Modelado de Derivación de Código para el Desarrollo de Sistemas Colaborativos con Awareness
Modeling and Code Derivation for Collaborative Systems with Awareness

Luis Mariano Bibbo[1], Roxana Giandini[1,2,3], and Claudia Pons[1,2,4]

[1] LIFIA - Facultad de Informática - UNLP, La Plata, Argentina
[2] CICPBA, Comisión de Investigaciones Científicas, Buenos Aires, Argentina.
[3] Grupo GIDAS - UTN - Facultad Regional La Plata
[4] Centro de Altos Estudios en Tecnología Informática (CAETI) de la Universidad Abierta Interamericana (UAI), Buenos Aires, Argentina

Resumen La construcción de sistemas colaborativos con awareness es una tarea muy compleja. En este artículo se presenta la forma de utilización del 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 los sistemas colaborativos, especialmente los procesos colaborativos, protocolos y awareness. A partir de casos de modelado se muestra una sintaxis concreta -editores gráficos- que permiten construir modelos de sistemas colaborativos. Estos son independientes de la plataforma de implementación y están formalmente preparados para derivarlos utilizando transformaciones MDD. Luego se presenta una semántica del lenguaje a través de transformaciones de modelo a texto donde se obtiene versiones Web con tecnologías JavaScript, MongoDB y Websockets. Esto aporta a los desarrolladores de Sistemas Colaborativos un conjunto de herramientas que les permiten por un lado modelar los sistemas y por otro obtener aplicaciones ejecutables con aspectos centrales resueltos como la implementación de procesos colaborativos, awareness y el control de las operaciones que los roles realizan en el sistema.

Abstract Building collaborative systems with awareness is a very complex task. This article presents the use 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. A concrete syntax - graphic editors - used to build models of collaborative systems is presented and modeling examples are

shown. The models are independent of the implementation platform and are formally prepared to be derived using MDD transformations. Then model-to-text transformations where Web versions are obtained with JavaScript, MongoDB and Websockets technologies, are introduced as language semantics. This gives the Collaborative Systems developers a set of tools that allow them to model and develop executable applications to test the specified functionality. From the models, the result will be the implementation of collaborative processes, awareness and the operations that the roles perform in the system.

Keywords: Collaborative Software · Awareness · Model-Driven Engineering · Meta-Model · Code Generation.

1 Introducción

Los sistemas colaborativos atienden a las necesidades de grupos de usuarios interactuando en un espacio compartido. De acuerdo a Ellis et al. [10], 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". Los usuarios suelen utilizar estos sistemas para compartir información, comunicarse, coordinar actividades y colaborar. Esta tecnología brinda un equilibrio entre el trabajo individual de los participantes y la colaboración que realizan los usuarios para lograr un objetivo grupal. La colaboración está dada a través de las herramientas donde los usuarios construyen el contenido compartido. Una efectiva colaboración se mejora cuando la participación de los usuarios está coordinada, como se explicó en el trabajo de Grudin [12]. La coordinación se modela a través de procesos colaborativos que determinan en qué orden se llevan adelante las actividades del grupo. También la interacción de los participantes dentro de una actividad puede ser diseñada usando los protocolos de colaboración que definen las operaciones concretas que pueden realizar los roles en cada estado de una actividad.

El Awareness según Dourish y Bellotti [9], es "conocimiento de las actividades de otros que provee contexto para tus propias actividades". Permite a los usuarios coordinar su trabajo basado en el conocimiento de lo que otros hacen o han hecho. Los usuarios podrán ver por ejemplo los cambios en los documentos o el grado de avance de alguna tarea. Esta información -el awareness- motiva la colaboración espontánea y puede ser un disparador para comunicarse con los otros colaboradores.

Entre los awareness más comunes aparecen: presencia, ubicación, 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. Los distintos tipos de awareness, pueden asociarse, agrupándolos para mejorar la calidad de la información. En general, cuando se muestra la presencia del usuario se le puede

agregar otro tipo de información como el estado, la ubicación o la actividad que está realizando.

Para construir sistemas colaborativos con awareness se buscó un paradigma de construcción de software de calidad y que sea capaz de sobrevivir a la evolución de sus requisitos funcionales, manteniéndose flexible a los cambios en la tecnología que lo sustenta. En el libro de Pons, Giandini y Perez se detalla cómo MDD [13] (por sus siglas en inglés: Model Driven software Development) ofrece mejorar los procesos de construcción de software. Sus postulados básicos son los siguientes: los modelos asumen un rol protagónico en el proceso de desarrollo del software; los modelos pasan de ser entidades contemplativas para convertirse en entidades productivas a partir de las cuales se deriva la implementación en forma automática, como se explicó en el trabajo de Brambilla [8]. La iniciativa MDD promueve:

1. **Abstracción**: el uso de un mayor nivel de abstracción tanto en la especificación del a resolver como de la solución correspondiente, en relación con los métodos tradicionales de desarrollo de software.
2. **Automatización**: el aumento de confianza en la automatización asistida por computadora para soportar el análisis, el diseño y la ejecución.
3. **Estandarización**: el uso de estándares industriales como medio para facilitar las comunicaciones, la interacción entre diferentes aplicaciones y productos, y la especialización tecnológica.

Uno de los beneficios en el desarrollo de software que conlleva aplicar MDD, es la adaptación a los cambios tecnológicos ya que los modelos de alto nivel están libres de detalles de la implementación, lo cual facilita la adaptación a los cambios que pueda sufrir la plataforma tecnológica subyacente o la arquitectura de implementación. 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), focalizados y especializados 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. La técnica más usada para especificar la estructura de un DSL es el metamodelado donde se define qué elementos pueden existir en el modelo. Por ejemplo, en el metamodelo de UML se encuentra a “Class”, “Activity”, “StateMachine”, etc. que luego aparecerán instanciadas en un modelo UML.

Este artículo propone la utilización del lenguaje CSSL v2.0 [7] para obtener modelos de sistemas colaborativos, para luego transformarlos en versiones Web ejecutables. Para mostrar las ventajas del lenguaje y herramientas asociadas, organizamos el trabajo de la siguiente manera: en la sección siguiente se comenta la sintaxis abstracta y concreta del lenguaje CSSL, luego se analizan casos de modelado prototípicos, a continuación se presenta una semántica del lenguaje mediante derivación de código a partir de los modelos obtenidos y finalmente se presentan conclusiones y líneas de trabajo futuro.

2 El Lenguaje CSSL

Los lenguajes específicos de dominio, CSSL entre ellos, cuentan con una sintaxis abstracta que especifica su estructura. Por otro lado se necesita una notación específica o sintaxis concreta, que habitualmente es una notación gráfica, para que los usuarios puedan utilizar el lenguaje. Es importante destacar que una misma sintaxis abstracta podría tener diferentes sintaxis concretas. En el caso del lenguaje CSSL se usó un metamodelo para expresar la sintaxis abstracta y un conjunto de editores gráficos que soportan la sintaxis concreta permitiendo instanciar el metamodelo.

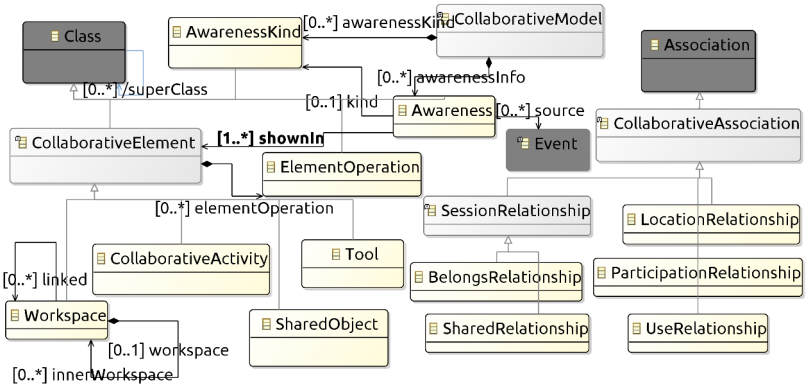


Figura 1: Modelo Conceptual con Awareness

2.1 Sintaxis abstracta: Metamodelo

El metamodelo fue construido como una extensión UML [4], contando con todas sus metaclases, presentadas en las figuras 1, 2 y 3 en gris oscuro, y otras específicas del CSSL. Fue inspirado en una revisión sistemática presentada en el trabajo [6], donde se buscaban recursos de ingeniería de software y modelos de sistemas colaborativos, destacándose los trabajos [14], [11], [5].

En los siguientes diagramas, se discute las características del lenguaje CSSL. En la figura 1 aparecen las metaclases principales del metamodelo, entre ellas *CollaborativeElement* subclase de Class de UML y *CollaborativeAssociation*, subclase de Association de UML. Esto nos permite trabajar con los conceptos principales de los sistemas colaborativos: “Shared Object”, “Tool”, “Collaborative Activity”, “Workspace”, “User”, “Role” y asociarlos expresando las características del sistema. Por ejemplo que una actividad utiliza determinadas herramientas e intervienen algunos roles.

También se puede especificar el awareness que maneja el sistema. Por un lado, se ve en la figura 1 que la clase *CollaborativeModel* conoce un conjunto de tipos

de awareness *AwarenessKind* y por otro, la clase *Awareness* que representa la información de awareness que se mostrará en los elementos colaborativos usando la relación *shownIn*.

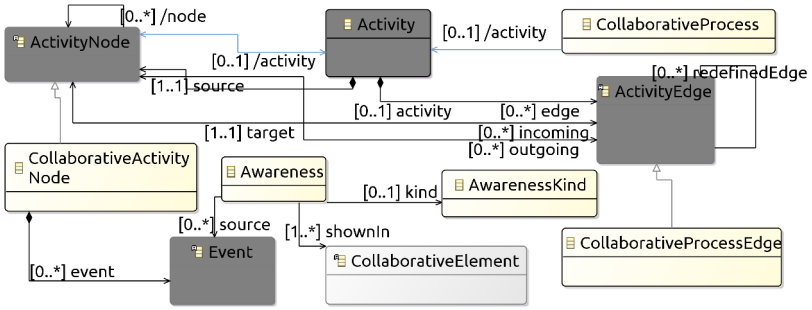


Figura 2: Modelo de Procesos Colaborativos con Awareness

En la figura 2, se muestra que la clase que representa a los procesos colaborativos *CollaborativeProcess*, tiene como comportamiento (Behavior) a una instancia de la clase Activity de UML, que está compuesta por nodos (ActivityNodes) y ejes (ActivityEdge). El metamodelo CSSL extiende estas dos últimas permitiendo tener actividades colaborativas como nodos *CollaborativeActivityNode* y ejes que se activan a partir de las operaciones que ejecutan los roles *CollaborativeProcessEdge*. También puede verse que los nodos colaborativos contienen un conjunto de eventos que pueden originar la actualización de información de awareness, que se mostrará en algún elemento colaborativo (Workspace, Tool, etc.).

En la figura 3, se presenta el modelo de protocolo, que define el comportamiento de una actividad colaborativa. Se utiliza una máquina de estado (StateMachine) para representar el protocolo de una actividad. El modelado consiste en describir los estados por lo que pasa una actividad, definiendo a los estados a partir de las operaciones que los roles pueden realizar y los eventos asociados a las mismas. Se ve en el metamodelo que la metaclase *CollaborativeActivityState*, subclase de State de UML, tiene un conjunto de operaciones asignadas a través de *assignedRoleElementOperation* y otro conjunto de operaciones, *outgoingRoleElementOperation*, que activan una transición a otro estado.

2.2 Herramientas del CSSL

Utilizando el metamodelo CSSL, se puede instanciar la metaclase *CollaborativeModel* -subclase de Model de UML- que agrupa a los elementos que intervienen en el sistema colaborativo como elementos empaquetados “PackageElement”. Así se pueden crear instancias de CollaborativeRole, CollaborativeActivity, Workspace, Process, etc.

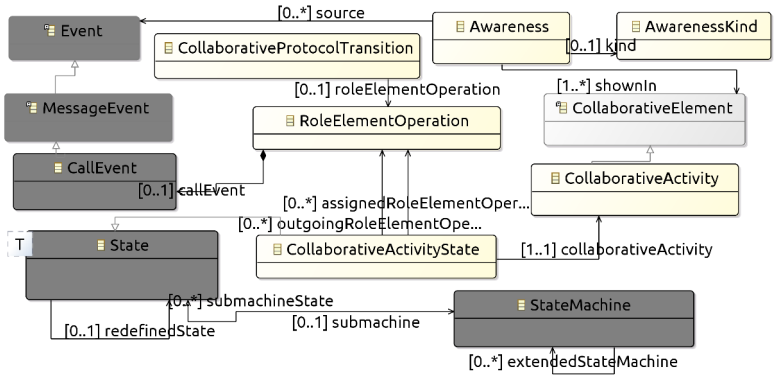
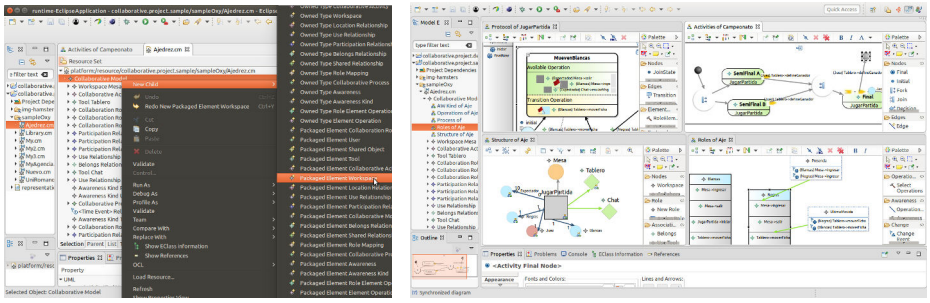


Figura 3: Modelo de Protocolos con Awareness



(a) Editor de UML (b) Editores de CSSL

Figura 4: Editores para instanciar el Metamodelo

Crear modelos instanciando a mano las clases del metamodelo, resulta una trabajo arduo y poco amigable. En la imagen 4a se muestra cómo se puede instanciar el metamodelo a partir de un editor estándar de UML. Primero se crea una instancia de CollaborativeModel y a partir de ella se van instanciando sus "hijos" de ese modelo, como se muestra en la figura 4a, donde se está creando una instancia de Workspace. CSSL brinda una sintaxis abstracta de los sistemas colaborativos y en este caso la sintaxis concreta sería la misma. Usando esta forma de instanciar el metamodelo puede verse un listado en forma de árbol con todas las clases y asociaciones instanciadas.

Para brindar mayor flexibilidad y legibilidad al lenguaje CSSL, se pueden crear otras sintaxis concretas que suelen ser más amigables para los diseñadores, es decir construir un lenguaje específico (DSL -Domain Specific Language-). Para el lenguaje CSSL se crearon un conjunto de editores, que fueron desarrollados utilizando el proyecto Sirius [3] de Eclipse. Los editores soportan una sintaxis concreta, que especifican las características de los sistemas colaborativos y permiten a los desarrolladores describir los elementos del sistema, los procesos colaborativos, los protocolos, el awareness, entre otros elementos del sistema. En la figura 4b se muestran cuatro de los editores desarrollados.

1. **Estructura del sistema:** Cuenta con un editor donde se agregan conceptos y se los relacionan. También se pueden indicar en qué elemento se mostrará información de awareness.
2. **Roles del sistema:** Se definen cuáles serán los roles que intervienen en el sistema y qué operaciones tienen asignadas. Se define también qué acciones actualizan awareness.
3. **Diagrama de Procesos:** Para cada proceso se muestra las actividades colaborativas que lo componen y en qué orden se ejecutan. En este editor se puede indicar cómo se actualiza el awareness a medida que el proceso va avanzando.
4. **Diagrama de actividad:** Muestra los estados por los que pasa una actividad colaborativa. También se puede indicar como se actualiza la información de awareness ante las acciones que ejecutan los roles.

Otros editores permiten crear procesos, tipos de awareness, agregar operaciones a los espacios, a las actividades y otros de configuración del sistema.

3 Ejemplos de Modelado

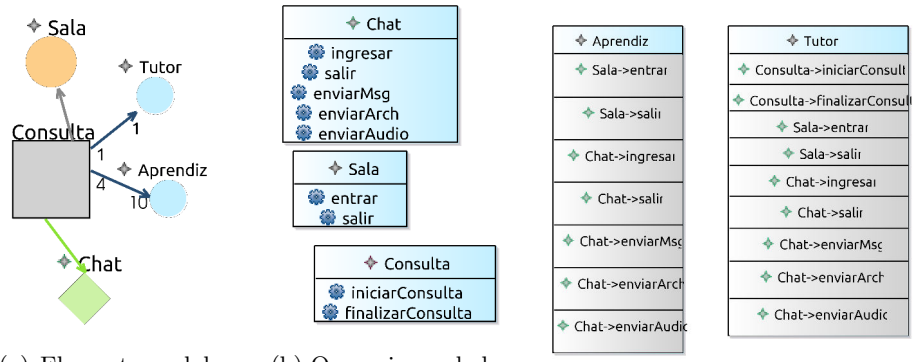
3.1 Modelado de un chat simple

Para entender los beneficios de contar con el lenguaje CSSL que permite describir modelos de sistemas colaborativos, se presenta en esta sección un conjunto de casos comunes en los sistemas colaborativos. Primero se muestra en las figuras 5, el diseño de elementos colaborativos que cuentan con un conjunto de operaciones.

Elementos colaborativos con operaciones: Se inicia el recorrido presentando el diseño de una herramienta muy usada en los sistemas colaborativos: el chat. Las operaciones que están disponible en la mayoría de los Chat son: *ingresar, salir, enviar mensaje, enviar archivo, enviar audio*, etc.

Las herramientas como el chat se utilizan en el contexto de alguna actividad colaborativa. Supongamos que se quiere modelar que el chat se utiliza en una actividad colaborativa (*Consulta*) que de desarrolla en un espacio colaborativo (*Sala*). Las actividades colaborativas también pueden tener operaciones asociadas, por ejemplo la Consulta puede tener las operaciones *iniciarConsulta* y *finalizarConsulta*. El espacio colaborativo "Sala" también puede tener operaciones asignadas, en particular, por ser un espacio los usuarios van a poder *entrar* o *salir* del mismo. Las operaciones de los elementos colaborativos se muestra en la figura 5b .

La figura 5a presenta el diseño de una actividad colaborativa que se llama "Consulta" que utiliza al Chat para comunicarse. Esta actividad tiene dos roles que participan: uno es el "Tutor" y otro es el "Aprendiz". En el diseño puede verse que participa un tutor y entre cuatro y diez Aprendices.



(a) Elementos colaborativos (b) Operaciones de los elementos (c) Operaciones de los roles

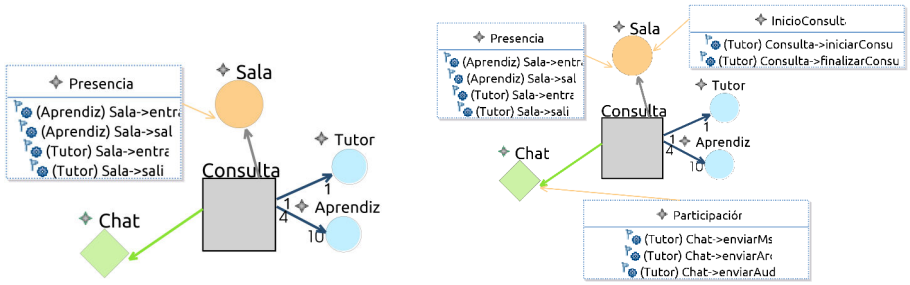
Figura 5: Modelo de un chat simple

Tomando en cuenta las operaciones asignadas a los elementos colaborativos el diseñador tiene que indicar cuáles de ellas se asignan a cada uno de los roles. En el diseño de la figura 5c se define que el rol tutor es el encargado de iniciar y finalizar la consulta. Así mismo, las operaciones del chat pueden ser ejecutadas por ambos roles.

3.2 Modelado del Awareness

El modelado del awareness consiste en describir qué tipo de información de awareness se mostrará en los elementos colaborativos; en nuestro caso la sala, la

consulta o el chat. Si se quiere mostrar la presencia de los usuarios en la sala se debe crear un elemento de awareness del tipo "presencia", asociarlo a la sala y asignarle los eventos que lo mantienen actualizado como se muestra en la fig 6a.



(a) Awareness de Presencia (b) Diseño de Awareness

Figura 6: Modelado de Awareness

De manera similar se puede agregar awareness a otros elementos colaborativos, Por ejemplo indicar cuando un tutor inició una Consulta o mostrar la cantidad de participaciones que hacen los aprendices de la misma. El diseño de los tres awareness de ejemplo se muestran en la figura 6b

3.3 Procesos y protocolos

Más allá de los aspectos que definen la estructura del sistema colaborativo, hay otros aspectos dinámicos que explican el comportamiento de los sistemas.

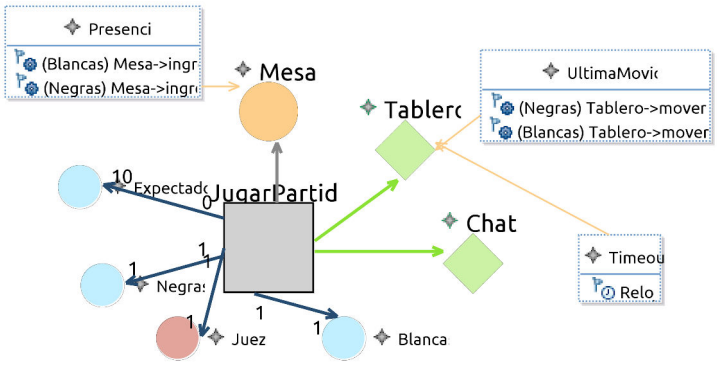


Figura 7: Modelo de Ajedrez

Por un lado, se puede 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.

Para explicar como se especifica un proceso colaborativo, se parte 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 se muestran en la figura 7.

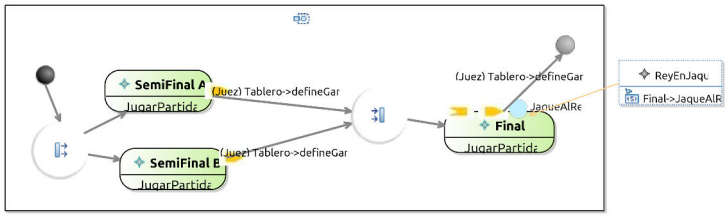


Figura 8: Proceso Colaborativo Campeonato

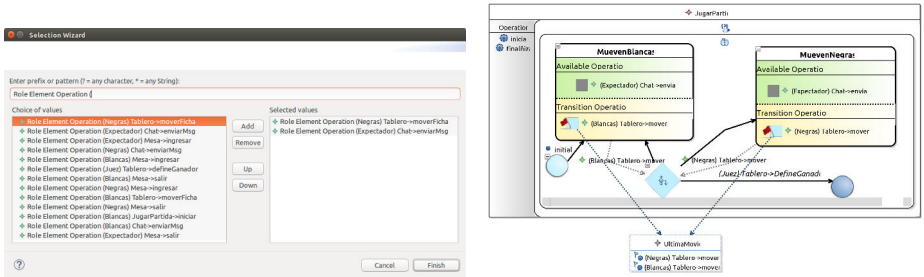
En la figura 8 se presenta el diseño de un proceso colaborativo que describe un campeonato de Ajedrez. Es una versión simplificada de un campeonato, que está compuesto de dos partidas semifinales y una partida final. Se muestra en la figura 8, que las dos semifinales tienen un indicación que la actividad produce un dato de salida, que es el jugador que ganó esa partida. En la actividad final, se reciben los ganadores de las semifinales y luego de la partida final se definirá qué jugador fue el ganador del campeonato. Este ejemplo de campeonato puede extenderse a versiones de 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. Las distintas actividades pueden estar asociadas a eventos que originan la actualización de la información de awareness que se muestra en algún otro elemento colaborativos. En el ejemplo, en la final se muestra un awareness cuando alguno de los jugadores provoca un jaque al rey.

En la figura 7, se definió que en el tablero se mostrará el awareness que el rey está en jaque. En la figura 8, se está modelando/diseñando el evento que actualiza el mismo awareness. Más adelante se explicará cómo estos diseños pueden llevarse a una implementación.

Otros aspectos dinámicos que se diseñan con CSSL son los protocolos, que definen el comportamiento dentro de una actividad colaborativa. En el ejemplo, que se muestra en la figura 7 el modelado de la única actividad: *JugarPartida*. El protocolo en las actividades, está compuesto por estados que se definen a partir de las operaciones que los roles pueden ejecutar en cada momento.

En la figura 9, se muestra una versión simplificada de la actividad *JugarPartida*, que cuenta con dos estados: *MuevenBlancas*, donde el jugador el jugador



(a) Operaciones en los estados (b) Protocolo de JugarPartida

Figura 9: Protocolos de actividades colaborativas

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 pueden enviar mensajes en el chat en los ambos 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 de 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 9a, 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 mostró en la figura 5c.

En la figura 9b, se muestra también 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.

Los modelos expresados en el lenguaje CSSL son independientes de la plataforma de implementación, ya que no hacen referencia a los sistemas operativos, los lenguajes de programación, el hardware, la topología de red, etc. Partiendo de estos modelos, en la próxima sección se presenta una transformación, que permite obtener código ejecutable a partir de los modelos construidos con el lenguaje CSSL.

4 Derivación de Código

En el contexto de MDD, CSSL es un PIM (Platform Independent Model). En este apartado se presenta una transformación desarrollada con una herramienta de eclipse, llamada Acceleo [1], que toma conceptos del lenguaje CSSL y obtiene un modelo específico para una plataforma particular, un PSM (Platform Specific Model). Para analizar una transformación a un modelo ejecutable se eligió una

arquitectura de transformación que define para cada elemento del Metamodelo CSSL, un elemento en el metamodelo destino, como se muestra en la figura 10(A).

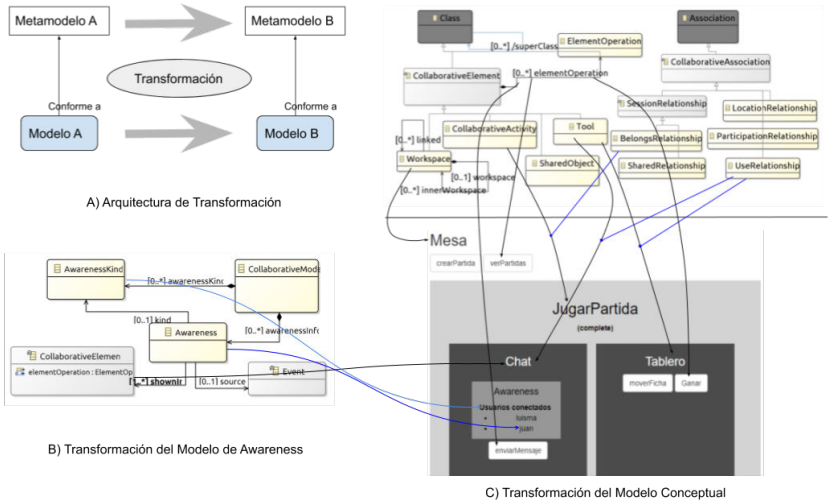


Figura 10: Transformación de Modelos

A partir del mapeo elaborado se eligió una plataforma destino que permita implementar los conceptos mencionados en la transformación (Aplicación, Componente, Boton, Proceso, Clases (Tipos), Permisos, Role, Usuario, Componente Contenedor). Se eligió una arquitectura Cliente-Servidor (Web), por lo tanto para cada elemento del lenguaje se traduce a uno o varios componentes de la implementación, que se elaboran tanto para el cliente como para 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 es el caso de 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 realiza a través de una API REST que se define 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.

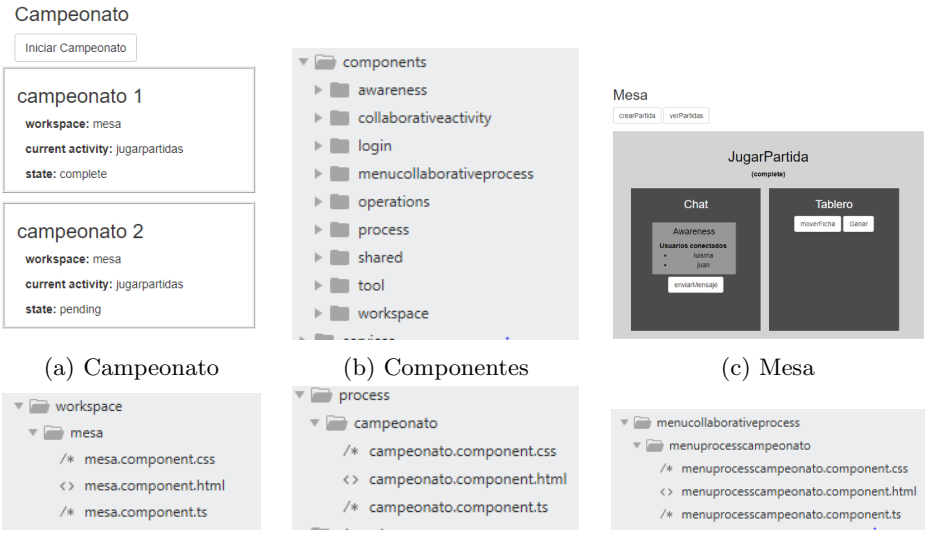
La elección final recayó en TypeScript, que es un lenguaje de programación basado en JavaScript, con la ventaja que es un lenguaje tipado que permite crear estructuras de clases y puede ejecutarse tanto en el cliente como en el servidor. En definitiva la transformación consiste en tomar una instancia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

4.1 Transformación del PIM al PSM

Se transforma entonces cada uno de los elementos colaborativos que aparecen en el modelo de entrada, en componentes de Angular (Awareness, Collaborativeactivity, Operations, Process, AwarenessKind, Tool y Workspace), como se muestra en las figuras 10(B y C) y otras dos componentes específicas que manejan el login y el menú como se muestra en la figura 11b.

Se describe a continuación la transformación de los elementos principales del lenguaje.



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

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

Figura 11: Estructura de la plataforma de implementación

Transformación de los Procesos Colaborativos Los procesos son los componentes que se destacan del modelo. Nos determinan el orden en que las actividades colaborativas que realizan los usuarios se llevan a cabo. Cada proceso va

a ser un componente 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 11e para el proceso campeonato. Para cada proceso también se crea un componente Angular (fig. 11f) que se utiliza para mostrar las opciones en el menú para todos los procesos que el usuario tenga acceso.

Al abrir el componente campeonato, se muestra una página donde se listan todos los campeonatos donde el usuario tiene participación (figura 11a). 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 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 11a. A medida que distintos usuarios van creando campeonatos se van agrupando para organizar los miembros de las partidas de los campeonatos.

Transformación del Workspace Dentro del modelo, el workspace es el lugar donde ocurren las actividades. El elemento workspace también será una componente Angular que funciona como contenedor de otros componentes, Tools y CollaborativeActivity. La implementación de la transformación se muestra en la figura 11c, los botones muestran las operaciones, que los roles pueden ejecutar, de las actividades, las herramientas y los espacios.

Transformación de la actividad Las actividades son las tareas que realizan los usuarios dentro de los procesos y están contenidas en un workspace. Las actividades pasan por distintos estados, cuando se crean se iniciarán en estado *pendiente* y significa que faltan usuarios para arrancar, cuando están todos los usuarios la actividad podrá arrancar como fue modelado en el diseño.

5 Conclusiones

Los sistemas colaborativos tienen aspectos 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 del estado de la colaboración mientras ésta se lleva a cabo. Se comenta en este trabajo un lenguaje específico CSSL v2.0 [7] para describir sistemas colaborativos. El lenguaje define los conceptos principales de los sistemas colaborativos, especialmente los procesos colaborativos, protocolos y awareness. Se presenta una sintaxis concreta a través de un conjunto de editores donde se crean modelos de sistemas colaborativos instanciando los conceptos del lenguaje. Además de los aspectos estáticos se modelan aspectos dinámicos como procesos colaborativos, protocolos y awareness.

Adhiriendo a la metodología MDD, 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 (Model to Text) implementada con Acceleo [1], que permite obtener una versión Web a partir de los modelos colaborativos. Se utiliza una tecnología liviana y flexible basada en javascript (Express, Node.js, Angular, MongoDB, etc.), para obtener versiones ejecutables. Asimismo se comenta cómo se mapean los conceptos del sistema colaborativo en componentes de la aplicación, tanto en el server 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 proceso colaborativos y se maneja el estado de las instancias de cada uno de ellos. Finalmente se muestra el funcionamiento del awareness implementado con la tecnología de WebSockets.

Los resultados de este artículo 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 [2], 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 con otras relacionadas con UML (editores, mapeadores, etc.).

Finalmente 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.

Para continuar el trabajo de investigación se está estudiando la aplicación o 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.

Referencias

1. Acceleo — Home, https://www.eclipse.org/acceleo/
2. Eclipse - an open development platform, http://www.eclipse.org/
3. Sirius - The easiest way to get your own Modeling Tool, https://www.eclipse.org/sirius/
4. UML 2.5 Diagrams Overview, https://www.uml-diagrams.org/uml-25-diagrams.html

5. Belkadi, F., Bonjour, E., Camargo, M., Troussier, N., Eynard, B.: A situation model to support awareness in collaborative design (2013). https://doi.org/10.1016/j.ijhcs.2012.03.002
6. Bibbo, L.M., Giandini, R., Pons, C.: 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) (Ciencias Informáticas), 111–122 (2016)
7. Bibbo, L.M., Giandini, R., Pons, C.: DSL for Collaborative Systems with Awareness. SLISW - Simposio Latinoamericano de Ingeniería de Software; XLIII CLEI - Conferencia Latinoamericana de Informática (2017)
8. Brambilla, M., Cabot, J., Wimmer, M.: Model-Driven Software Engineering in Practice. Synthesis Lectures on Software Engineering **1**(1), 1–182 (9 2012). https://doi.org/10.2200/S00441ED1V01Y201208SWE001, http://www.morganclaypool.com/doi/abs/10.2200/S00441ED1V01Y201208SWE001
9. Dourish, P., Bellotti, V.: Awareness and coordination in shared workspaces. Proceedings of the 1992 ACM conference on Computer-supported cooperative work - CSCW '92 pp. 107–114 (1992). https://doi.org/10.1145/143457.143468, http://portal.acm.org/citation.cfm?doid=143457.143468
10. Ellis, C.A., Gibbs, S.J., Rein, G.: Groupware: some issues and experiences. Communications of the ACM **34**(1), 39–58 (1 1991). https://doi.org/10.1145/99977.99987, http://portal.acm.org/citation.cfm?doid=99977.99987
11. Gallardo, J., Molina, A.I., Bravo, C., Redondo, M.A., Collazos, C.A.: 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). https://doi.org/10.1016/j.eswa.2010.05.005
12. Grudin, J.: Computer Supported Cooperative Work: History and Focus. Computer (1994)
13. Pons, C., Giandini, R.S., Pérez, G.: Desarrollo de software dirigido por modelos. Editorial de la Universidad Nacional de La Plata (EDULP) / McGraw-Hill Educación (2010), http://sedici.unlp.edu.ar/handle/10915/26667
14. Teruel, M.A., Navarro, E., López-Jaquero, V., Montero, F., Jaen, J., González, P.: Analyzing the understandability of Requirements Engineering languages for CSCW systems: A family of experiments. Information and Software Technology (2012). https://doi.org/10.1016/j.infsof.2012.06.001