura 5.38.

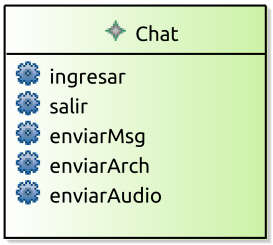


Figura 5.38: Operaciones del Chat

Por otro lado, los usuarios que participan en los sistemas colaborativos realizan distintas actividades colaborativas. Por ejemplo, un sistema que permite que dos usuarios se conecten a una actividad colaborativa que se llama "Consulta". Esta actividad tiene por un lado; dos roles que participan: uno es el "Tutor" y otro es el "Aprendiz" - en el diseño que se muestra en la figura 5.39 puede verse que pueden participar un rol "Tutor" y entre cuatro y diez "Aprendices"-. También se indica que para la "Consulta" se modela que estos roles utilizarán al Chat para comunicarse.

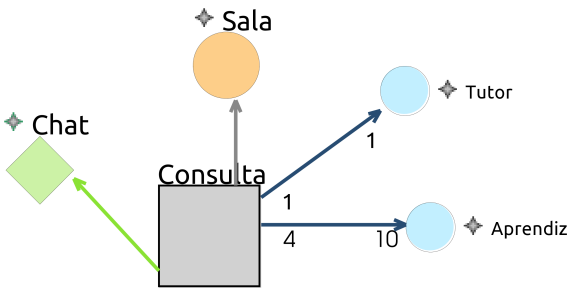


Figura 5.39: Modelo de la Sala

Una actividad colaborativa también puede tener operaciones específicas: por ejemplo, para la

actividad “Consulta”, se puede tener las siguientes operaciones: “iniciarConsulta” y “finalizarConsulta”. De la misma manera el elemento colaborativo “Sala”, también puede tener operaciones propias y por ser un espacio, los usuarios van a poder entrar o salir del mismo. Para los espacios y las actividades se especifican las operaciones de igual manera que las herramientas. En la figura 5.40 se muestran las operaciones de los elementos colaborativos del ejemplo.

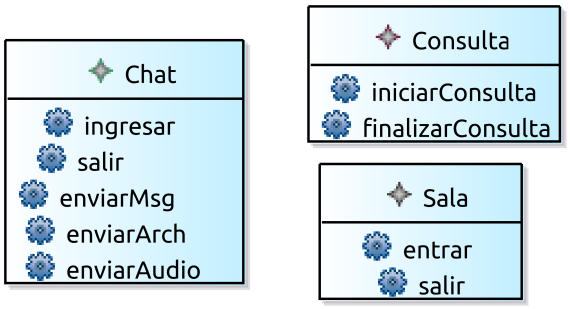


Figura 5.40: Operaciones de los elementos colaborativos

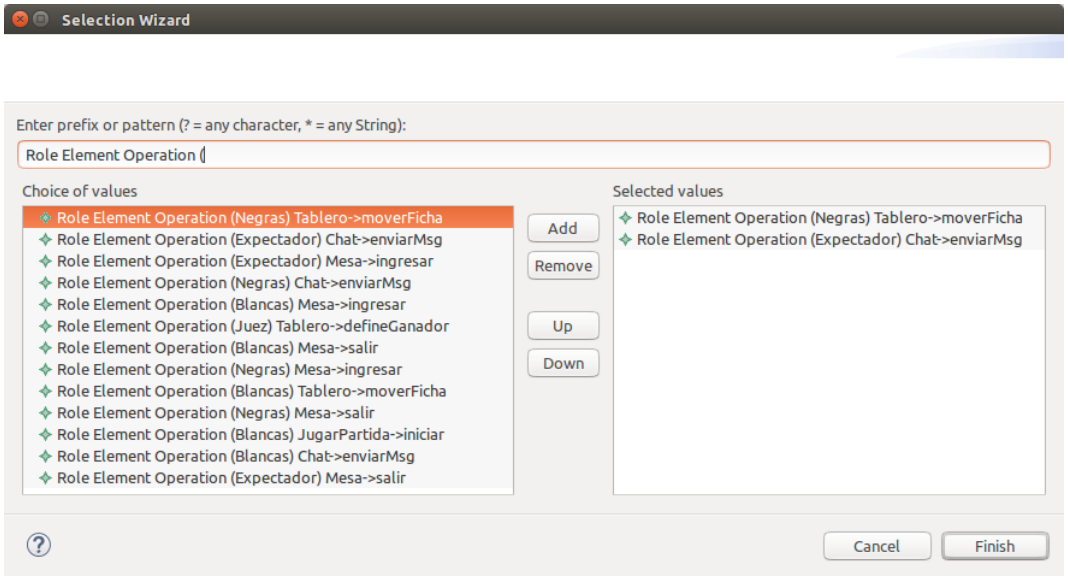


Figura 5.41: Panel de asignación de Operaciones

Una vez que se definen las operaciones de los elementos colaborativos, se continúa indicando cuales de ellas pueden ejecutar cada uno de los roles del sistema. En el ejemplo, se modela que el rol tutor es el encargado de iniciar y finalizar la consulta; las operaciones del chat pueden ser ejecutadas por ambos roles. Puede verse en la figura 5.41 cómo se asignan las operaciones a los roles, donde se arma la terna de rol, elemento colaborativo y operación de la siguiente manera: **(role)element->operation()** y en la imagen 5.42 se muestra cómo quedaron los roles con las

operaciones asignadas.

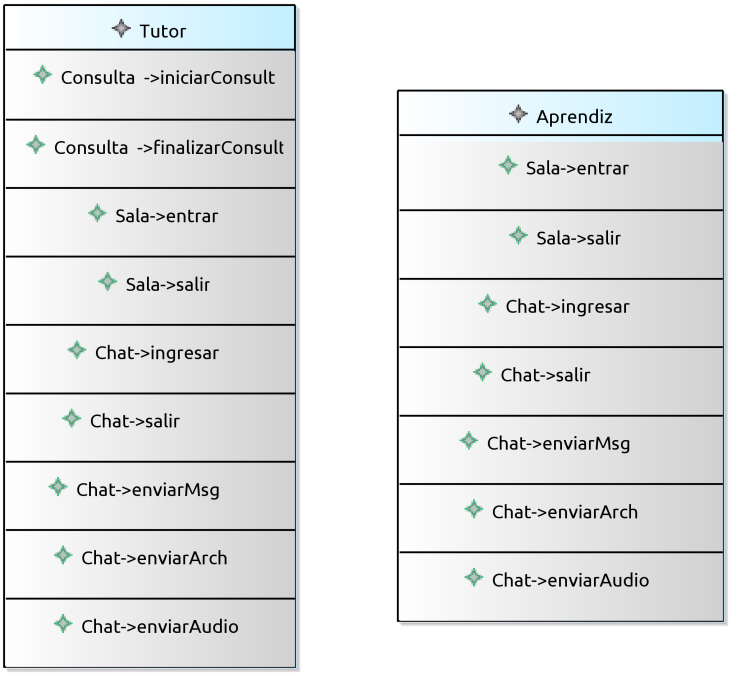


Figura 5.42: Operaciones asignadas a los roles

El awareness es un concepto específico de los sistemas colaborativos, ya que no aparece en otro tipo de sistemas. Se muestra en esta sección como se modela la información de awareness relacionada con la sala, la consulta o el chat. En el ejemplo se quiere mostrar la presencia de los usuarios en la sala. En este caso, el modelo va a tener un tipo de awareness llamado "presencia". Luego se modela que en la "sala" se va a mostrar la presencia de los usuarios. En el modelo quedaría como se muestra en la siguiente figura 5.43.

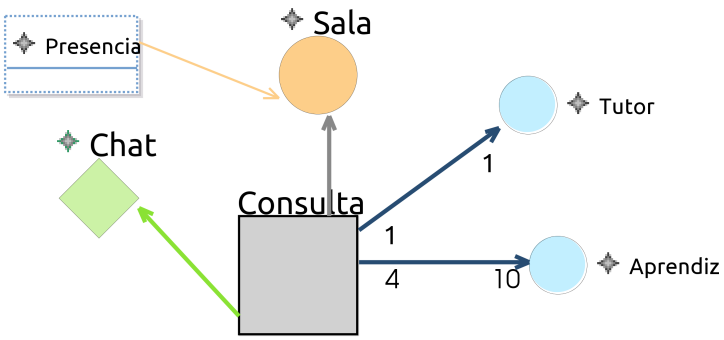


Figura 5.43: Awareness de presencia en la Sala

Por otro lado, se modela qué eventos van a disparar que se actualice la información de awareness

de presencia, que se origina con distintos eventos del sistema o eventos relacionados con las operaciones que realizan los usuarios. El editor permite asignar distintos eventos y cómo se vinculan con el awareness que se mostrará en otro elemento colaborativo. En la figura 5.44 se muestra cómo queda el diseño.

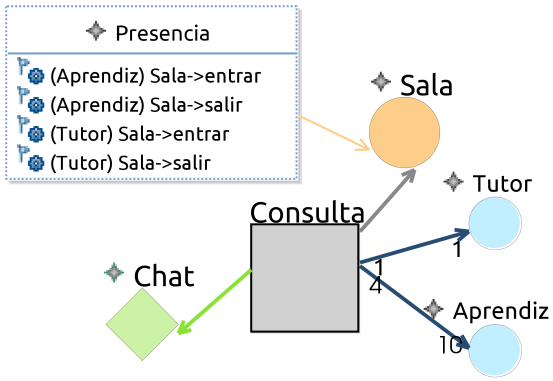


Figura 5.44: Awareness de presencia con eventos

De la misma manera se vincula el awareness a otros elementos colaborativos. Por ejemplo, se puede querer saber cuándo un tutor inició una Consulta. También se puede querer mostrar la participación de los usuarios en el chat. En el ejemplo que se muestra en la figura 5.45, se muestran los distintos awareness asociados a los elementos colaborativos.

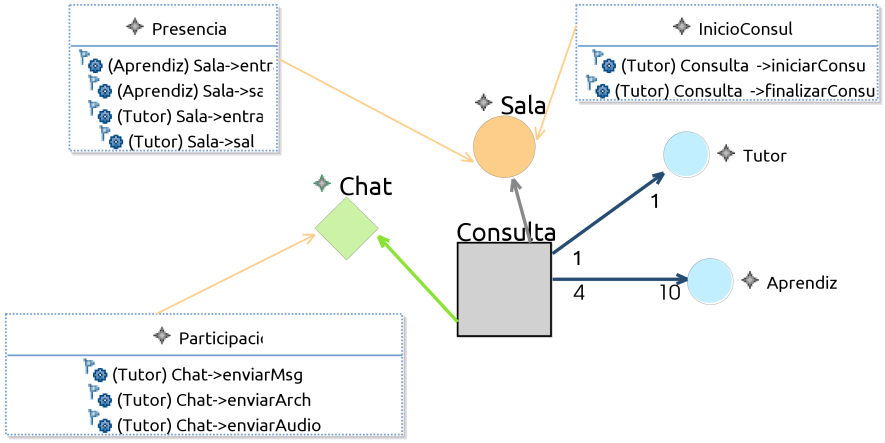


Figura 5.45: Awareness con eventos

5.3.2. Caso 2: Dinámica del Sistema

Más allá de la estructura del sistema colaborativo hay otros aspectos dinámicos que explican el comportamiento de los mismos. En la sección "Modelado de la Dinámica del Sistema" del capítulo anterior se explicó que el lenguaje CSSL tiene dos recursos para diseñar cuestiones dinámicas del sistema.

Por un lado, CSSL permite diseñar los procesos colaborativos que organizan en qué orden se realizan las actividades, y por otro, los protocolos que definen las operaciones que pueden realizan los roles dentro de cada actividad. En las próximas secciones se muestra un caso de modelado que ejemplifica los dos casos.

5.3.2.1. Modelado de Procesos Colaborativos

Para explicar cómo se especifica un proceso colaborativo, se toma el caso de una aplicación colaborativa para algún juego de mesa, en este caso el ajedrez. Los elementos colaborativos básicos que aparecen en un sistema para jugar al Ajedrez y algunos elementos de awareness, que se muestran en la figura 5.46.

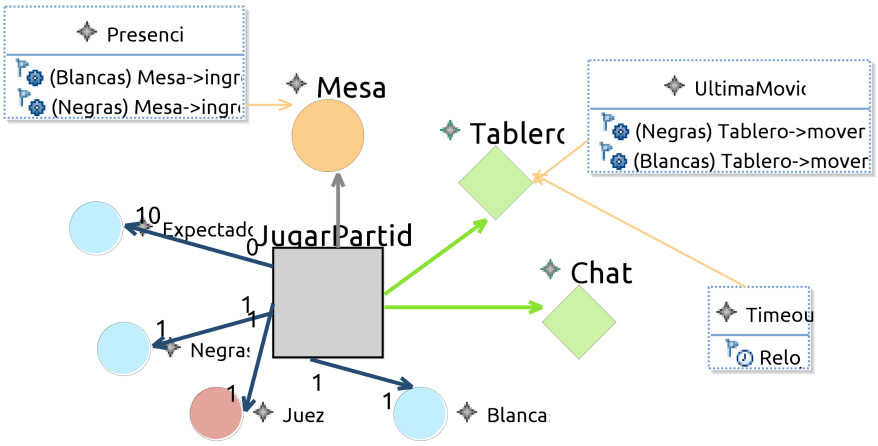


Figura 5.46: Modelo de Ajedrez con Awareness

En la figura 5.47 se presenta el diseño de un proceso colaborativo que describe un campeonato de Ajedrez. Es una versión simplificada de un campeonato compuesto de dos partidas semifinales y una partida final. Se muestra en la figura 5.47 las dos semifinales indicando que la actividad produce un dato de salida; el jugador que gana la partida. En la actividad final, se reciben

los ganadores de las semifinales y luego de la partida final, se definirá el jugador ganador del campeonato.

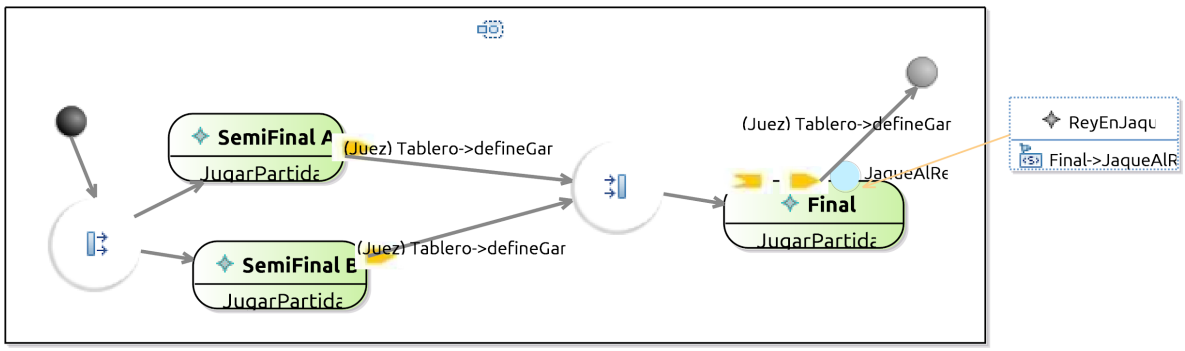


Figura 5.47: Proceso Colaborativo Campeonato

Este ejemplo de campeonato puede extenderse a versiones más complejas con zonas, o partidas todos contra todos o campeonatos de eliminación simple. Los procesos colaborativos tienen también relación con información de awareness, que surge durante su desarrollo. A las actividades, que forman parte de un proceso colaborativo, se le pueden adicionar eventos que pueden originar información de awareness,como es el caso del evento de "JaquealRey"de la figura 5.47 y que se mostrara en algún otro elemento colaborativo. Este awareness particular se activa cuando algún jugador provoca un jaque al rey del oponente. En la figura 5.46 se diseña que en el tablero se mostrará que el rey está en jaque. Luego en el capítulo de derivación de código se explicará cómo se traducen estos diseños a una implementación concreta.

5.3.2.2. Modelado de Protocolos

Otro aspecto dinámico que se diseña con CSSL son los protocolos, que definen el comportamiento dentro de una actividad colaborativa. El protocolo define los estados que tienen las actividades y qué operaciones se pueden los roles ejecutar en cada uno de ellos. En el ejemplo que se muestra en la figura 5.46, se tiene una única actividad: *JugarPartida*.

En la figura 5.48 se muestra una versión simplificada de un protocolo para jugar al ajedrez para la actividad "JugarPartida", que cuenta con dos estados: *MuevenBlancas* donde el jugador el jugador que tiene el rol Blancas puede mover una pieza y otro *MuevenNegras* donde puede mover el jugador que tiene el rol Negras. En el ejemplo, también se ve que los espectadores

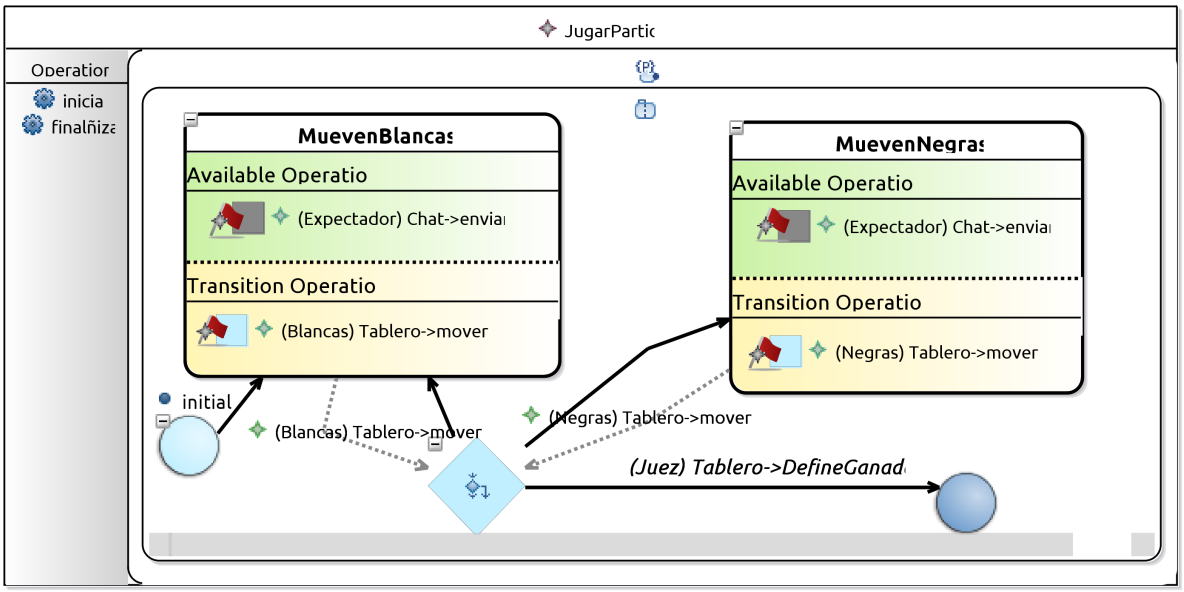


Figura 5.48: Protocolo de la Actividad JugarPartida

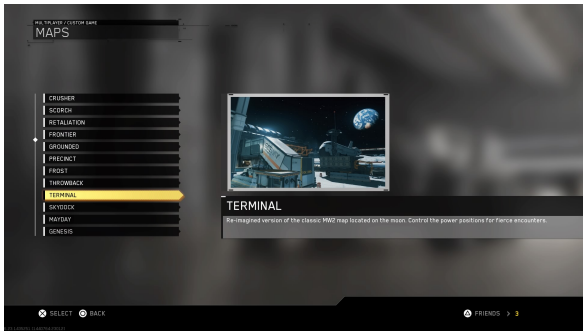
pueden enviar mensajes en el chat en los dos estados. El cambio de estado se produce cuando un jugador ejecuta el moverFicha. El protocolo especifica que luego de cada movida, se controle que no haya "jaque mate", lo que determinaría el final la partida y se definirá cuál de los jugadores es el ganador. En caso contrario se pasa al otro estado donde el que puede mover la ficha es el oponente.

En la figura 5.41 se ve cómo se seleccionan las operaciones disponibles dentro de cada estado. En el panel de la izquierda, está el listado de todas las operaciones que tienen asignados los roles que pertenecen a la actividad. En el panel de la derecha, aquellas que se asignan a un estado. Las operaciones que se listan son las que se definieron para cada rol como se explicaron en la figura 5.42

En la figura 5.48 también se muestra cómo se vincula el awareness con operaciones que se utilizan en los protocolos. En el ejemplo se muestra en el editor una bandera roja en las operaciones involucradas en los estados que originan la información de awareness que se mostrará en algún elemento colaborativo.

5.3.2.3. Modelado del Awareness

En esta sección, se describe los recursos que tiene el lenguaje CSSL para modelar el awareness. Para ello se eligió un videojuego multiusuario llamado “Call of Duty”, donde los jugadores se enfrentan a otros usuarios, de forma individual o en equipos. El juego permite que los usuarios recorran distintos escenarios (lugares) y cuando se encuentran con otros jugadores rivales pueden usar sus armas para “matarlos”. Hay varios juegos que responden a esta lógica donde los jugadores “luchan” o “batallan” para llegar a un objetivo. En algunos casos, el objetivo puede ser: capturar una bandera, tomar el control de un palacio o sumar puntos a partir de eliminar a los opositores. Estos juegos, se caracterizan por tener mucha información de awareness sobre lo que está pasando con los otros usuarios mientras transcurre la batalla.



(a) Mapas del Sistema

(b) Lugar en el Mapa

(c) Espacio dentro del Avión

Figura 5.49: Espacios del Sistema

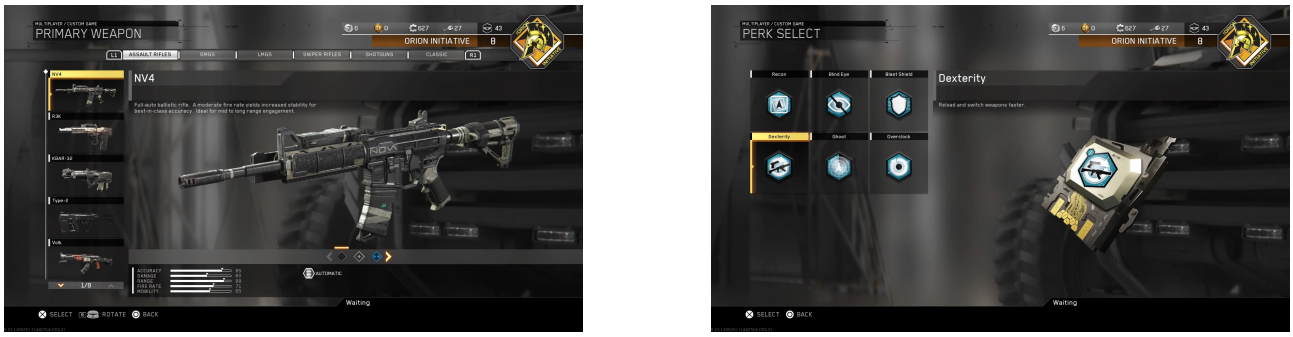
Antes de modelar el awareness, es necesario describir la estructura del sistema y algunos procesos característicos. Para analizar las características del juego, se realiza una actividad de ingeniería inversa, donde se diseña, usando el lenguaje CSSL, las características del juego a partir de lo que muestran las imágenes.

La primera estructura que se puede modelar son los espacios que se utilizan en el sistema. En

la figura 5.49b se observa distintos mapas donde el usuario puede jugar. Cuando el usuario, elige por ejemplo, el mapa “Terminal”, el usuario va poder navegar por distintos lugares dentro de la terminal. Por ejemplo, en la figura 5.49 se muestra la imagen del “Avión” que es uno de los lugares donde el usuario se puede mover dentro del aeropuerto, que también se muestra la figura 5.49c.

El lenguaje CSSL tiene una metaclase Workspace que se usa para diseñar los espacios. También tiene las posibilidades de modelar los lugares, como el aeropuerto, que tiene otros Workspaces dentro de él, con la relación de innerworkspace, como se presentó en la figura 4.5. Por otro lado, el usuario se mueve dentro de los espacios y puede cambiar de espacio navegando a espacios vinculados. Para esto, la metaclase Workspace mantiene estos espacios vinculados con la relación linked.

De acuerdo a la dinámica del juego, la actividad colaborativa que se desarrolla dentro de los espacios, la llamaremos “batalla”. En el lenguaje CSSL, existe una metaclase CollaborativeActivity para modelar las actividades colaborativas como esta.



(a) Herramientas (Armas)

(b) Herramientas (Poderes)

Figura 5.50: Herramientas del Sistema

En las figuras 5.50a y 5.50b se pueden ver las armas y poderes que se pueden elegir al empezar la partida. Éstos dos conceptos son las herramientas con las que cuenta el usuario para modificar el estado de la colaboración, ya que son los recursos con los que cuenta el usuario para modificar los objetos compartidos. Lo que en otros sistemas son editores de texto, o pizarras compartidas, en el Call for Duty son las armas y los poderes algunas de las herramientas colaborativas. Otra herramienta es un canal de audio que tienen los jugadores para comunicarse mientras juegan.

También existe el concepto de Rol, como se muestra en la figura anterior. Cada rol tiene algunas

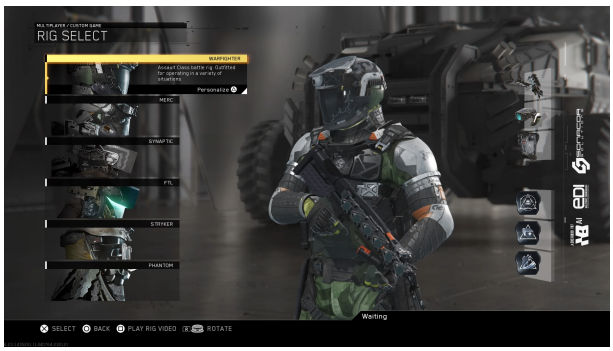
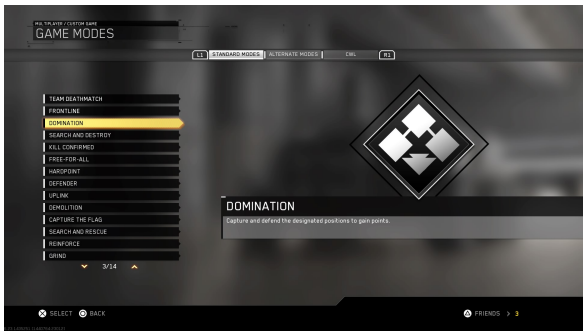


Figura 5.51: Roles del juego Call of Dutty

características propias que afectan el funcionamiento de las actividades. En CSSL, existe la metaclase CollaborationRole para modelar este tipo de roles. Los usuarios pueden elegir el rol, como se muestra en la figura 5.51 o se los asigna el sistema.

Por otro lado, el juego no consiste de una sola batalla, sino que para conseguir algún objetivo se deberán encadenar distintas batallas - en un concepto que llamaremos “Partida”. Por ejemplo, hay una modalidad de juego llamada “Infected”, que comienza con un usuario “infectado” y en distintas batallas, a medida va matando a otros usuarios los va infectando. La partida termina cuando están todos infectados y el ganador es el último “sano”. Los roles “sanos”, también pueden defenderse y matar a algún rol Infectado. En ese caso el “infectado”, revive en otra zona del mapa con el mismo rol donde seguirá persiguiendo a los sanos. En el lenguaje CSSL, existe una metaclase “CollaborativeProcess” que se usa para modelar un proceso que involucra distintas actividades colaborativas. En el caso de una “Partida Infected”, que está compuesto por distintas actividades, entre ellas “Batalla Infected”, y así modelar la Partida “Infected” como un proceso colaborativo.

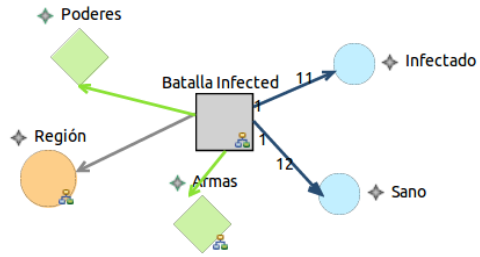


(a) Modalidades del Juego (Procesos)

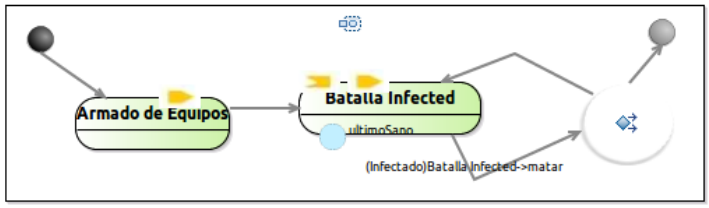
(b) Sincronización de la Actividad

Figura 5.52: Procesos del Sistema

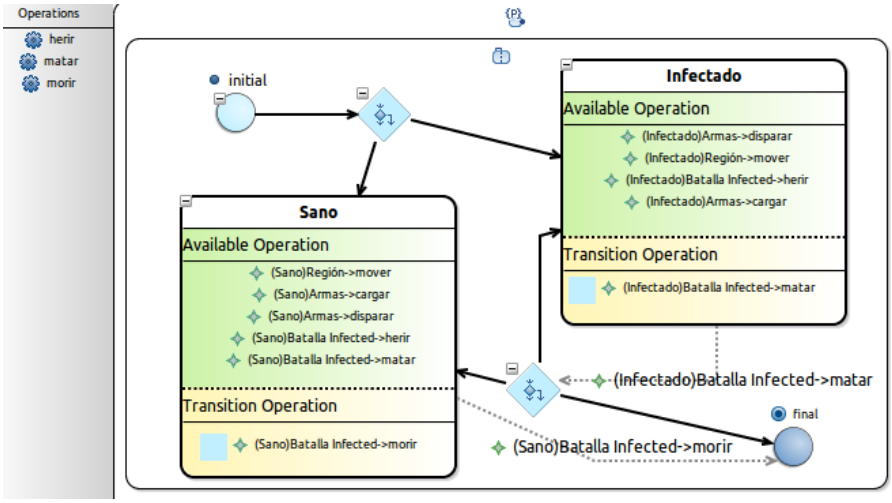
En las imágenes 5.52a y 5.52b se muestra, cómo se elige y se inicia un proceso llamado: “Partida Infected”.



(a) Estructura Simplificada de Infected



(b) Proceso Infected



(c) Protocolo Infected

Figura 5.53: Modelo Simplificado del Call of Duty (Modalidad Infected)

Ahora que los elementos colaborativos, procesos y protocolos fueron especificados - para el caso simplificado de un tipo de partida: “Partida Infected”, se puede describir la información de awareness que se muestra en el sistema. El modelado del awareness consiste en identificar qué tipos de información de awareness se mostrará. Analizando el juego identificamos los siguientes tipos a) Presencia, b) Amigos, c) Rol+Ubicación, d) Puntaje, e) Deceso (quién mató a quién), f) Infectados y sanos, g) Balas en el arma, h) Tiempo de partida.

En la figura 5.54 puede verse que los siguientes tipos de awareness. 1) Rol+Ubicación, 2) Deceso, 3) Infectados y sanos 4) Balas en el arma.

El awareness de **rol+ubicación** está asociado al espacio, ya que este tipo de awareness aparece en otros tipos de batallas que se desarrollan en el mismo espacio.

Por otro lado, tenemos información de awareness específicos de la batalla en la que el usuario

Figura 5.54: Awarenes de la batalla Infected

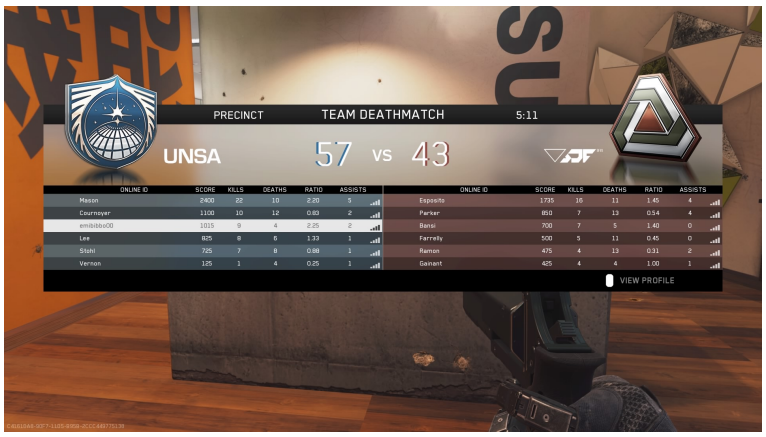
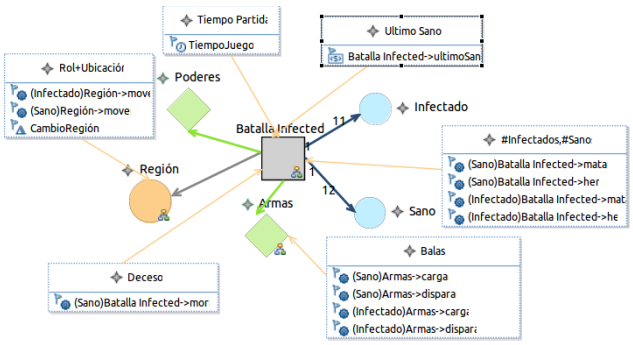


Figura 5.55: Resultados de la batalla Infected

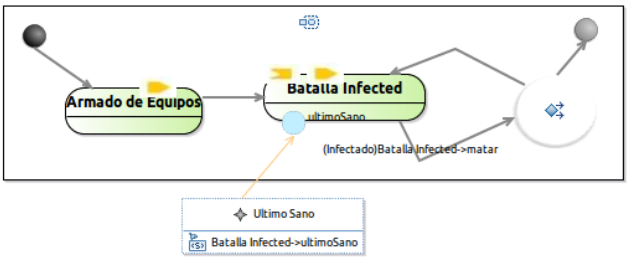
está participando. Uno es el awareness de deceso, que indica quién mató a quién y con qué arma, otro es el que indica cuantos son los infectados y cuantos los sanos hay en esta batalla, el último sano, que se destaca cuando queda solo un usuario sano en la partida y el tiempo que lleva esta partida. Finalmente, tenemos una información de awareness relacionada al arma que indica cuántas balas quedan disponibles en el arma.

En la figura 5.55, se puede ver el resultado que refleja el awareness de Puntaje que se fue acumulando durante la Partida. Según la especificación el puntaje se iba actualizando a medida que los usuarios mataban a los contrincantes. Eso se refleja en los valores que aparecen en la tabla de resultado.

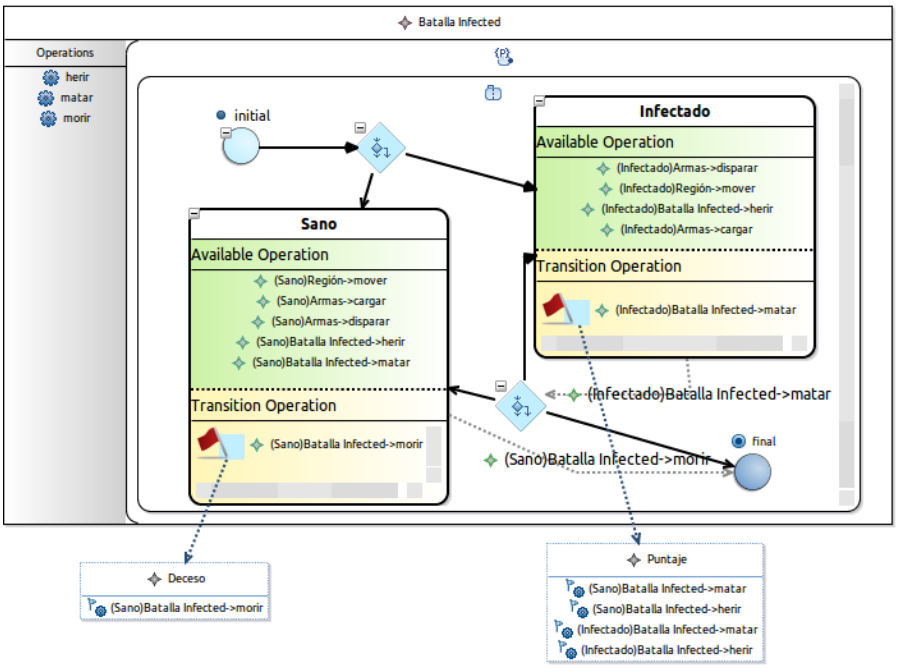
Para modelar estos awareness se especifica, donde se muestran y a partir de qué eventos se



(a) Estructura Simplificada de Infected con Awareness



(b) Proceso Infected con Awareness



(c) Protocolo Infected con Awareness

Figura 5.56: Awareness en el Modelo Simplificado del Call of Duty (Modalidad Infected)

actualizan, como se mueStra en las figuras 5.56. Como se puede ver en las figuras 5.56, la especificación de los awareness involucró tanto el diagrama de elementos colaborativos que modela la estructura del sistema, como los procesos colaborativos, como los protocolos. Para cada uno de los awareness, se especificó un conjunto de eventos que lo mantienen actualizado. Por ejemplo, cuando el usuario dispara el arma, se actualiza la cantidad de balas disponibles.

5.3.3. Caso 3: Modelado del DimSum Thinklet

Los ThinkLets, presentados en el trabajo de Robert et al. [54], son patrones de diseño para modelar dinámicas de trabajo colaborativo. Los facilitadores e ingenieros de colaboración utilizan ThinkLets para diseñar procesos de colaboración para tareas recurrentes y transferir esos diseños a los profesionales para que los ejecuten por sí mismos sin la intervención continua de facilitadores profesionales. La colección de thinkLets forma un lenguaje de patrones para crear, documentar, comunicar y aprender diseños de procesos de grupos. En [54] se presenta como ejemplo un thinkLet completamente documentado, llamado DimSum. En este thinkLet, el objetivo del equipo es crear una declaración específica, precisa, sobre un tema, de una manera que todos estén de acuerdo, la comprendan y que se adapte a los intereses de todos los miembros del equipo. Cada miembro propone una terminología candidata para la declaración común. Luego, los participantes obtienen las palabras y frases que más les gustan de las declaraciones de los candidatos para crear una nueva declaración común. Periódicamente, todos los participantes proponen nuevas declaraciones candidatos basadas en el borrador actual de la declaración común. El ciclo continúa hasta que surge una versión que todos los participantes aceptan.

Aquí, el patrón de colaboración es un ciclo de generación de conceptos, aclaración de significado y construcción de consenso sobre la declaración final. Este patrón de colaboración se puede modelar de forma fácil y precisa aplicando CSSL.

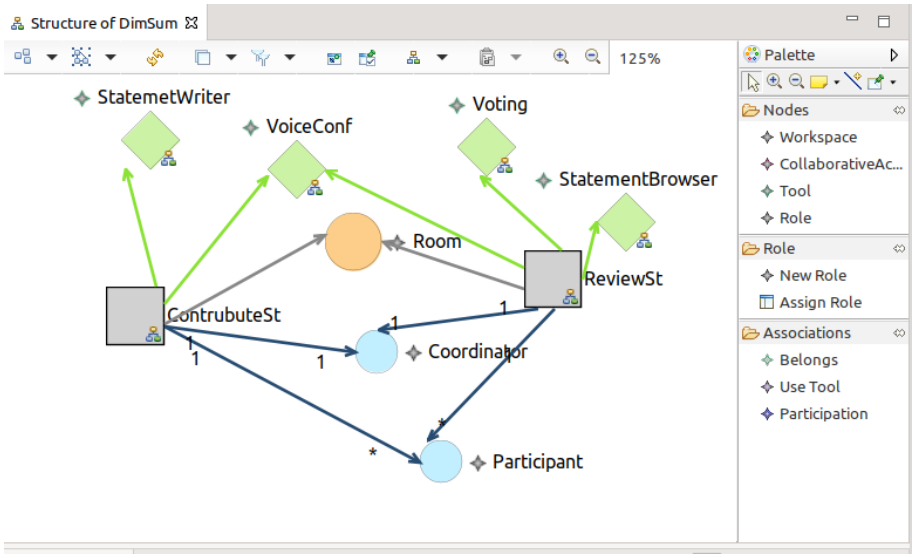


Figura 5.57: Entorno del DimSum ThinkLet

Antes de comenzar con el diseño de la interacción entre los participantes, se debe describir el entorno donde los usuarios colaboran. Las herramientas, los roles y el lugar virtual donde los usuarios se reunirán para desarrollar las actividades colaborativas involucradas en el DimSum thinkLet, que se identifican en un primer paso y que se muestra el entorno completo en la Figura 5.57. El modelo especifica dos actividades colaborativas: una para la generación de conceptos (ContributeSt) y otra para la clarificación del significado y la construcción de consensos (ReviewSt). Ambas actividades colaborativas se realizan en un mismo lugar virtual denominado Room. Coordinador y participante son los dos roles involucrados en la tarea. Finalmente, la especificación describe el conjunto de herramientas colaborativas que los usuarios / roles emplean para llevar a cabo la tarea. El juego de herramientas incluye un navegador de sentencias/declaraciones, un editor de sentencias/declaraciones, un sistema de conferencias de voz y una herramienta de votación.

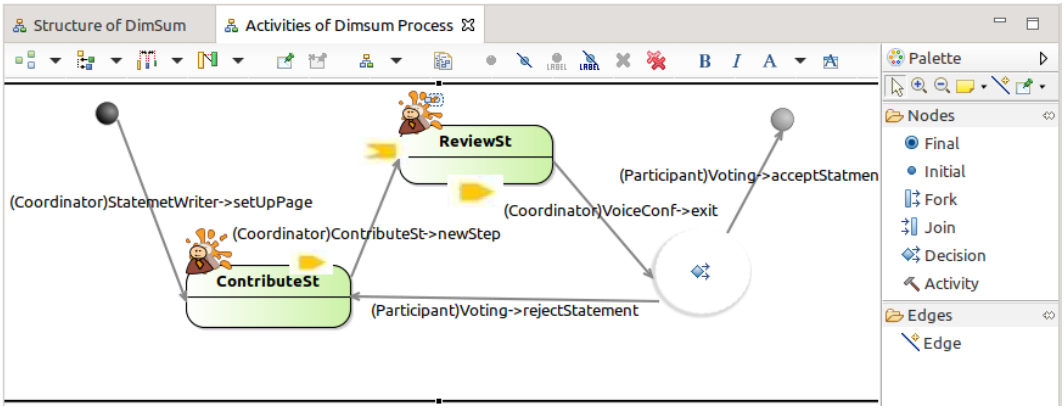


Figura 5.58: Modelo del proceso del DimSum ThinkLet

El modelo de proceso colaborativo que involucra el Dimsum Thinklet se ilustra en la Figura 5.58. El modelo especifica las actividades ContributeSt y ReviewSt y las transiciones que definen el proceso colaborativo. Las actividades están etiquetadas con iconos del trabajo de [17] que muestran el tipo de actividad y si producen información o no.

Luego, cada una de las actividades colaborativas se modela en CSSL. Para ello, se especifican los protocolos. La Figura 5.59, muestra las interacciones entre los participantes en la actividad colaborativa ContributeSt. El modelo cumple con las especificaciones DimSum Thinklet donde se describe que la actividad pasa por diferentes estados. En cada uno de ellos, los usuarios pueden realizar diferentes operaciones. El modelo muestra un primer estado de la actividad llamado Configuración donde el coordinador explica el objetivo de la actividad. Luego, en el siguien-

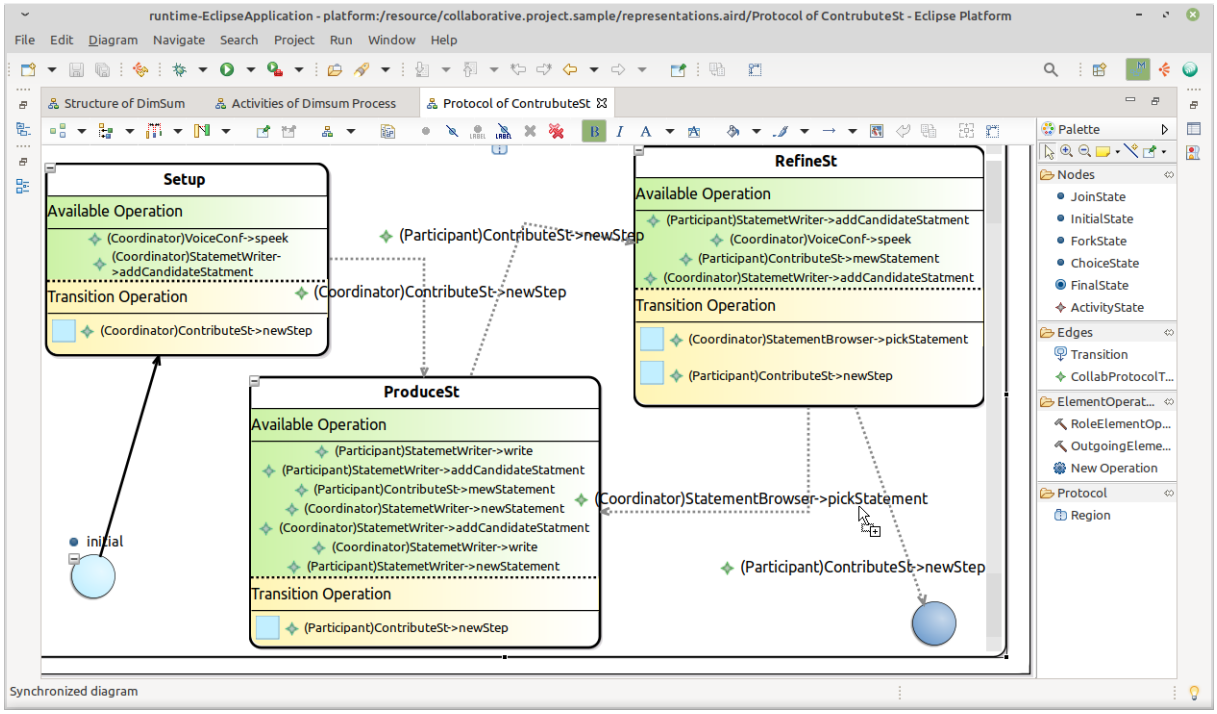


Figura 5.59: Modelo del protocolo de la actividad ContributeSt

te estado, llamado ProduceSt, se generan y recolectan las contribuciones de los participantes y finalmente en el último estado, llamado RefineSt, se ajustan las oraciones candidatas. La actividad continúa cíclicamente hasta que se alcanza un consenso entre todos los participantes.

5.3.3.1. Modelo de Awareness para el Dimsum Thinklet

Los modelos anteriores describen la estructura y dinámica de DimSum thinkLet. Pero otro elemento importante en un diseño de sistema colaborativo es la especificación de la awareness. Y de hecho, el modelado del awareness es una de las fortalezas más importantes de CSSL, ya que permite al ingeniero de colaboración o diseñador especificar dónde deben mostrarse los componentes de la awareness y qué eventos desencadenan la actualización de cada uno de ellos.

Aunque la especificación de awareness enriquecería notablemente el alcance, DimSum thinkLet no define los tipos de awareness a aplicar. Por lo tanto, los siguientes cinco tipos de awareness se incorporan al modelo thinkLet: Presence, Speaking, Statements, VotingResult, TimeTo. La Figura 5.60 muestra las instancias de los tipos de awareness con los detalles de los eventos que los actualizan. Es de destacar que la mayor parte de los awareness se definió como sincrónica y no transitoria (no almacenada en el sistema). Excepto en el caso del awareness Statements

que deben almacenarse en el sistema para su posterior recuperación. La figura 5.60, también muestra que los componentes de awareness están vinculados a diferentes tipos de elementos (Workspace Room, VoiceConf Tool, Vooting Tool, ContributeSt Activity).

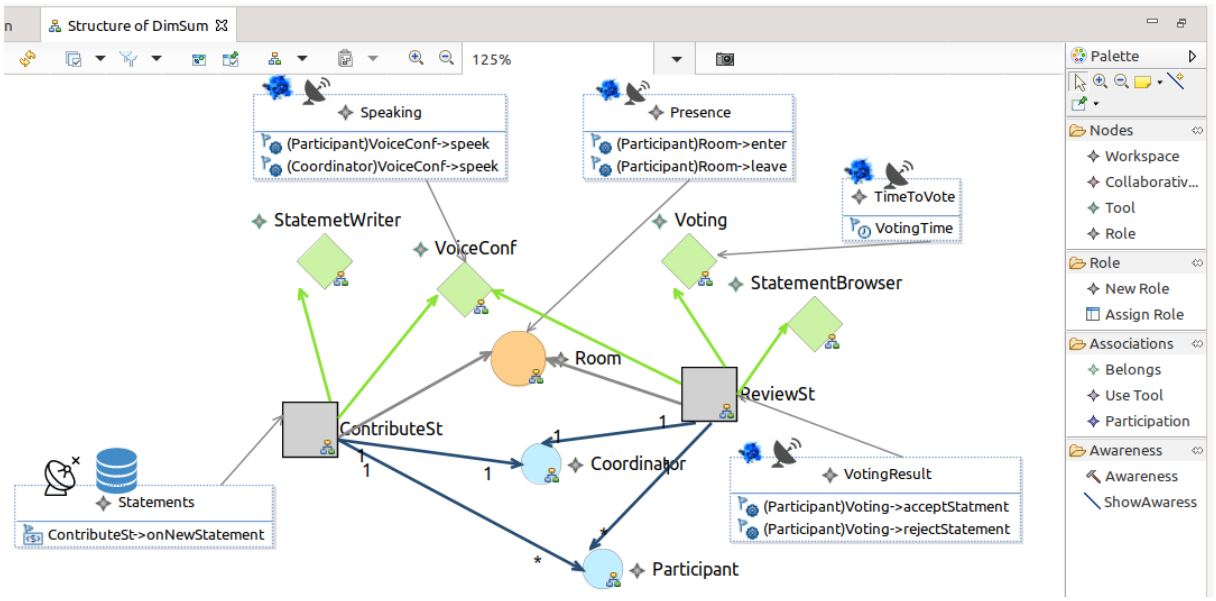


Figura 5.60: Modelo del entorno del DimSum ThinkLet con Awareness

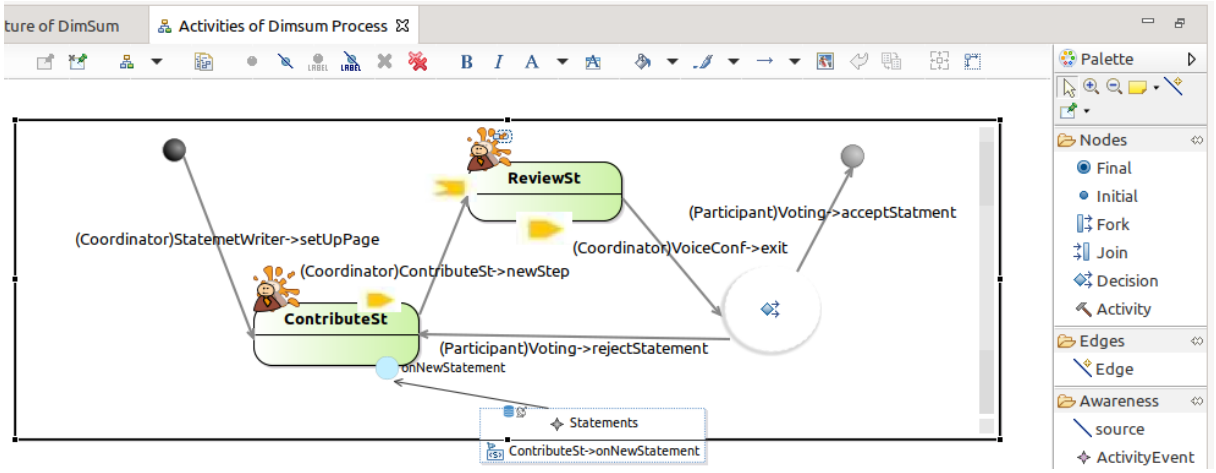


Figura 5.61: Modelo del proceso de DimSum ThinkLet con awareness

La mayoría de las actualizaciones de los awareness se basan en las acciones que realizan los usuarios. El sistema está atento a las operaciones que realizan los roles en el sistema y activa la actualización de awareness cuando corresponde. Por otro lado, otros tipos de awareness están vinculados a eventos no originados por los usuarios. Por ejemplo, el tiempo transcurrido, como se especifica en el Awareness TimeToVote, donde hay un evento de tiempo UML activado para

controlar la cantidad de tiempo permitido para votar. El awareness también está vinculado a eventos dentro de los procesos y se puede especificar como se muestra en la Figura 5.61.

5.4. Conclusiones del capítulo

Utilizando el proyecto de Eclipse *Sirius*, se construyó un conjunto de editores gráficos para trabajar con el lenguaje CSSL. El objetivo fue brindar a los usuarios del lenguaje, diseñadores de Sistemas Colaborativos, mayor flexibilidad y legibilidad. Cada editor es una vista de parte de un modelo de Sistema colaborativo donde se expresa distintos aspectos del Sistema Colaborativo que se está modelando. Esto permite que los diseñadores trabajen con una sintaxis concreta, gráfica en este caso, para describir el sistema.

Sirius permite definir representaciones (views) para las clases de un metamodelo. Para estas metaclases se eligieron distintas formas y colores (cuadrados, elipse, etc.) y mecanismos de agruparlos y relacionarlos. Estas representaciones tienen una paleta de edición que permite crear instancias de las metaclases del lenguaje CSSL y relaciones entre ellas. Así es que se van construyendo los modelos de sistemas colaborativos a partir de las metaclases del lenguaje CSSL.

Se crearon editores para describir la estructura del sistema, donde se crean los elementos principales del sistema y cómo se asocian usando las relaciones de pertenencia, uso de herramientas y participación en actividades. También permite vincular el awareness a estos elementos.

Se presentaron tres casos de estudio, para comprobar la capacidad expresiva del lenguaje y los editores creados. Primero se crea la estructura del sistema colaborativo, describiendo los elementos que lo componen y las relaciones entre ellos. El diagrama se completa agregando la información de awareness asociada a este modelo. En el segundo caso, se hace foco en aspectos dinámicos de los sistemas colaborativos y especialmente en el modelado de awareness en juegos. En el tercer caso se presenta el modelado del DimSum ThinkLet de los autores (Robert O. Briggs, Gert-Jan de Vreede, Gwendolyn L. Kolfschoten - 2007).

Capítulo 6

Derivación de Código

En esta sección, se analiza cómo utilizar los modelos construidos con el lenguaje CSSL, para automatizar el proceso de implementación. Esto significa tomar modelos independientes de la plataforma (PIM – Platform Independent Model), que luego son transformados en un modelo específico para la plataforma particular (PSM – Platform Specific Model).

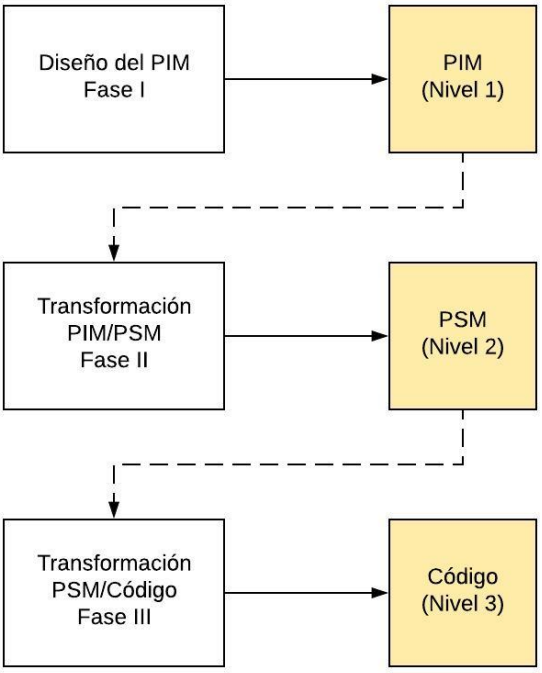


Figura 6.1: Pasos y Niveles en MDA

Partiendo de un modelo PIM, que en este caso es una instancia del metamodelo CSSL, y

mediante el uso de transformaciones, se va recorriendo los distintos niveles del desarrollo MDA, como se muestra en la figura 6.1.

Por otro lado, partiendo de un PIM se puede obtener uno o más PSMs, dependiendo de la cantidad de plataformas tecnológicas que se desee abarcar. Por ejemplo, a partir de un modelo PIM se podría se podría obtener un modelo de Objetos y un modelo de base de datos Relacional. Esto logra una gran independencia entre la definición de la funcionalidad y la plataforma que lo va a implementar, como se trató en de Fuentes, L., y Vallecillo, A [53].

En el contexto de MDA, una instancia de CSSL es un PIM (Platform Independent Model), ya que no hace referencia a los sistemas operativos, los lenguajes de programación, el hardware, la topología de red, etc. Para analizar una transformación a un modelo ejecutable se elige la arquitectura de transformación que define para cada elemento del Metamodelo CSSL, un elemento en el metamodelo destino, como se muestra en la figura 6.2.

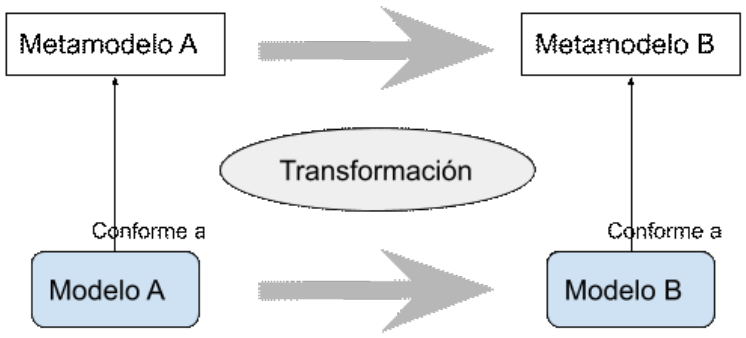


Figura 6.2: Arquitectura de Transformación

En la tabla 6.1, se propone una transformación para cada una de las metaclases del lenguaje CSSL. Esta asociación, luego se utiliza para desarrollar una implementación en una plataforma en particular.

Metaclases	**Transformación**
CollaborativeModel	Aplicación: Para cada modelo se crea un sistema colaborativo o una aplicación.
CollaborativeElement	Componente Abstracto: Los elementos colaborativos serán componentes visibles en el sistema. Estos elementos tienen operaciones que aparecen como botones en la interfaz del sistema.

CollaborativeElement: Workspace	Componente Contenedor: Componente visible en el sistema que contendrá las actividades que le pertenecen y las operaciones definidas para él.
CollaborativeElement: CollaborativeActivity	Componente Contenedor: Componente visible en el sistema que contendrá las herramientas que se usan y las operaciones definidas para la actividad.
CollaborativeElement: Tool	Componente Contenedor: Componente visible en el sistema que contendrá las operaciones definidas para la herramienta.
CollaborativeElement: SharedObject	Componente Contenedor: Componente visible en el sistema que contendrá las operaciones definidas para él.
ElementOperation	Botón: Aparecen en los elementos colaborativos y representan las operaciones relacionadas que se pueden ejecutar en cada uno de ellos
CollaborativeAssociation	Mantiene asociaciones entre los elementos colaborativos
CollaborativeAssociation: BelongRelationship	Esta asociación indica que una CollaborativeActivity pertenece a un Workspace. Usando esta relación se agrega la componente de la actividad a la del espacio.
CollaborativeAssociation: UseRelationship	Esta asociación indica que una Tool se usa en una CollaborativeActivity. Usando esta relación se agrega la componente de la herramienta a la de la actividad.
CollaborativeAssociation: ParticipationRelationship	Esta asociación indica que un CollaborationRole participa en una CollaborativeActivity. Usando esta relación se definen cuáles y cuántos roles participan en cada actividad.
User	Usuario registrado en el sistema.
CollaborationRole	Rol dentro del sistema. Con este elemento se controlan los permisos de los usuarios.
RoleElementOperation	Habilita a un rol a ejecutar una operación de un elemento colaborativo. Hace que el botón sea visible para un rol determinado. Por otro lado activa un Event que indica que la operación fue ejecutada. Esto es usado para mostrar información de awareness en algún lugar del sistema.

CollaborativeProcess	Componente contenedor compuesto por instancias de CollaborativesNodes, nodos de control (initial, fork, join, decision, final) y ejes que unen estos nodos. Representa un proceso del sistema.
CollaborativeActivityNode	Representa a una instancia de una actividad dentro de un proceso.
CollaborativeProcessEdge	Eje de un proceso que se activa cuando un rol ejecuta una operación (RoleElementOperation).
CollaborativeActivityState	Representa un estado dentro de una actividad colaborativa. Está definida como un conjunto de RoleElementOperation que tiene asignado.
CollaborativeProtocol-Transition	Es una cambio de estado dentro de una actividad y que es originado en una operación (RoleElementOperation) que realiza algún en el estado origen.
AwarenessKind	Tipos de información de awareness. Define las características de cada uno de ellos.
Awareness	Componente visible en el sistema que agrega a la/s componente/s definido usando la relación shownIn. Conjunto de eventos que actualizarán la información de la componente, que se encuentran en la relación source.

Tabla 6.1: Mapeo de las Metaclases del lenguaje CSSL

A partir del mapeo elaborado se eligió una arquitectura destino que permita implementar los conceptos mencionados en la transformación (Aplicación, Componente, Boton, Proceso, Clases (Tipos), Permisos, Role, Usuario, Componente Contenedor).

6.1. Tecnología Destino

En este apartado se presenta una transformación desarrollada con una herramienta de eclipse llamada Acceleo [55] que toma conceptos del lenguaje CSSL y obtiene un modelo específico para una plataforma particular, un PSM (Platform Specific Model). El resultado de la transformación, es un conjunto de archivos con código en algún lenguaje de programación y se eligió una arquitectura destino, en este caso una arquitectura Cliente-Servidor, en particular una Web.

Por lo tanto, para cada elemento del lenguaje se define qué componentes del sistema se tiene que construir; tanto en el cliente como en el server. Por otro lado, se optó por un lenguaje liviano y que no requiere una configuración complicada (como es el caso de JavaScript y tecnologías asociadas a dicho lenguaje, como Node.js y frameworks del lado del cliente como Angular o React.). También se definió que el intercambio de datos entre el cliente y el servidor se realizará a través de una API REST, que funciona como una interfaz entre sistemas que utilicen el protocolo HTTP para obtener los datos o advertir la ejecución de alguna operación, usando un formato XML o JSON.

La elección final recayó en TypeScript, que es un lenguaje de programación basado en JavaScript, con la ventaja que es un lenguaje tipado, que permite crear estructuras de clases y puede ejecutarse tanto en el cliente como en el servidor.

En definitiva, la transformación consiste en tomar una instancia del lenguaje CSSL y producir como resultado una aplicación Web implementada en TypeScript, donde el desarrollador luego podrá agregar código propio para adaptar el resultado final a sus intereses.

Finalmente para implementar el awareness se eligió la tecnología de **WebSocket**, que permite establecer una conexión entre el navegador del usuario y el servidor. Esta conexión es bidireccional y dura el tiempo que se mantenga abierto el navegador del cliente. Cuando hablamos de bidireccional nos referimos a que ambos, tanto el cliente como el servidor, pueden enviar y recibir mensajes por el canal o conexión establecida. Los WebSockets innovaron la web y provocaron que sea aún más dinámica, por esta razón se estandarizó y se generó un protocolo propio llamado **WS**, que al igual que **HTTPS**, también tiene su versión segura, **WSS**.

6.2. Transformación del PIM al PSM

La estrategia de transformación (PIM a PSM) elegida, consiste en tomar a cada uno de los conceptos del modelo PIM para construir componentes en Angular, que conforman un PSM específico. Para esto, se recorre el árbol de un modelo colaborativo, por ejemplo, el de la figura 6.3, y se trabaja con cada uno de los elementos que dependen del collaborative model.

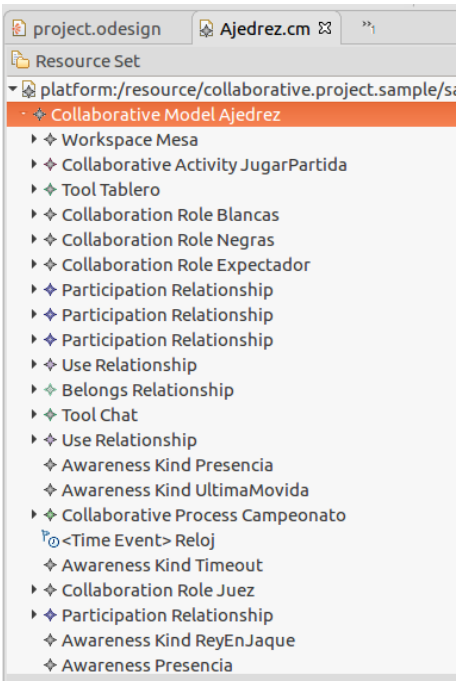


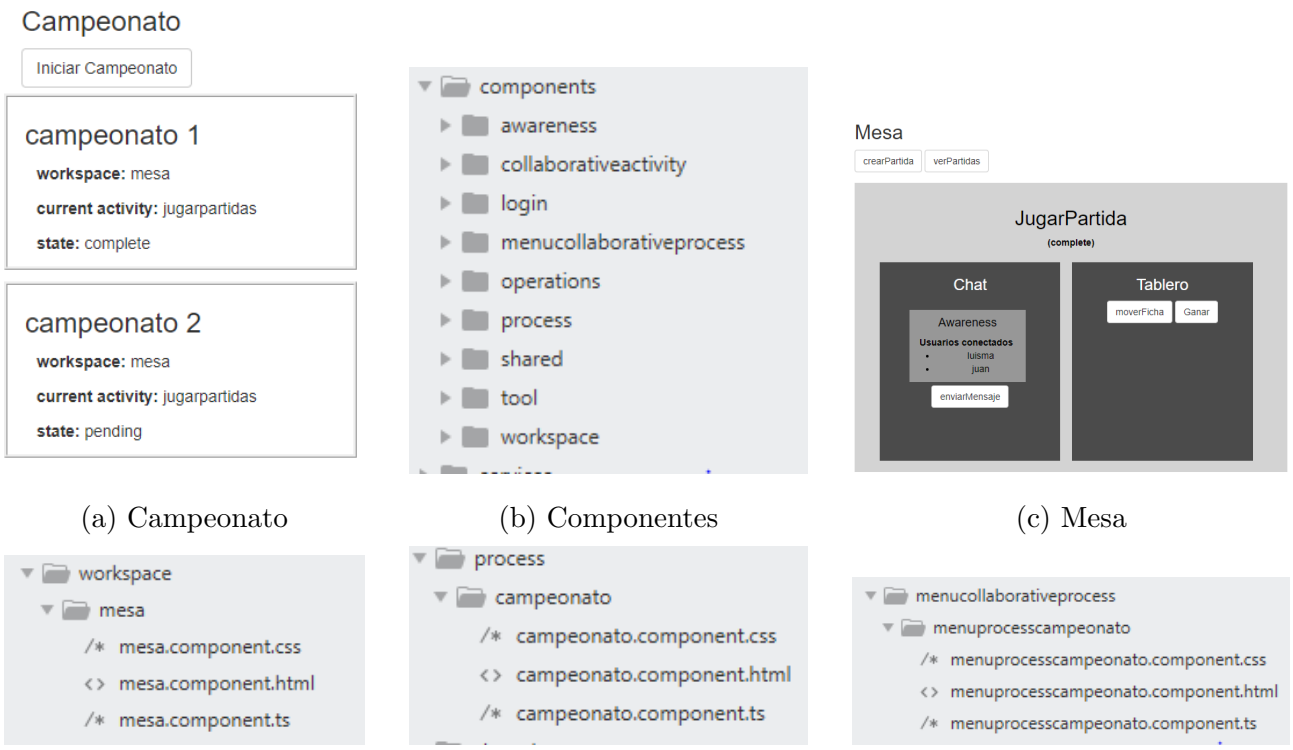
Figura 6.3: Árbol del Modelo de Ajedrez

Se transforma entonces los elementos colaborativos que aparecen en el modelo de entrada, en componentes de Angular (awareness, collaborativeactivity, operations, process, shared object, tool y workspace) y otras dos componentes específicas que manejan el login y el menú, como se muestra en la figura 6.4b.

Se describe en la próxima sección cómo se definió la transformación de los elementos principales del lenguaje.

6.2.1. Transformación de los Procesos Colaborativos

Los procesos se transforman en componentes que se mostrarán en el sistema indicando el orden en que las actividades colaborativas que realizan los usuarios se llevan a cabo. Cada proceso



(a) Campeonato (b) Componentes (c) Mesa

(d) Workspace (e) Process (f) MenuProcess

Figura 6.4: Estructura de la plataforma de implementación

va a ser un componente Angular y se identificará por su nombre, teniendo su propio archivo TypeScript, su template .html, y su archivo de estilo como se ve en la figura 6.4e para el proceso campeonato. Para cada proceso también se crea otra componente Angular (fig. 6.4f), que se utiliza para mostrar las opciones en el menú para todos los procesos al que el usuario tenga acceso.

En el ejemplo, al abrir el proceso “campeonato”, se muestra una página donde se listan todos los campeonatos donde el usuario tiene participación (figura 6.4a). Para cada instancia, se puede ver el nombre, el workspace, la actividad actual y el estado. Si el usuario hace click en una de estas instancias, automáticamente el sistema lo redirige a la pantalla de la actividad actual de ese proceso (campeonato).

El mecanismo de transformación de modelos, implica tomar algunas decisiones: por ejemplo, elegir el momento en que se crean las instancias de los elementos que fueron modelados. Para nuestro ejemplo, se decidió que el usuario puede crear instancias de un proceso campeonato en cualquier momento, a través de un botón como se muestra en la figura 6.4a. A medida que distintos usuarios van creando campeonatos, se van agrupando para organizar los miembros de

las partidas de los campeonatos.

6.2.2. Transformación del Workspace

Dentro del modelo, el workspace es el lugar donde ocurren las actividades. El elemento workspace también será un componente Angular que funciona como contenedor de otros componentes: Tools y CollaborativeActivity. La implementación de la transformación se muestra en la figura 6.4c, los botones muestran las operaciones de las actividades, las herramientas y los espacios que los roles pueden ejecutar en ese espacio.

6.2.3. Transformación de la Actividad Colaborativa

Las actividades son las tareas colaborativas que realizan los usuarios dentro de los procesos y están contenidas en un workspace. Las actividades pasan por distintos estados, que al crearse se iniciarán en estado *pendiente* y significa que faltan usuarios para iniciar, y cuando están todos los usuarios la actividad podrá arrancar como fue diseñado. En la transformación que se muestra en la figura 6.5, puede verse que la actividad JugarPartida aparece en el espacio Mesa porque hay una asociación que los vincula (BelongRelationship). La implementación de la transformación crea una componente Angular para las actividades y las agrega a la componente del espacio donde se desarrolla esa actividad.

6.2.4. Transformación de las herramientas (Tool)

Las herramientas se usan en las actividades colaborativas y por ser elementos colaborativos tienen operaciones tal como se muestran en la figura 6.5. En el ejemplo de la figura se muestra que la actividad JugarPartida usa 2 herramientas, el Tablero y el Chat, como se especificó con la asociación UseRelationship. Por ser elementos colaborativos tienen operaciones que aparecen como botones en la figura. Las herramientas se transforman en componentes y se agregan a la componente de la actividad donde se usa la herramienta.

En la figura 6.5 se muestra el vínculo que hay entre las clases del metamodelo y las componentes que aparecen en el sistema.

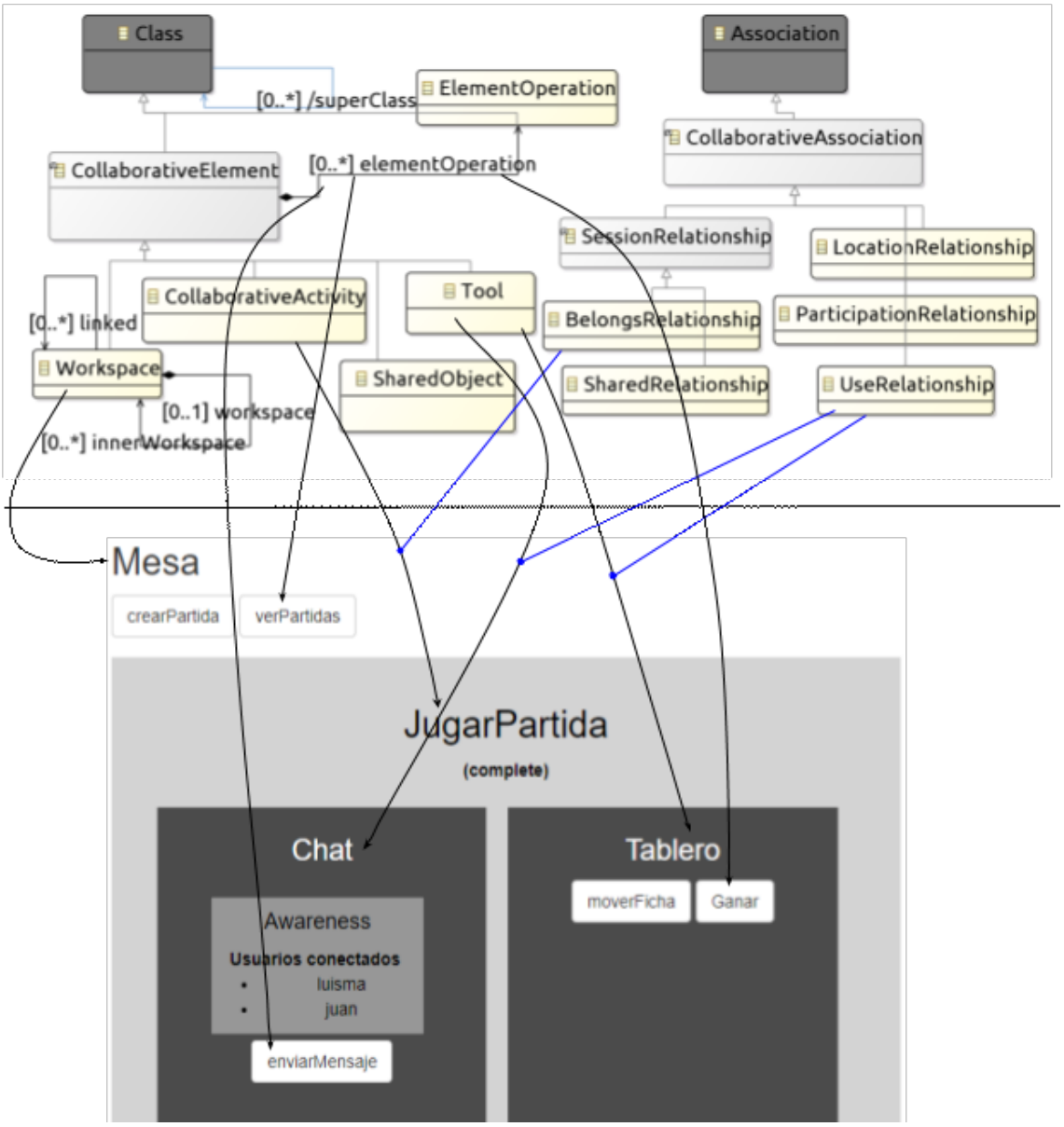


Figura 6.5: Mapeo entre el Modelo y la Aplicación

6.2.5. Transformación del Awareness

Una vez que se modela los elementos colaborativos, se define como se agrega el awareness a dichos elementos. Cómo se muestra en la figura 6.6, los elementos colaborativos pueden mostrar información de awareness, que es una componente que se agrega a la componente del elemento colaborativo.

En el ejemplo se muestra el tipo de Awareness "Usuarios conectados" se muestra en la herramienta (Tool del metamodelo) **"Chat"** y que mostrará los usuarios que están actualmente en línea en esa herramienta.

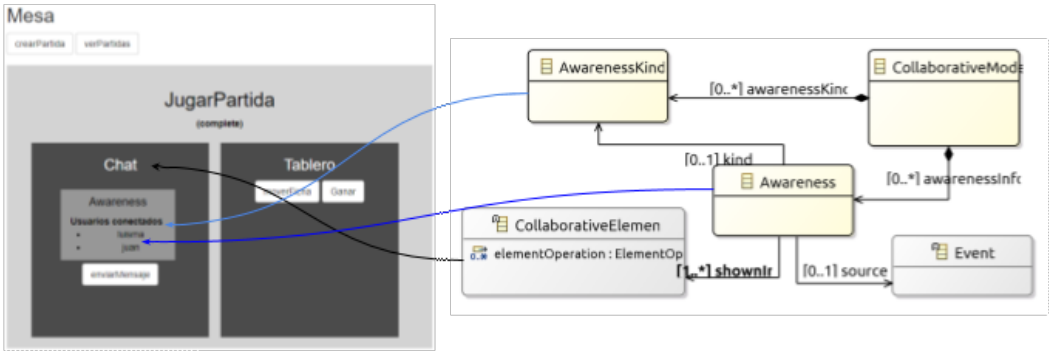


Figura 6.6: Mapeo del Awareness con la Aplicación

Es sabido que va el listado de usuarios se actualizará cuando algún usuario se conecte o se desconecte. Por lo tanto, estas acciones que el usuario realiza disparan eventos que automáticamente actualizarán la información de awareness.

Teniendo en cuenta las necesidades del awareness se implementó una solución que consiste: de una componente Angular que se agrega a las componentes de los elementos colaborativos donde se muestra ese awareness y un vínculo utilizando la tecnología de Websocket para mantener actualizado la información que se muestra. Al mismo tiempo se contará con un servidor de Websockets donde se registrarán todo los eventos relacionados con el Awareness del sistema.

6.3. Conclusiones del capítulo

La transformación de modelo a texto implementada brinda una semántica al lenguaje CSSL. Es decir que interpreta a los modelos construidos con el lenguaje y los transforma en objetos concretos. En este capítulo se presentó una transformación que tiene los siguientes beneficios para los desarrolladores de Sistemas Colaborativos.

- Estructura de desarrollo: A partir de la transformación se obtiene una aplicación web con una estructura de directorios, que permite organizar y reconocer de forma ágil cada uno de los componentes generados asociados con su correspondiente elemento del modelo. Se obtiene entonces una estructura para componentes para el servidor, componentes para el cliente, la estructura de la base de datos y componentes relacionados con el servicio de websocket que implementa el awareness.

- Gestión del usuario: El sistema web resultante dispone de funcionalidades relacionadas con el usuario. Por lo tanto, el modelador no debe preocuparse por el manejo de la sesión ni por la autenticación de los usuarios. El sistema ya tiene incorporado el registro del usuario y el inicio/cierre de sesión del mismo. Además, aporta un nivel de seguridad en las rutas generadas, ya que comprueba, mediante el uso del token, la sesión del usuario en cada acceso.

- Manejo de roles del sistema: Los roles que se crean en el metamodelo son traducidos al sistema web y permiten al modelador gestionar diferentes perfiles, determinando qué elementos serán visibles y a cuáles tendrán acceso los distintos usuarios. También brinda la posibilidad de configurar la cantidad de usuarios que participan en un rol y cuáles son más concluyentes.

- Creación de los elementos colaborativos y operaciones: El sistema resultante se compone por distintos componentes que representan a los elementos colaborativos modelados. Cada uno de ellos con sus operaciones que se transforman en botones en el sistema. También el sistema controla qué operaciones puede ejecutar cada uno de los roles dentro del sistema.

- Procesos y actividades colaborativas: Estos elementos permiten modelar la dinámica del sistema. En la transformación se obtiene un sistema que permite que los usuarios se enrolen a los procesos y permite que los usuarios participen de las actividades. También el sistema mantiene el estado de cada actividad dentro de cada proceso, esto significa que el sistema controla los usuarios que pueden entrar en cada actividad.

- Protocolos de colaboración: Las actividades colaborativas pasan por distintos estados. Con la transformación ejecutada se obtiene una implementación de una máquina de estados que se mantiene durante la ejecución del sistema. El sistema controla también los eventos (generalmente operaciones que ejecutan los roles dentro de las actividades) que se producen en el sistema y va actualizando los estados de las actividades.

- El Awareness: El sistema crea una infraestructura para implementar la funcionalidad de awareness. A partir de los modelos, se crean distintos componentes para mostrar la información de awareness que se agregan a los elementos colaborativos. También se crea un servicio de websockets que se conecta con esos componentes y envían mensajes de actualización a partir de los eventos que se producen en el sistema.

Capítulo 7

Evaluación, Validación y Verificación del Metamodelo

7.1. Comparación con trabajos relacionados

CSSL es un lenguaje específico de dominio construido como una extensión UML utilizando el mecanismo de metamodelado. Se introdujo una primera versión del lenguaje en [13]. La versión actual cumple con todos los requisitos indicados en la sección de requisitos 3.2.2 del capítulo 3.

A diferencia del trabajo de Gallardo [39], con CSSL, el contenido de los editores no se puede modelar específicamente, pero CSSL permite integrar cualquier tipo de herramienta colaborativa en el diseño, describiendo qué operaciones pueden ejecutar los roles en cada momento. En cuanto al espacio de trabajo, CSSL permite a los ingenieros diseñar relaciones entre espacios de trabajo como la inclusión entre espacios. Esta facilidad permite cubrir varios dominios colaborativos (e-learning, lluvia de ideas, juegos colaborativos, etc.). También permite integrar un diseño específico de la información de awareness asociada a los diferentes elementos del modelo.

Aunque la propuesta de utilizar thinklets [5] es interesante para documentar la dinámica de la colaboración, CSSL proporciona una visión más completa del sistema. Además de modelar la interacción, CSSL define las herramientas colaborativas (Tools) que se utilizarán en cada momento, el entorno de colaboración (Workspace) y especialmente el modelado de awareness, indicando el tipo de información, en qué elemento colaborativo se mostrará y cómo se mantendrá

actualizado. En esta sección se presentó un ejemplo de un modelo de thinklet con CSSL, como un caso de estudio.

La diferencia entre CSSL y la mayoría de las propuestas relacionadas es que CSSL está formalmente vinculado a UML, ya que extiende clases UML que permiten modelar procesos colaborativos (extendiendo Activity) y protocolos (extendiendo StateMachine). Estos diseños combinan el poder del UML, con sus metaclases de Fork, Join, Decision, etc., con las acciones (operaciones) realizadas por los roles en actividades colaborativas.

La popularidad y madurez de UML favorece que CSSL se pueda adoptar ampliamente para modelar sistemas colaborativos. Además, brinda los beneficios de aprovechar las herramientas de software de modelado UML disponibles.

En cuanto a la cobertura del lenguaje específico, el poder expresivo de CSSL es suficiente para representar todos los conceptos encuestados. Además, en CSSL el modelado de awareness esta integrado sin problemas al resto de los modelos, y brinda a un nuevo y potente diseño en este tema. Por otro lado, a partir de los modelos, se puede obtener una implementación concreta de forma automática a través de transformaciones de modelos utilizando el enfoque de desarrollo de software dirigido por modelos. Otro beneficio se da en términos de estandarización, reutilización de código y automatización.

7.2. Evaluación del Metamodelo

Se pretende en esta sección, realizar una evaluación del metamodelo a partir de métricas establecidas en la comunidad académica. Para este análisis, se tomó como referencia el trabajo de S Robert et al. [54], donde los autores presentan un conjunto de tres métricas que determinan cuantitativamente cuan bueno es un Perfil UML[9] y que se puede aplicar a los metamodelos. Las métricas evalúan al Perfil o Metamodelo, midiendo: a) Cuán cercano es a la semántica de UML, b) Cuán complejo es, c) Cuán configurable es -.

Las métricas son:

- ANLA (Average Number Location Application o Aplicación del Promedio de Ubicacio-

[9]Explicado en la sección 4.1 en base al trabajo de Lidia Fuentes y Antonio Vallecillo [53]

nes): Da el número promedio de elementos UML a los que los estereotipos de un Perfil UML pueden ser aplicados y se calcula como:

$$\mathbf{anla} = \mathbf{1/n} \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{Mi} \tag{7.1}$$

Donde n es el número de estereotipos del perfil y Mi es la cantidad de metaclases a las que el estereotipo i-ésimo se puede aplicar. El estereotipo puede ser aplicado a cualquier subclase de la metaclase que extiende. Por lo que se obtendrá un anla mayor cuando las metaclases extendidas se encuentren más cerca del elemento "raíz" del metamodelo referenciado por el perfil que de las "hojas". Por ejemplo, un estereotipo que extiende la metaclase Element puede ser aplicado a 199 metaclases. El ANLA da un indicio de cuánto depende el perfil, del metamodelo UML. Un ANLA alto, representa un perfil que extiende metaclases cercanas a los elementos raíz del metamodelo, y debido a que la semántica de los mismos es más general, el perfil resultante no será tan dependiente del metamodelo UML. Por el contrario, un ANLA bajo, representa un perfil que extiende metaclases cercanas a las hojas, y obtendremos un perfil dependiente del metamodelo de UML.

Para aplicar esta métrica al lenguaje CSSL (un Metamodelo), hay que tomar cada metaclase y analizar si extiende una metaclase de UML cercana a la raíz o más cercana a las hojas de UML. Esto se puede ver a simple vista en la figura 7.1, que tiene un árbol de las principales metaclases de UML, dónde en naranja están las Metaclases de UML extendidas por el lenguaje CSSL.

Para obtener un valor que represente el nivel de ANLA de un metamodelo, se realiza una variante de la fórmula ANLA original que, para cada metaclase, se cuente el largo del recorrido entre las sub-clases que va desde la metaclase extendida hasta llegar a la hoja más lejana. Por otro lado, hay que contar el largo del recorrido hacia la raíz. Por ejemplo, para una metaclase que extienda a la metaclase Class, se tiene que el recorrido inferior entre las subclases de Class da un RI=3 y el recorrido hacia arriba da RS=7. Para este caso, el resultado quedaría 3/(3+7)=0.3.

La variante de la fórmula de anla quedaría así:

$$\mathbf{anla1} = \mathbf{1/n} \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{RIi}/(\mathbf{RSi} + \mathbf{RIi}) \tag{7.2}$$

- ALDIT (Average Leaf Depth of Inheritance Tree o Profundidad Promedio de la Hoja en el Árbol de Herencia): Representa la longitud promedio del árbol de herencia en los

estereotipos. Para aplicar el ALDIT al metamodelo se cuenta las nuevas metaclases que extienden una metaclase de CSSL y calculamos el promedio con la siguiente formula:

$$\mathbf{aldit} = \mathbf{1/n} \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{Li} \tag{7.3}$$

Donde n es el número de número de metaclases agregadas y para cada una de ellas Li = 1 si extiende una metaclase de CSSL y Li=0 si extiende alguna metaclase de UML.

- ASWA (Average Stereotype With Attributes o Promedio de Estereotipos con Atributos): Como su número lo indica, da un promedio de la cantidad de estereotipos que tienen atributos. Para aplicar al metamodelo CSSL, el ASWA da un promedio de la cantidad de metaclases que extienden metaclases de UML que tienen atributos, de la siguiente manera:

$$\mathbf{aswa} = \mathbf{1/n} \sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{Ai} \tag{7.4}$$

Donde n es el número de metaclases de CSSL y Ai = 1 si la metaclase tiene uno o más atributos, o Ai = 0 en cualquier otro caso.

El ANLA da un indicio de cuánto depende el perfil del metamodelo UML (el metamodelo que extiende UML). Un ANLA alto representa un metamodelo que extiende metaclases cercanas a los elementos raíz de UML, y debido a que la semántica de los mismos es más general, el metamodelo resultante no será tan dependiente del metamodelo UML. Las métricas ALDIT y ASWA dan una idea de la complejidad del metamodelo que extiende UML considerado. Por su parte ASWA también muestra si el metamodelo que extiende UML puede ser parametrizado por los usuarios. Para estudiar estas métricas se tomará en cuenta el metamodelo que extiende UML (el metamodelo CSSL), de las figuras 4.5, 4.13, 4.14, 4.12.

Al aplicar las definiciones de las métricas al metamodelo CSSL se obtiene los siguientes valores. CSSL cuenta con 25 metaclases, entonces el valor $\mathbf{n = 25}$.

- **ANLA**:
  - Las metaclases que extienden la metaclase “Class” pueden ser aplicados también a las metaclases “Component” y “AssociationClass”. 9 metaclases *2=18

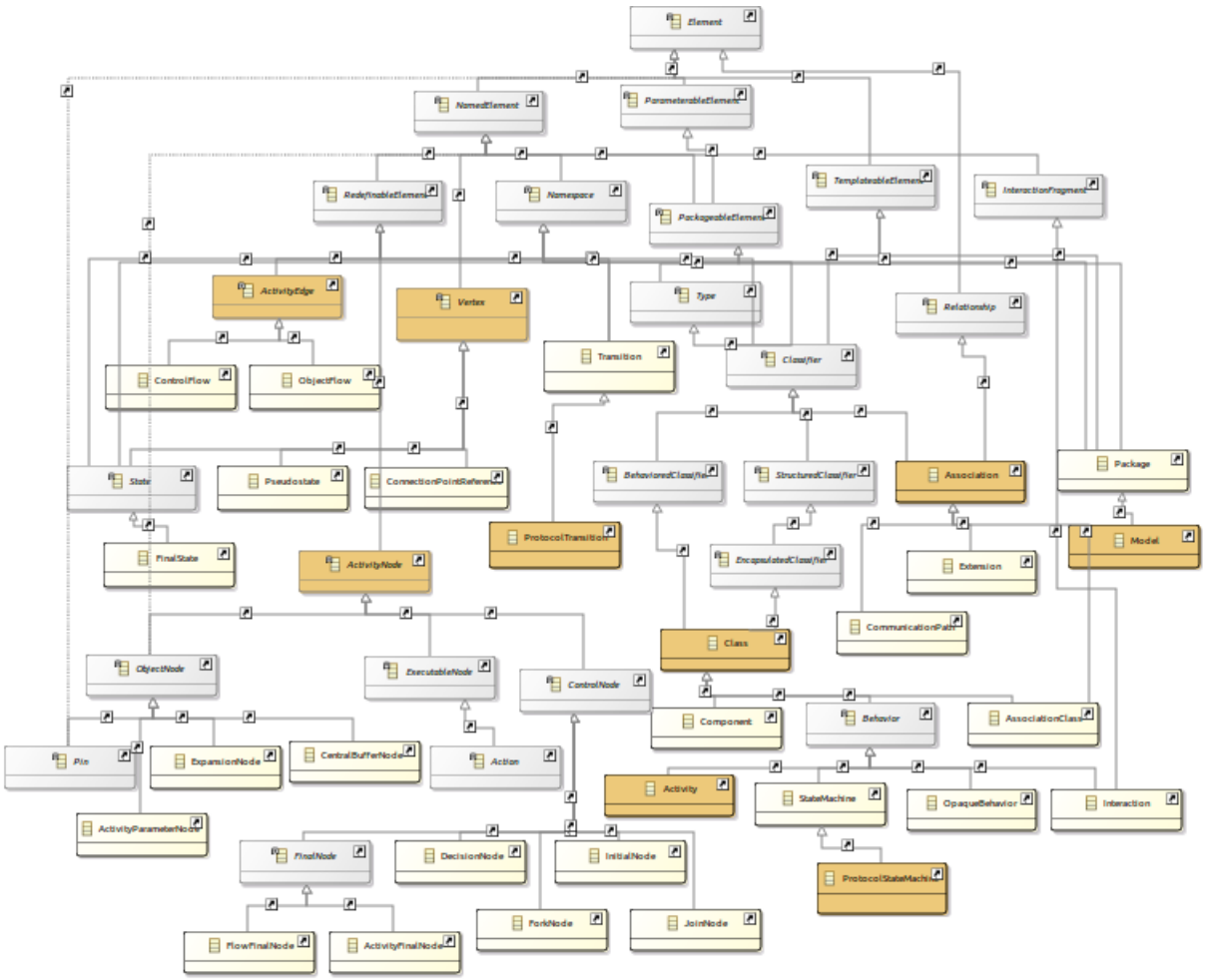


Figura 7.1: Metaclases de UML para calcular el ANLA

- Los que extienden a la metaclase “Model”, solo pueden ser aplicados a esa metaclases. 1 clase *1=1

- Los enumerativos permiten clasificar las metaclases según la cantidad de literales tengan. En el metamodelo CSSL se cuenta con los siguientes enumerativos “EnumActivityKind”, “EnumGroupkind”, “EnumSubActivityKind” que tienen 3, 4 y 5 literales respectivamente =12

- Las metaclases que extienden ”Association” también pueden aplicarse a: AssociationClass, Extension, CommunicationPath. En consecuencia,se tiene 6 metaclases *3=18

- Se cuenta con una metaclase que extiende ActivityNode que puede aplicarse a 3 metaclases de UML (ControlNode, ExecutableNode, ObjectNode). Por lo tanto, se suma 3

- Se cuenta con una metaclase que extiende ActivityEdge que puede aplicarse a 2 metaclases de UML (ObjectFlow y ControlFlow). Por lo tanto, se suma 2
- Una metaclase que extiende Vertex que se aplican a 3 metaclases (ConnectionPointReference, Pseudostate, State) que suma 3
- En en caso de la metaclase que extiende ProtocolTransition suma 1
- **El ANLA da 58/25=2.32**

  Este resultado es relativamente bajo, teniendo en cuenta que los perfiles analizados en el trabajo de Robert et al. [54] - SPEM [32], SysML [52], MARTE [56], UPDM [57] and SPT [58] - tienen valores de 19, 14, 49, 6.8 y 29 respectivamente. Esto quiere decir que el metamodelo CSSL depende de la semántica de UML, lo cual permite reutilizar las herramientas que implementan el metamodelo de UML (editores, conversores de código, perfiles, etc.) con el metamodelo CSSL.

- **ALDIT**:
  - Para el cálculo de esta métrica se tienen 9 metaclases del metamodelo CSSL cuyas superclases son a su vez metaclases. Por lo que de la fórmula de aldit se obtiene:
  - **ALDIT= 9/25 = 0.36**
- **ASWA**:
  - En este caso la cantidad de metaclases del metamodelo CSSL que poseen algún atributo es 14, entonces:
  - **ASWA= 14/25= 0.56**
- De los resultados de ALDIT y ASWA, se puede ver que el metamodelo CSSL tiene una complejidad baja (0.36), con un árbol de herencia corto y que brinda buenas posibilidades de parametrización por parte de los usuarios a partir de los atributos que se pueden utilizar (0.56).

7.3. Validación del Modelo Conceptual

Según el trabajo de Sargent [59], validar un modelo conceptual significa determinar que (1) las teorías y asunciones en la construcción del modelo son correctas, y (2) la representación del

modelo del dominio, la estructura, la lógica y las relaciones matemáticas y causales del modelos son razonables, para el propósito previsto del modelo.

Luego Sargent [59], indica que las teorías y asunciones subyacentes en la construcción del modelo se testean usando análisis matemáticos y métodos estadísticos. Esto significa revisar las opiniones de otros expertos en la materia, que utilicen los mismo conceptos, estructuras y relaciones para validar el modelo conceptual. Por esta razón, se llevó a cabo una revisión sistemática [37] para corroborar que la comunidad científica acepta los conceptos, las estructuras y relaciones incluidos en el Metamodelo CSSL.

Con la revisión sistemática de la literatura [37], se analizaron distintos modelos y ontologías de Sistemas Colaborativos, viendo que en general utilizan los mismos elementos conceptuales. Estos son "Shared Object", "Tool", "Collaborative Activity", "Workspace", "User/Role", "Group". Estos elementos también tienen relación con el concepto de awareness. Como se vió en [22], una de las preguntas que se utilizan para identificar el awareness son: "What roles will the other members of the group assume?" en donde aparece el concepto de Rol asociado al awareness. También hay otros ejemplos de awareness donde se preguntan: How can I help other participants to complete the project? or What are they doing? or where are they?. Además, la mayoría de los trabajos incorporan el concepto de "Task", que representa las tareas colaborativas que tienen que realizar los usuarios.

Como resultado de la revisión [37] se puede decir que el metamodelo CSSL contempla e incluye a todos los conceptos presentados en los distintos trabajos. Un aspecto que se puede destacar es que el metamodelo agrega el concepto "Awareness" relacionado a los "Collaborative Elements", a los "Collaborative Process" y a los "Protocol", permitiendo realizar diseños que incluyen "Awareness" en los modelos creados con el metamodelo.

Se puede concluir entonces, que el modelo conceptual del metamodelo puede ser considerado correcto y razonable y entonces válido. Esta afirmación surge a partir de la revisión de los trabajos de distintos expertos, donde presentaron los conceptos que definen a los sistemas colaborativos con awareness.

7.4. Verificación del Modelo

En el trabajo de Sargent [59], también se explica que la verificación del modelo se puede realizar a través de una verificación computarizada. De esta forma se asegura que la programación y la implementación del modelo conceptual sean correctas. Así se llega a que los modelos construidos sean implementables y que los programas que se desarrollan a partir de estos modelos funcionan. Para ayudar a garantizar que se obtenga un programa informático correcto, se deben utilizar los procedimientos de desarrollo de software que se encuentran en el campo de la ingeniería de software, tanto en el desarrollo, el diseño del programa y la implementación del programa informático.

Para el caso del metamodelo asociado al lenguaje CSSL, en el capítulo 6 se presentó una semántica del lenguaje en la forma de una transformación de modelo a texto. Con la implementación se obtiene una versión web ejecutable que resuelve aspectos centrales de los sistemas colaborativos. Una vez obtenido el sistema en funcionamiento, se cuenta con un sistema web que permite:

- Registrar usuarios/roles en el sistema
- Seguridad, manejo de sesión, login y logout
- Permitir que los usuarios/roles ejecuten operaciones de los elementos colaborativos
- Accedan a espacios colaborativos
- Participen de actividades colaborativas
- Utilicen herramientas colaborativas
- Coordinar la participación de los usuarios dentro de las actividades colaborativas (Protocolos)
- Ordenar las actividades colaborativas en procesos colaborativos
- Manejar la creación de instancias dentro del sistema
- Brindar una plataforma de Awareness
- Construcción del esqueleto de la Aplicación

Una vez que el programa ha sido desarrollado y corregido de posibles bugs, se pueden realizar los primeros test para ver si el funcionamiento del sistema es correcto. En esta etapa, se realizan distintos tipos de test que van desde test de unidad (unit test), test de integración (integration test) a test de aceptación de usuarios (acceptance test). Por ejemplo, se puede verificar que la navegación entre los espacios colaborativos sea natural, que los awareness especificados sean claros y que las actividades sean suficientes para que el sistema cumpla con las necesidades del que se pretenden modelar.

En este punto se validan los requerimientos originales y se prueban alternativas de diseño para lograr una aplicación que funcione correctamente. Los analistas podrán corregir los modelos con el lenguaje CSSL y obtener versiones del sistema aplicando las transformaciones mencionadas. De esta forma se evidencian las ventajas de MDD permitiendo iterar entre el modelado y testing varias veces hasta obtener el sistema que cumpla con los requerimientos originales.

7.5. Conclusiones del capítulo

En el capítulo se realiza una evaluación, validación y verificación del lenguaje CSSL. La evaluación determinó que el lenguaje depende de la semántica de UML, lo cual permite reutilizar las herramientas vinculadas a UML (editores, conversores de código, perfiles, etc.) con el metamodelo CSSL. También se puede ver que el metamodelo CSSL tiene una complejidad baja con un árbol de herencia corto y que brinda buenas posibilidades de parametrización por parte de los usuarios.

Por otro lado, el modelo conceptual del lenguaje se considera correcto y razonable a partir del resultado de la revisión sistemática de distintos trabajos de expertos de la comunidad científica. En especial, la inclusión del concepto “Awareness” relacionado a los “Collaborative Elements”, a los “Collaborative Process” y a los “Protocol”, permitiendo realizar diseños que incluyen “Awareness” en los modelos creados con el metamodelo.

Finalmente, la verificación del lenguaje CSSL, se obtiene a partir de la implementación de una semántica concreta que permite obtener versiones Web ejecutables. Con las aplicaciones funcionando, los desarrolladores cuentan con un conjunto de funcionalidad colaborativa que permite fácilmente comprobar si los diseños responden a los requerimientos del sistema que se

está construyendo.

Capítulo 8

Conclusiones

Los sistemas colaborativos tienen aspectos muy complejos de desarrollar. Especialmente el hecho de tener un conjunto de datos compartidos que cualquier colaborador puede modificar y el awareness que requiere mantener a los usuarios informados sobre el estado de la colaboración mientras ésta se lleva a cabo. Existen algunos modelos abstractos [22, 47, 60] que permiten describir los conceptos principales de estos sistemas. Sin embargo, el diseño de conceptos tales como awareness y procesos colaborativos no están totalmente cubiertos por estos modelos. Por otro lado, existen algunos frameworks y herramientas [8, 46, 48] que implementan la funcionalidad común y facilitan el desarrollo. Pero estos productos tienen falencias a la hora de mejorar el reuso de componentes, adaptarse a los cambios tecnológicos y proveer calidad en los productos que se desarrollen.

Siguiendo los principios de MDD en el desarrollo de los sistemas colaborativos, en esta tesis se presenta el lenguaje específico de dominio CSSL v2.0 [13]. El lenguaje propuesto permite modelar situaciones colaborativas dinámicas como procesos, protocolos y awareness. Fue definido como una extensión de UML, usando el mecanismo de metamodelado y fue implementado usando herramientas open source sobre la plataforma de Eclipse [56]. Se desarrolló una sintaxis concreta para CSSL a través de un conjunto de editores (CSSL Tool), que permiten crear modelos de sistemas colaborativos, instanciando los conceptos del metamodelo desarrollado. Además de las características estáticas se pueden modelar aspectos dinámicos como procesos colaborativos, protocolos y awareness. Los modelos escritos con la herramienta CSSL Tool, son independientes del framework, lenguaje o herramientas de implementación.

Continuando con la metodología MDA, se provee una semántica al lenguaje a través de transformaciones de modelo a texto, obteniendo código ejecutable a partir de los modelos expresados con el lenguaje CSSL v2.0. Se presenta una transformación implementada con Acceleo [55], que permite obtener una versión Web a partir de los modelos colaborativos. Se eligió un conjunto de tecnologías, livianas y flexible basadas en javascript (Express, Node.js, Angular, MongoDB, etc.). Asimismo se describe cómo se mapean los conceptos del sistema colaborativo en componentes de la aplicación, tanto en el servidor como en el cliente. La implementación permite obtener una versión ejecutable que resuelve el manejo de los usuarios y roles, qué acciones pueden realizar en cada momento y se controla a cuál espacio pueden ingresar. Se focaliza en la implementación de los procesos colaborativos y controla el estado de las instancias de cada uno de ellos. Finalmente se muestra el funcionamiento del awareness que se implementa a través de WebSockets.

Los resultados de este trabajo, comprueban que el lenguaje CSSL v2.0, permite definir de forma precisa, concisa y amigable los conceptos abstractos de los sistemas colaborativos, incluyendo los procesos colaborativos, protocolos y awareness. Se destaca la implementación de una sintaxis concreta a través de editores gráficos, basados en herramientas open source sobre Eclipse [61], que permiten a los diseñadores construir modelos que simplifican la comunicación entre los miembros del equipo de desarrollo. Un aspecto importante de destacar es que los modelos están construidos manteniendo la compatibilidad con UML lo que posibilita el intercambio con otras herramientas, como editores, traductores u otros perfiles de UML. Esto permite a los diseñadores combinar las herramientas desarrolladas para CSSL y para UML.

A través de transformación de modelos se obtienen versiones ejecutables, utilizando tecnología Web relacionadas a Javascript (Express.js, Angular.js, Node.js), MongoDB y Websockets, que validan la estructura de los modelos y brindan una plataforma para continuar el desarrollo. Los desarrolladores continúan a partir de un modelo implementado que tiene resuelto el manejo de los usuarios, roles y operaciones, procesos colaborativos y awareness.

Finalmente se realizó una evaluación del metamodelo para verificar que su uso tiene una complejidad baja, con una semántica fuertemente asociada a UML y con buenas posibilidades de configuración. Por otro lado, se verificó la correctitud del metamodelo usando la revisión de los trabajos de distintos expertos de la comunidad científica. También se realizó una validación del metamodelo a partir de la transformación donde se obtiene una versión web ejecutable que

resuelve aspectos centrales de los sistemas colaborativos.

8.1. Trabajo Futuro

Para continuar el trabajo de investigación se abrieron cuatro líneas de acción para los próximos años. La primera tiene que ver con la vinculación con otros perfiles de UML que complementen el lenguaje CSSL. Por ejemplo, utilizar perfiles Mobile para aplicarlo a alguno de los conceptos del lenguaje y combinar posibles transformaciones para obtener variantes de las versiones ejecutables. Esto permitiría armar una factoría de aplicaciones colaborativas para distintas plataformas, explotando la potencia de las herramientas de transformación de modelos.

Por otro lado, se pretende incorporar herramientas de evaluación de usabilidad. Especialmente para evaluar el impacto del funcionalidad de awareness en la usabilidad de los sistemas colaborativos. Esto permitirá, no sólo, evaluar las aplicaciones colaborativas producidas (juegos colaborativos, entornos de enseñanza, toma de decisiones grupales) sino evaluar el lenguaje CSSL tanto en su sintaxis como su semántica.

Otra línea de trabajo que se abre, es la posibilidad de incorporar funcionalidad colaborativa a los editores implementados para soportar la representación gráfica del lenguaje. Este desafío permitirá construir herramientas colaborativas para desarrollar sistemas colaborativos.

Finalmente. se buscará desarrollar un diseño de una arquitectura para soportar servicios de awareness para sistemas colaborativos de gran escala. El objetivo de esta arquitectura es el de soportar distintos tipos de awareness (presencia, ubicación, acción, etc.) con sus diferentes variantes de implementación (sincrónicos/asincrónicos – volátiles/persistentes, etc.). La arquitectura propuesta, orientada a microservicios, deberá proveer escalabilidad (vertical y horizontal), mejorar el mantenimiento de los sistemas desarrollados, propiciar el desarrollo y la integración continua y facilitar la experimentación e integración de nuevas tecnologías.

Bibliografía

[1] C. A. Ellis, S. J. Gibbs, and G. Rein, "Groupware: some issues and experiences," *Communications of the ACM*, vol. 34, pp. 39–58, 1 1991.

[2] J. Grudin, "Computer Supported Cooperative Work: History and Focus," *Computer*, 1994.

[3] J. Grudin and S. E. Poltrock, "Computer-Supported Cooperative Work and Groupware," *Advances in Computers*, vol. 45, pp. 269–320, 1 1997.

[4] G. Henri Ter Hofte and D. H. J. Van Der Lugt, *Working Apart Together Foundations for Component Groupware*. PhD thesis, University of Twente, 1998.

[5] R. O. Briggs, Gert-Jan de Vreede, and G. Kolfschoten, "ThinkLets for E-collaboration," *igi-global.com*, 2007.

[6] P. Dourish and V. Bellotti, "Awareness and coordination in shared workspaces," *Proceedings of the 1992 ACM conference on Computer-supported cooperative work - CSCW '92*, pp. 107–114, 1992.

[7] C. Gutwin, S. Greenberg, and M. Roseman, "Workspace awareness in real-time distributed groupware: Framework, widgets, and evaluation," in *Proceedings of HCI '96*, 1996.

[8] C. Gutwin and S. Greenberg, "A descriptive framework of workspace awareness for real-time groupware," *Computer Supported Cooperative Work*, 2002.

[9] P. Dourish and S. Bly, "Portholes: Supporting awareness in a distributed work group," *dl.acm.org*, 1992.

[10] C. Pons, R. S. Giandini, and G. Pérez, *Desarrollo de software dirigido por modelos*. Editorial de la Universidad Nacional de La Plata (EDULP) / McGraw-Hill Educación, 2010.

[11] A. G. Kleppe, J. B. Warmer, and W. Bast, *MDA explained : the model driven architecture : practice and promise*. Addison-Wesley, 2003.

[12] S. J. Mellor, *MDA distilled : principles of model-driven architecture*. Addison-Wesley, 2004.

[13] L. M. Bibbo, D. García, and C. Pons, "A Domain Specific Language for the Development of Collaborative Systems," *2008 International Conference of the Chilean Computer Science Society*, pp. 3–12, 11 2008.

[14] L. M. Bibbo, "Modelado de Sistemas Colaborativos," *Tesis de Magister en Ingeniería de Software; Universidad Nacional de La Plata*, pp. 1–143, 10 2009.

[15] L. M. Bibbo, R. Giandini, and C. Pons, "DSL for Collaborative Systems with Awareness," *SLISW - Simposio Latinoamericano de Ingeniería de Software; XLIII CLEI - Conferencia Latinoamericana de Informática*, 2017.

[16] ISO, "Unified Modeling Language Specification Version 2.5.1."

[17] A. Solano, T. Granollers, C. A. Collazos, and C. Rusu, "Proposing formal notation for modeling collaborative processes extending HAMSTERS notation," in *Advances in Intelligent Systems and Computing*, vol. 275 AISC, pp. 257–266, Springer Verlag, 2014.

[18] B. Bauer, S. Roser, and J. P. Müller, "Adaptive Design of Cross-Organizational Business Processes Using a Model-Driven Architecture," in *Wirtschaftsinformatik 2005*, pp. 103–121, Heidelberg: Physica-Verlag HD, 2005.

[19] J. Highsmith and A. Cockburn, "Agile software development: The business of innovation," 2001.

[20] S. Greenberg and C. Gutwin, "Implications of We-Awareness to the Design of Distributed Groupware Tools," *Computer Supported Cooperative Work: CSCW: An International Journal*, vol. 25, no. 4-5, 2016.

[21] C. A. Collazos, F. L. Gutiérrez, J. Gallardo, M. Ortega, H. M. Fardoun, and A. I. Molina, "Descriptive theory of awareness for groupware development," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, pp. 4789–4818, 12 2019.

[22] F. Belkadi, E. Bonjour, M. Camargo, N. Troussier, and B. Eynard, "A situation model to support awareness in collaborative design," 2013.

[23] S. Greenberg and M. Roseman, "Using a Room Metaphor to Ease Transitions in Groupware," *Sharing Expertise: Beyond Knowledge Management*, 2003.

[24] B. Boehm W., "Software Engineering," *IEEE Transactions on Computers*, vol. C-25, pp. 1226–1241, 12 1976.

[25] W. F. Bauer and A. M. Rosenberg, "Software," in *Proceedings of the December 5-7, 1972, fall joint computer conference, part II on - AFIPS '72 (Fall, part II)*, (New York, New York, USA), p. 993, ACM Press, 1972.

[26] "IEEE - The world's largest technical professional organization dedicated to advancing technology for the benefit of humanity.."

[27] OMG, "Meta Object Facility Specification v1.4," *4, April*, vol. 4, no. April, pp. 02–04, 2002.

[28] J.-M. Favre, J. Estublier, and M. Blay-Fornarino, *L'ingeènierie dirigeèe par les modeàles au-delaà du MDA*. Hermeàs Science Publications, 2006.

[29] R. Giandini, *Un Marco Formal para Transformaciones en la Ingeniería de Software Conducida por Modelos*. PhD thesis, Universidad Nacional de La Plata, 2007.

[30] https://www.omg.org/, "OMG — Object Management Group."

[31] "About the Meta Object Facility Specification Version 2.5."

[32] "About the Software & Systems Process Engineering Metamodel Specification Version 2.0."

[33] "About the Common Warehouse Metamodel Specification Version 1.1."

[34] M. Brambilla, J. Cabot, and M. Wimmer, "Model-Driven Software Engineering in Practice," *Synthesis Lectures on Software Engineering*, vol. 1, pp. 1–182, 9 2012.

[35] R. Giandini and C. Pons, "Un lenguaje de Transformación específico para Modelos de Proceso del Negocio," *CLEI*, no. Mdd, 2010.

[36] https://www.omg.org/mda/, "Model Driven Architecture (MDA) — Object Management Group."

[37] L. M. Bibbo, R. Giandini, and C. Pons, "Sistemas Colaborativos con Awareness: Requisitos para su Modelado," *XLV Jornadas Argentinas de Informática e Investigación Operativa (45 JAIIO) - Simposio Argentino de Ingeniería de Software (ASSE 2016)*, no. Ciencias Informáticas, pp. 111–122, 2016.

[38] B. Kitchenham and O. Brereton, "Systematic literature reviews in software engineering–a systematic literature review," *Elsevier*, 2009.

[39] J. Gallardo, C. Bravo, and M. A. Redondo, "A model-driven development method for collaborative modeling tools," *Journal of Network and Computer Applications*, vol. 35, pp. 1086–1105, 5 2012.

[40] A. I. Molina, W. J. Giraldo, M. Ortega, M. A. Redondo, and C. A. Collazos, "Model-driven development of interactive groupware systems: Integration into the software development process," *Science of Computer Programming*, vol. 89, pp. 320–349, 9 2014.

[41] M. A. Teruel, E. Navarro, V. López-Jaquero, F. Montero, and P. González, "An extension of i * to Model Requirements for CSCW Systems Applied to Conference Preparation System with Collaborative Reviews," in *5th International i* Workshop (iStar'11)*, pp. 84–89, 2011.

[42] M. A. Teruel, E. Navarro, V. López-Jaquero, F. Montero, J. Jaen, and P. González, "Analyzing the understandability of Requirements Engineering languages for CSCW systems: A family of experiments," *Information and Software Technology*, 2012.

[43] M. A. Teruel, E. Navarro, V. López-Jaquero, F. Montero, and P. González, "Comparing Goal-Oriented Approaches to Model Requirements for CSCW," in *Evaluation of Novel Approaches to Software Engineering*, pp. 169–184, Springer, Berlin, Heidelberg, 2013.

[44] M. A. Teruel, E. Navarro, V. López-Jaquero, F. Montero, and P. González, "CSRML Tool: A visual studio extension for modeling CSCW requirements," in *CEUR Workshop Proceedings*, 2013.

[45] M. A. Teruel, E. Navarro, V. Lopez-Jaquero, F. Montero, and P. Gonzalez, "A design pattern for representing Workspace Awareness," in *Proceedings of the 2014 IEEE 18th International Conference on Computer Supported Cooperative Work in Design, CSCWD 2014*, 2014.

[46] A. Kamoun, S. Tazi, and K. Drira, "FADYRCOS, a semantic interoperability framework for collaborative model-based dynamic reconfiguration of networked services," *Computers in Industry*, vol. 63, no. 8, pp. 756–765, 2012.

[47] J. Gallardo, A. I. Molina, C. Bravo, M. A. Redondo, and C. A. Collazos, "An ontological conceptualization approach for awareness in domain-independent collaborative modeling systems: Application to a model-driven development method," in *Expert Systems with Applications*, 2011.

[48] F. Gallego, A. I. Molina, J. Gallardo, and C. Bravo, "A conceptual framework for modeling awareness mechanisms in collaborative Systems," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6949, pp. 454–457, 2011.

[49] V. Vieira, P. Tedesco, and A. C. Salgado, "Using a metamodel to design structural and behavioral aspects in context-sensitive groupware," in *Proceedings of the 2010 14th International Conference on Computer Supported Cooperative Work in Design, CSCWD 2010*, pp. 59–64, 2010.

[50] A. I. Molina, J. Gallardo, M. A. Redondo, M. Ortega, and W. J. Giraldo, "Metamodel-driven definition of a visual modeling language for specifying interactive groupware applications: An empirical study," *Journal of Systems and Software*, vol. 86, no. 7, 2013.

[51] https://www.uml.org/index.htm, "UML-Specification."

[52] "About the OMG System Modeling Language Specification Version 1.6 beta."

[53] Lidia Fuentes and Antonio Vallecillo, "An introduction to UML profiles," *cepis.org/upgrade*, no. II (04), 2004.

[54] S. Robert, S. Gérard, F. Terrier, and F. Lagarde, "A Lightweight Approach for Domain-Specific Modeling Languages Design," in *2009 35th Euromicro Conference on Software Engineering and Advanced Applications*, pp. 155–161, IEEE, 2009.

[55] https://www.eclipse.org/acceleo/, "Acceleo — Home."

[56] "About the UML Profile for MARTE Specification Version 1.2."

[57] "About the Unified Profile for DoDAF and MODAF Specification Version 2.1.1."

[58] "About the UML Profile for Schedulability, Performance, & Time Specification Version 1.1."

[59] R. G. Sargent, "Verification and validation of simulation models," *Journal of Simulation*, vol. 7, no. 1, pp. 12–24, 2013.

[60] M. A. Teruel, E. Navarro, V. López-Jaquero, F. Montero, and P. González, "A CSCW Requirements Engineering CASE Tool: Development and usability evaluation," *Information and Software Technology*, vol. 56, no. 8, 2014.

[61] http://www.eclipse.org/, "Eclipse - an open development platform."