DSL for Collaborative Systems with Awareness

Luis M. Bibbo, Roxana Giandini, and Claudia Pons

***Abstract*— Domain Specific Languages (DSLs) are high-level languages defined for combining expressiveness and simplicity by means of linguistic constructs which are close to the problem domain but independent of the complexities inherent to the underlying software implementations. This article presents the CSSL v2.0 language that allows defining in precise, concise and friendly manner the abstract concepts of collaborative systems. Specially, the language makes available the concepts of awareness and collaborative processes. The language is independent of both the framework and the development tools and allows the application of the MDD approach to the development of such systems. The CSSL v2.0 language was designed as a UML extension using the metamodeling mechanism and was implemented with open source tools on the Eclipse platform. It provides improvements on previous proposals by enabling more complete and complex specifications of collaborative situations.**

***Index Terms*— Collaborative software, Design tools, Domain Specific Languages, Software design, Software engineering.**

I. Introducción

Cuando nos referimos a sistemas colaborativos estamos pensamos 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: "Sistemas de computadoras que proveen una interfaz a un entorno compartido y que soportan a un grupo de usuarios que tienen un objetivo común". Esta tecnología brinda un equilibrio entre el trabajo individual de los participantes y la colaboración 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 [2]. Esto se refleja en procesos que determinan en qué orden se llevan adelante las actividades colaborativas 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.

El awareness [3] es la percepción o conocimiento del grupo y de las actividades que realizan los usuarios en el sistema compartido. Construir sistemas colaborativos es una tarea compleja ya que el software atiende a las necesidades de usuarios remotos, con requerimientos que cambian dinámicamente con muchos desafíos tecnológicos. Asimismo la incorporación de diversos dispositivos como computadoras personales, teléfonos celulares, dispositivos de realidad virtual y otros dispositivos móviles.

Por otro lado, el problema de construir 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, 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 [4]. 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.

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.

Trabajo enviado el 29-04-2017.

Luis Mariano Bibbo del laboratorio Lifia - Facultad de Informática de la Universidad Nacional de La Plata, La Plata (1900), Buenos Aires, Argentina. (e-mail: lmbibbo@lifia.info.unlp.edu.ar).

Roxana Gandini del laboratorio Lifia - Facultad de Informática de la Universidad Nacional de La Plata, La Plata (1900), Buenos Aires, Argentina. (e-mail: giandini@lifia.info.unlp.edu.ar) e investigadora asociada a CICPBA, Comisión de Investigaciones Científicas de la provincia de Buenos Aires, Argentina.

Claudia Pons del laboratorio Lifia - Facultad de Informática de la Universidad Nacional de La Plata, La Plata (1900), Buenos Aires, Argentina. (e-mail: cpons@lifia.info.unlp.edu.ar), investigadora adjunta de CICPBA, Comisión de Investigaciones Científicas de la provincia de Buenos Aires, Argentina y Directora del Centro de Altos Estudios en Tecnología Informática (CAETI) de la Universidad Abierta Interamericana (UAI), Buenos Aires, Argentina.

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.

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 encontramos a "Class", "Activity", "State Machine", etc. que luego aparecerán instanciadas en un modelo UML. Teniendo en cuenta que podemos extender el el metamodelo de UML, nos basamos en él para diseñar un metamodelo para Sistemas Colaborativos con Awareness. Nos proponemos entonces en este trabajo, construir un lenguaje específico para el dominio de los sistemas colaborativos denominado CSSL v2.0 que permita expresar las particularidades que tienen estos sistemas. Este lenguaje resulta el punto de partida de otras investigaciones y desarrollos donde se obtendrá código a partir de los modelos utilizando, por ejemplo, transformaciones de modelo a texto.

El trabajo se estructura de la siguiente manera; Sección II comentamos las necesidades y motivaciones para construir un DSL para sistemas colaborativos. Sección III analizamos los trabajos relacionados en el área. Sección IV presentamos el lenguaje CSSL v2.0 a través de un caso de estudio. Sección V elaboramos algunas conclusiones y las tareas que dan continuidad al trabajo.

II. Motivación

Desarrollar aplicaciones colaborativas es una tarea compleja ya que estos sistemas mantienen a un conjunto de actores/usuarios interactuando con un entorno compartido [13], [14]. Específicamente, es complicado modelar la funcionalidad de awareness que informa lo que está pasando con los otros usuarios mientras colaboran. Contar con métodos, procesos o lenguajes que guíen la construcción de 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 [5] presentamos el lenguaje CSSL que permite a los desarrolladores construir una conceptualización de un sistema colaborativo de una manera sencilla y amigable. CSSL fue definido e implementado usando artefactos estándares de MDD (por ejemplo MOF (meta-object facility) framework de modelado en Eclipse). Permite al diseñador relacionar los conceptos habituales de los sistemas colaborativos a través de asociaciones y brinda la posibilidad de especificar protocolos de interacción en las actividades colaborativas. Pero sin embargo, no incluye la posibilidad de especificar awareness asociada a los conceptos principales y no se permite especificar procesos colaborativos para organizar las actividades colaborativas que realizan los usuarios.

Por otro lado en [6] realizamos una revisión de la literatura académica sobre sistemas colaborativos con awareness y elaboramos un conjunto de requisitos que un metamodelo para sistemas colaborativos con awareness debería incluir. Dichos requisitos son:

Requisito 1: Contar con un modelo conceptual que contenga los conceptos que intervienen en los 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, etc.)

Requisito 3: Contar con un modelo de proceso colaborativo como un conjunto de actividades colaborativas que desarrollan los usuarios.

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

Requisito 5: Permitir modelar los distintos estados por lo que pasa una sesión de interacción (protocolos de interacción)

Requisito 6: Incorporar distintos tipos de awareness asociados a los estados por lo que pasa una sesión.

Requisito: 7: utilización de estándares en la construcción de los modelos que intervienen.

En consecuencia vemos necesario extender el lenguaje CSSL para incluir el concepto de awareness que surge del requisito 2. También incorporar el concepto de proceso colaborativo y permitir agregar awareness a las etapas del proceso lo que cubre el requisito 3 y 4. Finalmente actualizar las herramientas, teniendo en cuenta que las mismas respeten los estándares de MDD que responde al requisito 7. El objetivo en definitiva es el de diseñar e implementar el CSSL v2.0. Específicamente definiremos formalmente su sintaxis a través de meta-modelos soportados por editores gráficos que permitan a los diseñadores instanciar las construcciones del lenguaje para generar modelos de sistemas colaborativos. Mientras que la definición de su semántica se realizará en próximas etapas a través de transformaciones de estos modelos al código ejecutable del sistema colaborativo modelado.

III. Trabajos Relacionados

Como vimos en la revisión [6] son pocos los trabajos que presentan modelos abstractos para que los diseñadores de software incluyan el concepto de awareness en sistemas colaborativos. Encontramos en [7] la presentación de lenguaje CSRML que utiliza los conceptos de los sistemas colaborativos pero no tiene posibilidades de especificar procesos colaborativos. Al no incluir procesos colaborativos, no se puede brindar información de awareness relacionada con ellos. Por ejemplo, cuán avanzado está el proceso, cuántos están activos o cuál proceso está por comenzar. Asimismo la herramienta CSRML tool 2012 [8] 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.

En [9] separa la problemática del awareness dentro de los sistemas colaborativos en dos aspectos. Por un lado aparece el concepto de awareness con dos subclases Workspace 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. Se determina qué acción activa o dispara el awareness. 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. Dicho de otra manera por un lado tenemos representado el "Qué tipo" de awareness se informa y por otro el "Qué evento/acción activa" el awareness que se informa.

En [10] se presenta una revisión de trabajos científicos que brindan distintas clasificaciones de awareness. Comenta distintas definiciones y concluye el análisis puntualizando características del awareness

El awareness es un concepto multifacético. Distintos investigadores han propuesto distintas clasificaciones de awareness (social awareness, mutual awareness, task awareness, workspace awareness, etc.)

Un sistema colaborativo que quiera soportar awareness deberá responder a un conjunto de requerimientos relacionados con una lista de preguntas que se detallan a continuación.

El trabajo considera al "situation/context awareness" como la suma de tres tipos de awareness identificados (social awareness, task awareness, workspace awareness). Luego de definir Context (Situation) Awareness el autor analiza distintos trabajos que modelan requerimientos para soportar la representación compartida del awareness. Las investigaciones analizadas brindan un conjunto de conceptos que frecuentemente han sido usados en modelos focalizados en awareness. A continuación se enumeran los conceptos y se los relaciona con nuestro metamodelo.

- Context element: Es modelado como un conjunto de elementos. En nuestro caso, se puede representar como un CollaborativeElement
- Task and Activity: Esto describe que se espera que se haga o que es lo que se está realizando para completar una interacción. En nuestro caso tenemos las operaciones que realizan los roles que se consideran tareas que pueden disparar awareness.
- Resource: Describe un elemento del contexto. Este elemento es usado durante o para completar una interacción. En nuestro caso es un SharedObject
- 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 Modelling). En nuestro metamodelo la interacción simple se da con nuestro concepto de actividad colaborativa y cuando involucra un proceso se lo diseña como un Proceso colaborativo.
- Role: Describe un conjunto de operaciones que un usuario puede realizar en un determinado contexto. De alguna manera es el papel que le toca jugar en las actividades colaborativas. En algunas situaciones puede Por ejemplo el Líder podrá participar de las reuniones y podrá asignar tareas a otros roles. En nuestro metamodelo el CollaborativeRole por ser sub clase de uml::Class se le pueden agregar un conjunto de operaciones y que definen lo que un rol puede hacer.

Los trabajos en general utilizan los mismos elementos conceptuales que se encuentran en los sistemas colaborativos. Estos son "Shared Object", "Tool", "Collaborative Activity", "Workspace", "User/Role", "Group". Estos elementos también tienen relación con el concepto de awareness. Como vimos en [10], 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?. También la mayoría de los trabajos incorporan el concepto de "Task", que representa las tareas colaborativas que tienen que realizar los usuarios. Sin embargo, los trabajos no modelan procesos que incluyan esas tareas y que permitan darle un orden lógico a las mismas.

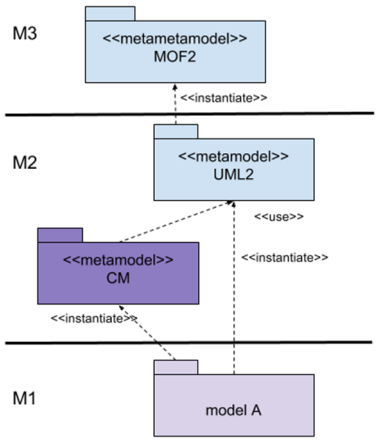

Figura 1: Capas de la OMG. El modelo M1 instancia metaclases de UML2 y de nuestro Metamodelo CM (Collaborative Metamodel)

IV. CSSL v2.0

A partir de nuestro trabajo previo [5], presentamos una versión de nuestro lenguaje CSSL v2.0. En esta versión, optamos por incluir el metamodelo de UML a nuestro lenguaje. Vemos en la Figura 1 que los modelos de los sistemas colaborativos pueden incluir todas las instancias de las metaclases de UML más las instancias de las metaclases que se definen en nuestro metamodelo que llamamos CM (Collaborative Metamodel). El metamodelo CM incluye los conceptos principales de los sistemas colaborativos con awareness: "Shared Object", "Tool", "Collaborative Activity", "Workspace", "User", "Role", "Group" lo cual nos permite modelar las relaciones básicas entre los elementos que intervienen en los sistemas colaborativos.

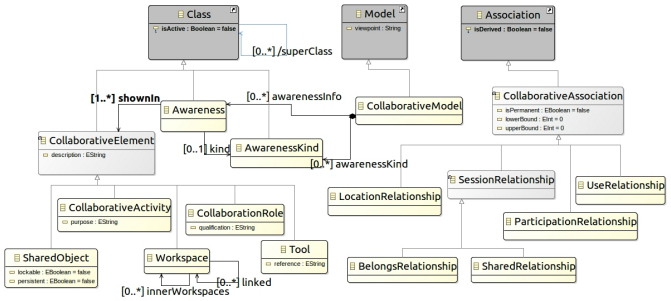

Figura 2: Submodelo de elementos y asociaciones colaborativas

En la Figura 2 puede verse en gris más oscuro las metaclases de UML distinguiéndolas de las otras metaclases que pertenecen a nuestro metamodelo. La primera metaclase que vamos a comentar es la que representa un modelo colaborativo: "CollaborativeModel" y es la que agrupa todos los elementos que están incluidos en un modelo de un sistema colaborativo. Esta metaclase es subclase de Model de UML. Model a su vez es subclase de Package y permite organizar y estructurar las clases que pertenecen al modelo. Las clases se dice que son contenidas o pertenecen a un modelo a través de packagedElement. Una instancia de la metaclase CollaborativeModel contendrá los elementos del sistema colaborativo y actuará como la raíz del modelo. A partir de esta instancia podremos acceder a todas las clases del modelo colaborativo.

Un modelo colaborativo cuenta principalmente con dos clases que son CollaborativeElement y CollaborativeAssociation. Estas clases son subclases de Class y Association. La metaclase Class es usada para describir objetos que tienen la misma estructura, comportamiento y las mismas restricciones y semántica.

A. *Instanciar el Modelo*

OMG (Object Management Group) define una arquitectura basada en cuatro niveles de abstracción que van a permitir distinguir entre los distintos niveles conceptuales que intervienen en el modelado de un sistema. Esos cuatro niveles son:

- M3. Meta metamodelo
- M2. Metamodelo
- M1. Modelo del usuario
- M0. Instancias en tiempo de ejecución

La responsabilidad primaria de la capa de meta-metamodelo es definir el lenguaje para especificar un metamodelo. Esta capa es conocida como M3, MOF (Meta Object Facility) [11] es un ejemplo de un meta-metamodelo que por lo general es más compacto que un metamodelo y a menudo define varios metamodelos.

Un metamodelo es una instancia de un meta-metamodelo y significa que cada elemento del metamodelo es una instancia de un elemento del meta-metamodelo. La responsabilidad primaria de la capa del metamodelo es definir un lenguaje para especificar modelos. Esta capa es conocida como M2; UML (Unified Modelling Language) y OCL [12] (Object Constraint Language) son ejemplos de metamodelos. En general estos son más detallados que los meta-metamodelos que los describen, sobre todo cuando ellos definen semántica dinámica.

Un modelo es una instancia de un metamodelo. La responsabilidad de esta capa es definir un lenguaje que describa los dominios semánticos, es decir, para permitirles a los usuarios modelar una variedad de dominios diferentes, como procesos, requerimientos, etc. Esta capa es conocida como M1. La instanciación del metamodelo (es decir la obtención de un modelo conocido como el Nivel M1 de los niveles de abstracción de la OMG) es una forma de obtener la sintaxis concreta del lenguaje.

B. *Mis Editores*

Nuestro lenguaje brinda una sintaxis abstracta de los sistemas colaborativos. Como fue construido a partir de UML, es un lenguaje formalizado. Utilizando el metamodelo podemos instanciar la metaclase CollaborativeModel que es subclase de Model de UML. Una instancia de CollaborativeModel contiene a los elementos que intervienen en el sistema. Así podemos crear CollaborativeRoles, CollaborativeActivity, Workspace, etc.

Crear modelos a partir de instanciar las clases del metamodelo resulta una trabajo arduo y poco amigable. En la

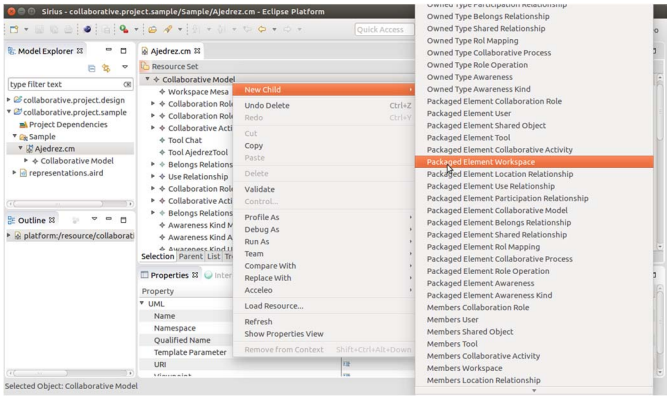

Figura 3: Instanciado el metamodelo CM usando los editores estándares de UML

Figura 3 vemos como sería instanciar el metamodelo a partir de su sintaxis abstracta. Primero se crea una instancia de CollaborativeModel y a partir de ella se van instanciando sus elementos de ese modelo como hijos. En el ejemplo vemos que se está creando una instancia de Workspace. En este caso la sintaxis abstracta y concreta sería la misma. Si usamos esta forma de instanciar el metamodelo veremos 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 que suelen ser más expresivas para los diseñadores. En nuestro caso tenemos una sintaxis para cada uno de los elementos de los sistemas colaborativos. Elegimos una representación de conceptos y relaciones para representar la estructura del sistema; una representación de tabla para ver las operaciones que puede tener cada rol, un diagrama de proceso para diseñar las actividades colaborativas y un diagrama similar a las

máquinas de estado para modelar el protocolo de una actividad colaborativa. Vemos en la Figura 4 los distintos editores de sintaxis concreta que nos servirán para instanciar el metamodelo. En el ejemplo concreto vemos como los distintos editores trabajan con el mismo modelo que es instancia de CollaborativeModel.

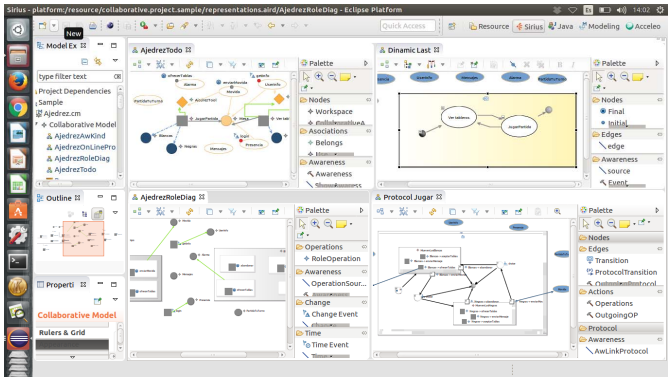

Figura 4: Editores creados para nuestro metamodelo.

Usando Sirius[1] desarrollamos un conjunto de editores gráficos basados en nuestro metamodelo. A partir de ellos podemos diseñar distintos aspectos de nuestros modelos de sistemas colaborativos.

En la Figura 5 vemos las paletas de acciones que se utilizan con los editores asociados al lenguaje CSSL v2.0.

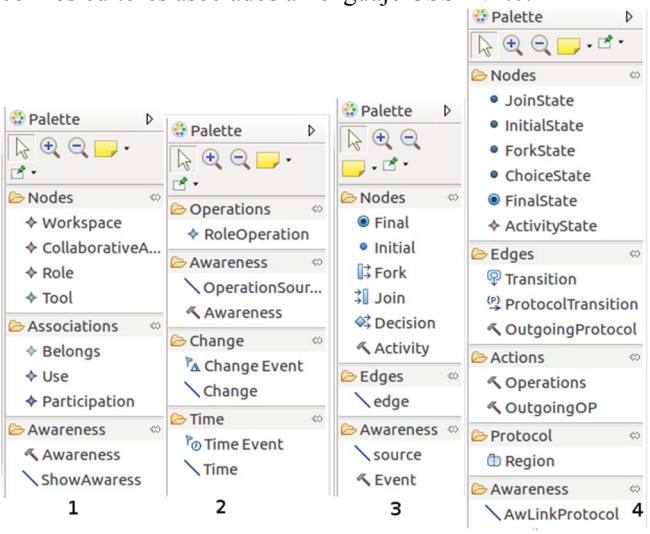

Figura 5: Paletas de acciones de los editores de CSSL v2.0

Con la primera paleta se agregan los conceptos y sus asociaciones. Podemos agregar elementos colaborativos como "Workspace", "Tool", "Role" y "CollaborativeActivity". También asociarlos usando las asociaciones de pertenencia, uso o participación. El resultado puede verse en la Figura 6. Usando el mismo editor puede incorporarse el awareness y vincularlo a los elementos del diseño como puede verse en la Figura 7.

Con la paleta 2 se puede trabajar con los roles y las operaciones que pueden realizar. Las operaciones pueden también ser origen de información de awareness. Esto significa decir que cuando un rol ejecuta una operación se mostrará en algún otro elemento alguna información de awareness. En nuestro caso, ocurre por ejemplo cuando el rol *Blancas* ejecuta una movida, esto origina el awareness que se muestra en la mesa o en el tablero de su oponente como se ve en la Figura 7.

Con la paleta 3 se trabaja con los procesos colaborativos. En la Figura 10 se muestra que se puede crear un proceso donde las actividades colaborativas son los nodos que intervienen en el proceso. Además se utilizan otros nodos (ControlNodes) que manejan el flujo del proceso como se muestra en la Figura 9. Ellos se usan para manejar el flujo de los tokens entre los nodos y representan un conjunto de nodos concretos como ser InitialNode, FinalNode, ForkNode, MergeNode, etc. El InitialNode es el primero en estar disponible al iniciar la actividad. El ForkNode recibe un eje de entrada y envía un token a cada eje de salida. Al proceso diseñado se lo relaciona con el awareness. Los nodos del proceso tienen un conjunto de eventos que pueden ser origen de información de awareness (ver Figura 10). En nuestro caso podemos decir que al finalizar una jugada se dispare un evento que informe una Alarma que se muestra en el tablero, como puede verse en la Figura 11.

C. *Caso de Estudio*

A partir de este modelo vamos a modelar un juego colaborativo, por ejemplo, un juego de tablero como el Ajedrez. A partir de una plataforma web (https://www.chess.com/) que permite jugar al ajedrez vamos a analizar los elementos que intervienen en el sistema.

En la imagen se ven las partidas de un usuario del sistema donde están resaltadas aquellas en las que el usuario tiene que jugar. Al ingresar en una de ellas el usuario puede realizar una movida.

A partir de los conceptos que aparecen en nuestro modelo y haciendo una ingeniería inversa podemos asumir que en los modelos aparecen los siguientes elementos:

- Roles: *Blancas* (Jugador que juega con fichas blancas), *Negras* (Jugador que juega con fichas negras, *Jugador* (Para un jugador en general)
- Herramientas colaborativas: *Tablero de Ajedrez*, *Chat*, *Visor de Partidas*
- Como actividad colaborativa tenemos: *Ver Tableros*, *Jugar Partida*
- Finalmente asumamos que estas actividades colaborativos se realizan en una solo espacio que lo llamaremos *Mesa*.

A partir de estos conceptos podemos iniciar un diseño usando un editor que nos permite agregar espacios, actividades colaborativas, herramientas y roles y relacionarlos entre sí. También vemos que la representación de cada uno de elementos del sistema puede ser diferente (incluso se podrían agregar gráficos) y son muy fácil de identificar por los diseñadores. El diseño de una versión simplificada queda de la siguiente manera:

[1] http://www.eclipse.org/sirius/: Sirius es un proyecto de Eclipse que permite crear herramientas de trabajo de modelado gráfico aprovechando las tecnologías de Eclipse Modeling, incluyendo EMF y GMF

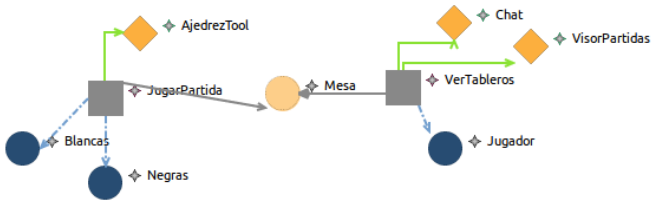

Figura 6: Representación del espacio colaborativo "*Mesa*"

Para modelar el awareness vemos en la Figura 2 las clases del lenguaje que se utilizan para modelar el awareness y que se relacionan con los otros elementos del sistema. Por un lado tenemos AwarenessKind que representa distintos tipos de awareness que soporta el sistema. En el metamodelo tenemos que el CollaborativeModel maneja un conjunto de tipos de awareness. Entre los tipos de awareness habituales encontramos la presencia, la ubicación, la actividad de los usuarios, etc. Por otro lado vemos en la Figura 2 la metaclase Awareness hace referencia a un conjunto de objetos con la relación "*shownIn*" donde se visualizará el awareness. Este conjunto representa los elementos colaborativos que recibirán la información de awareness y la presentarán en la interfaz. Tenemos en nuestro metamodelo que un mismo awareness puede mostrarse en varios lugares ya que la relación shownIn es de [1-*]. Por ejemplo la presencia del usuario se puede mostrar en un espacio colaborativo y en una herramienta como puede ser el chat.

A su vez la metaclase Awareness hace referencia a un evento con la relación "source" que representa el evento que origina el awareness. Los eventos en UML son algo que ocurre en un momento determinado. En la jerarquía de Event de UML aparece TimeEvent que es un evento que ocurre en un momento determinado, ChangeEvent que ocurre cuando se produce un cambio en algún valor establecido o un MessageEvent que ocurre cuando un mensaje es recibido.

En definitiva lo que muestra esta porción del modelo es que ante la ocurrencia de un evento se activa cierta información de awareness que será mostrada en uno o varios elementos colaborativos. Con este modelo se puede expresar por ejemplo que ante una operación que realiza alguno de los roles (cada operación tiene asociado una subclase de Event que se llama CallEvent), se muestra alguna información (Awareness) en algún elemento colaborativo. En el siguiente ejemplo distintos awareness que se relacionan con el modelo del Ajedrez

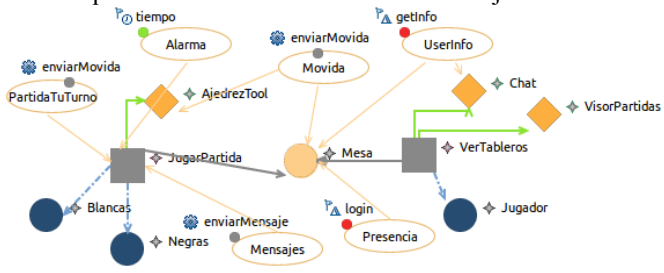

Figura 7: Representación del espacio con Awareness

Los sistemas colaborativos combinan distintas actividades colaborativas que desarrollan los usuarios. Los procesos colaborativos definen qué actividades se tienen que llevar a cabo en cada momento. Por ejemplo definen si hay actividades que pueden ejecutarse en simultáneo o que tareas deben sincronizarse o esperar que alguna otra termine. Para modelar correctamente un proceso colaborativo elegimos extender el modelo de actividades que se usa en UML. La clase CollaborativeProcess, como todas las clases de en UML tiene un comportamiento[2]. En nuestro modelo el comportamiento de un proceso colaborativo es una actividad (Activity) y se define el proceso en donde los usuarios/roles pasan por las distintas sesiones para llegar a un objetivo grupal. En definitiva el proceso se representa como un grafo nodos (ActivityNodes) y ejes (ActivityEdge). En UML ActivityNode se usa para modelar una etapa de la actividad/proceso. Para que un ActivityNode se ejecute se tiene que cumplir las condiciones que satisfagan el ActivityEdge de entrada.

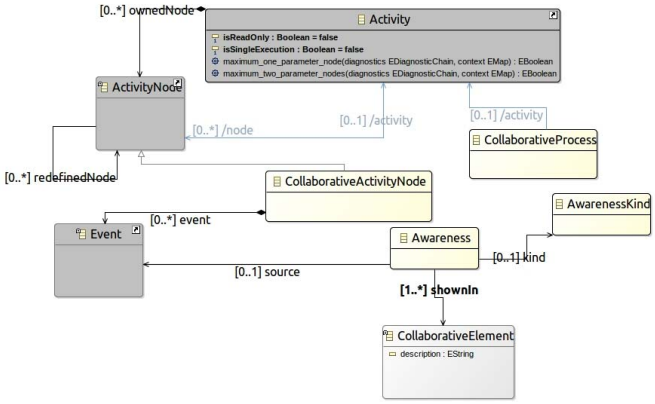

Figura 8: Submodelo de Proceso Colaborativo con Awareness

En UML se definen diversos nodos a partir de ActivityNode. En particular encontramos los ControlNodes, que se muestran en la Figura 9, y que son un tipo de nodos usados para controlar el flujo de tokens entre los nodos de una actividad. Entre los ControlNodes encontramos a InitialNodes y FinalNodes, que controlan el inicio y el fin de una actividad. También aparecen ForkNodes, JoinNodes, MergeNodes y DesisionNodes que permiten trabajar con el flujo de tokens.

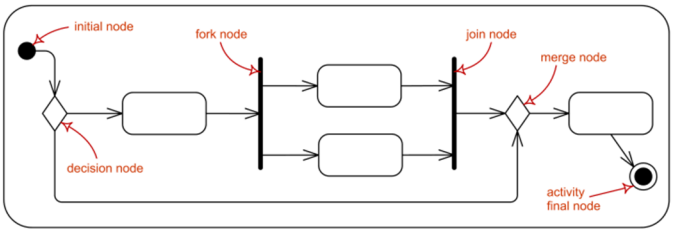

Figura 9: http://www.uml-diagrams.org/activity-diagrams-controls.html

Entonces para nuestro modelo el proceso colaborativo estará compuesto por un encadenamiento de actividades colaborativas. Para poder utilizar a una "ColaborativeActivity" como un nodo de una actividad, tenemos que hacer que la ColaborativeActivity sea subclase de ActivityNode.

Siguiendo con nuestro ejemplo del portal de Ajedrez identificamos las actividades colaborativas de *Jugar Partidas*

[2] En UML la clase Class es subclase de BehavioredClassifier. Eso significa que las clases en uml tienen una relación con la clases Behavior que se llama ownedBehavior. Los comportamientos pueden ser entre otros comportamientos Máquinas de estados (StateMachine) o Actividades (Activitiy)

y *Ver Tableros*. Definimos un proceso colaborativo que indica que hay que realizar varias partidas para poder jugar algún campeonato. En cualquier momento se puede ir a ver los tableros que tiene el jugador. La sintaxis concreta que definimos para expresar a los procesos es similar a los diagramas de procesos como los que se muestran en la Figura 9. La diferencia es que los nodos representan actividades colaborativas y que fueron modelados con la metaclase CollaborativeActivityNode. Como vemos en la Figura 8 esta metaclase tiene un conjunto de eventos que se activan al iniciar o al finalizar la actividad. El diagrama del proceso colaborativo es el que se muestra en la Figura 10.

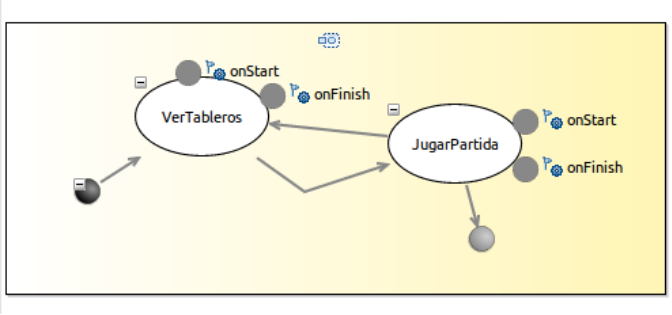

Figura 10: Proceso colaborativo

Los procesos colaborativos también están relacionados con información de Awareness. Como vimos en el submodelo de proceso colaborativo en la Figura 8, el awareness se muestra en objetos concretos del sistema como son los elementos colaborativos (Workspace, Tool, Collaborative Activity, Role) con la relación "shownIn". Pero estos awareness pueden originarse dentro de los procesos por ejemplo cuando se inicia o finaliza una actividad colaborativa o cuando se llegar al final de un proceso puede originar que cierta información de awareness sea mostrada en algún elemento colaborativo del sistema. En la Figura 11 se muestra como un evento que ocurre en una actividad colaborativa dentro de un proceso puede disparar un awareness.

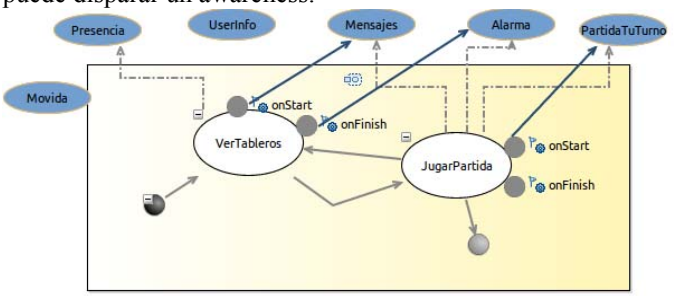

Figura 11: Proceso colaborativo con Awareness

Las actividades colaborativas pueden ser origen de distintos eventos. El diseñador puede definir que otros eventos tienen que ser atendidos en alguna actividad colaborativa. En la paleta 3 de la Figura 5 puede verse la operación de agregar eventos a una actividad colaborativa. En la codificación del sistema a partir de los modelos se deberá tener en cuenta el manejo de los eventos y la vinculación con el gestor de awareness.

Otra de las características relevantes de los sistemas colaborativos es que los distintos roles/usuarios interactúan entre sí accediendo a un modelo común. Esta interacción que se realiza dentro de una actividad colaborativa que es modelada con nuestro metamodelo a partir del comportamiento de la metaclases "CollaborativeActivity". Definimos que el comportamiento de una actividad colaborativa es una máquina de estado. Recordemos que "CollaborativeActivity" al ser subclase de "Class" puede tener un comportamiento que definimos que en este caso sea una instancia de "ProtocolStateMachine" ya que necesitamos que las operaciones que realizan los roles sean los que provoquen las transiciones.

En consecuencia decidimos extender las máquinas de estado de UML definiendo una subclase de "State" que represente un estado particular definido en la metaclase "CollaborativeActivityState". Como vemos en la Figura 12 este estado tiene relación con "RoleOperacion" donde se mantiene las operaciones de cada rol puede ejecutar dentro ese estado con la relación "asignedOperation".

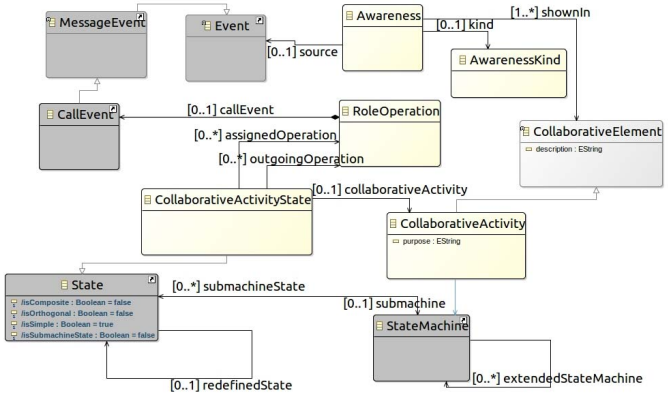

Figura 12: Submodelo de Protocolo de las actividades colaborativas

La metaclase de UML "ProtocolTransition" es usada en las máquinas de estado para conectar dos estados utilizando una operación como disparador de la transición. En nuestro metamodelo las instancias de "CollaborativeActivityState" se conectan entre si usando las operaciones que tienen asignadas y creando instancias de "ProtocolTransition". Además de estos estados se puede usar pseudo estados que se usan para agregar para completar la especificación de la máquina de estado. Estos pseudo estados son el estado inicial, final, fork, choice y join. Vemos en la Figura 5 en la paleta 4 que se pueden crear estados, agregar operaciones al estado y crear transiciones entre ellos.

En el ejemplo del portal de Ajedrez tenemos que la actividad colaborativa "*Jugar Partida*" tiene dos estados. En el primer estado es "*Juega las Blancas*", el rol "*Blancas*" puede "abandonar", "enviar un mensaje en el chat", "hacer una movida", "ofrecer tablas", etc. Solamente cuando hace una movida se cambia de estado y pasaremos al estado "*Juega las Negras*".

En este estado es el rol *Negras* es el que puede ejecutar las operaciones de abandonar, enviar un mensaje en el chat, hacer una movida, ofrecer tablas, etc. En el juego del ajedrez después de cada movida se chequea que haya algún escape para la pieza del rey adversario. En el pseudostate "choice" se verifica que haya jaque mate y en ese caso se moverá hacia el estado de fin del protocolo. En la Figura 13 de muestra la

sintaxis del protocolo de la actividad colaborativa "*Jugar Partida*".

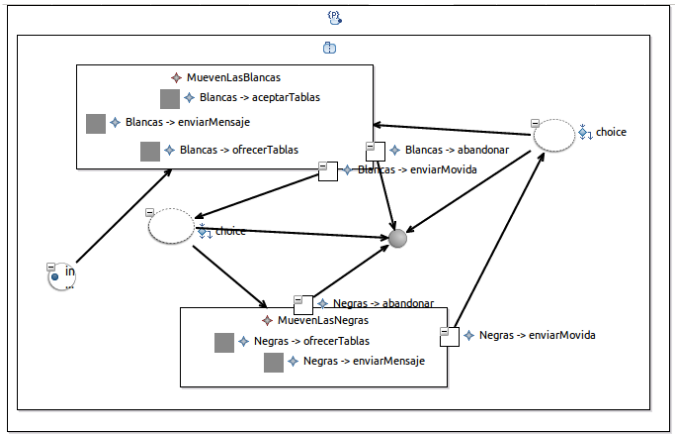

Figura 13: Protocolo colaborativo de la actividad "*JugarPartida*"

El metamodelo CM establece que las operaciones que ejecutan roles están relacionadas con la metaclase "RoleOperation" que tiene asociado un "CallEvent" como vemos en la Figura 12. Esta estructura nos permite usar las instancias de RoleOperation para definir que eventos pueden ser orígenes de alguna información de awareness. En definitiva lo que queremos expresar es que cuando un rol ejecuta alguna operación se mostrará el efecto en algún otro elemento colaborativo del sistema. En la Figura 14 puede verse por ejemplo, que cuando el rol *Blancas* ofrece tablas se dispara el awareness de alarma.

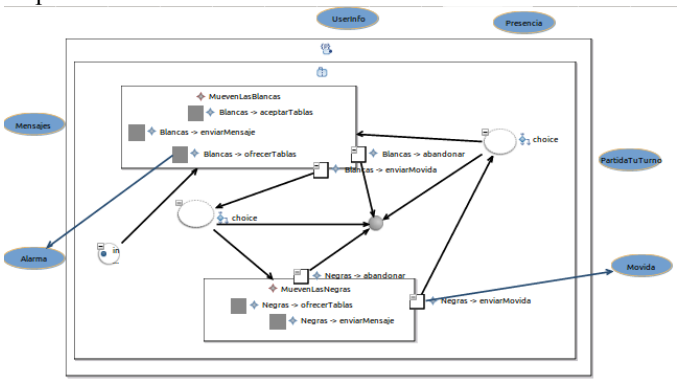

Figura 14: Protocolo colaborativo con información de Awareness

V. CONCLUSION

La construcción de sistemas colaborativos es una tarea muy compleja. Existen algunos modelos abstractos [7] [8] y [9] 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 [13] [14] 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. Muchas de estas carencias radican en que estas herramientas focalizan aspectos de implementación y no de modelado. En este contexto, consideramos al paradigma MDD que cambia el foco del desarrollo de software poniendo énfasis en el modelado y no en el desarrollo de código. Aplicar este paradigma al desarrollo de software colaborativo con awareness, es una propuesta prometedora para cubrir las deficiencias mencionadas.

Siguiendo los principios de MDD en el desarrollo de los sistemas colaborativos en este trabajo presentamos una versión actualizada del lenguaje específico de dominio CSSL [5], que denominamos CSSL v2.0. Nuestro lenguaje permite definir en forma precisa, concisa y amigable los conceptos abstractos de los sistemas, incluyendo el concepto de awareness y procesos colaborativos incluidos en la versión 2.0. El CSSL v2.0 fue definido como una extensión de UML usando el mecanismo de metamodelado y fue implementado con herramientas open source sobre la plataforma de Eclipse [15]. Los conceptos escritos en CSSL v2.0 son independientes del framework o herramientas de desarrollo. Nuestro lenguaje provee mejoras sobre las propuestas previas brindando definiciones más completas y complejas de situaciones colaborativas que pueden presentarse.

En la continuidad de este trabajo, se espera construir recursos que aborden los aspectos semánticos del lenguaje. Adhiriendo a la metodología MDD, se proveerá 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 están estudiando para este fin, distintas herramientas de transformación[3] como Acceleo, Jet, etc. que permitan obtener versiones ejecutables para distintos entornos y arquitecturas. Nuestros próximos pasos apuntan a definir como se mapean los conceptos del lenguaje a las implementaciones. Por ejemplo, los *workspaces* pueden transformarse en algún frame o ventana donde apareceran los conceptos relacionados con el *workspace*. También se podrá controlar en implementación los *roles* que podrán ingresar a ese *workspace*. Incluso se podrá implementar el flujo de *actividades* que involucran un *proceso colaborativo* definiendo que *rol* puede actuar en cada momento.

REFERENCES

[1] Ellis, C.A., Gibbs, S.J., Rein, G.L., Groupware: some issues and experiences, in: Communications of the ACM, 34(1) (1991).

[2] Gutwin, C., Greenberg, S., & Roseman, M. (1996). Workspace awareness in real-time distributed groupware: framework, widgets, and evaluation. In People and computers XI (Proceedings of the HCI'96).

[3] Gutwin, C., & Greenberg, S. (2002). A descriptive framework of workspace awareness for real-time groupware. CSCW Journal, 11, 411–446.

[4] Claudia Pons, Roxana Giandini y Gabriela Pérez. "Desarrollo de Software dirigido por Modelos - Conceptos teóricos y su aplicación práctica". Editorial EDULP & McGraw-Hill Educación, 2010. Volumen 1, 300 páginas. ISBN: 978-950-34-0630-4.

[5] Bibbo, Luis Mariano; García, Diego; Pons, Claudia; "A Domain Specific Language for the Development of Collaborative Systems," Chilean Computer Science Society, International Conference of the, pp. 3-12, 2008 International Conference of the Chilean Computer Science Society, 2008.

[6] Bibbo, Luis Mariano; Giandini, Roxana; Pons, Claudia. Sistemas Colaborativos con Awareness: Requisitos para su Modelado. 45 JAIIO. Presentado en el Argentine Symposium on Software Engineering (ASSE 2016). ISSN: 2451-7593.

[7] Teruel M.A., Navarro E., Lopez-Jaquero V., Montero F., Gonzalez P.:Analyzing the understandability of Requirements Engineering

[3] Model to text. https://eclipse.org/modeling/m2t/

languages for CSCW systems: A family of experiments. Information and Software Technology. (2012).
[8] Teruel M.A., Navarro E., Lopez-Jaquero V., Montero F., Gonzalez P.: CSRML Tool: A visual studio extension for modeling CSCW requirements. CEUR Workshop Proceedings. (2013).
[9] 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. Expert Systems with Applications. (2010)
[10] Belkadi F., Bonjour E., Camargo M., Troussier N., Eynard B.: A situation model to support awareness in collaborative design. International Journal of Human Computer Studies. (2013).
[11] Meta Object Facility (MOF) 2.0 Core Specification. OMG - (2005).
[12] Object Constraint Language OCL 2.0. OMG Final Adopted Specification. Document ptc/03-10-14. (2003).
[13] Tietze, D.A. (2001) `A framework for developing compo-nent-based co-operative applications', GMD Research Series No. 7/2001, ISBN: 3-88457-390-X.
[14] Guicking, A., Tandler, P. and Avgeriou P.Agilo: A Highly Flexible Groupware Framework. In Book Groupware: Design, Implementation, and Use. LNCS, Springer, Vol. 3706 (2005)
[15] Eclipse - an open development platform – http://www.eclipse.org

Luis Mariano Bibbo: Es Licenciado en Informática y Magister de la Universidad Nacional de La Plata. Participa en actividades de investigación en las áreas de modelado y métodos formales en el Laboratorio LIFIA de la Facultad de Informática, UNLP. Participa en proyectos de Investigación acreditados en UNLP. Ha realizado tareas de capacitación y asesoramiento tanto en el sector público como privado. Dirige trabajos para tesis de grado y posgrado en la Facultad de Informática de la UNLP. Es autor de diversos artículos en Conferencias y Workshops. Es docente en el área de Ingeniería de Software en el grado y el postgrado.

Roxana Giandini: es Doctora en Ciencias Informáticas de la UNLP (Universidad Nacional de La Plata). Tiene a su cargo en la UNLP, la cátedra ‘‘Desarrollo de Software Basado en Modelos’’. Es una de las autoras del libro ‘‘Desarrollo de Software Dirigido por Modelos. Conceptos teóricos y su aplicación práctica’’. Participa en actividades de investigación en las áreas de modelado y métodos formales en el Laboratorio LIFIA de la Facultad de Informática, UNLP. Es directora y co-directora de Proyectos de Investigación acreditados en UNLP y UTN. Ha realizado tareas de capacitación y asesoramiento tanto en el sector público como privado. Dirige trabajos para tesis de grado y posgrado en la Facultad de Informática de la UNLP y de la UNICEN. Es autora de diversos artículos en Revistas, Conferencias y Workshops. Es miembro de diversos Comités de Programas y proyectos. En 2010 fue representante por Argentina del CLEI elegido en la Conferencia Latinoamericana de Informática Asunción, Paraguay. Es Miembro del Comité Directivo de CIbSE – ‘‘Conferencia Iberoamericana en Software Engineering"-. Desde 2016 es Investigador Independiente Asociado de la CIC, Pcia. De Bs. As. Es co-Chair del Simposio Latinoamericano de Procesos de Negocio, arquitecturas y Sistemas Organizacionales del XLIII CLEI 2017.

Claudia Pons: Es Doctora en Ciencias Informáticas de la Universidad Nacional de La Plata (año 2000). Ha sido Investigadora del Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), y actualmente es Investigadora Adjunta en la Comisión de Investigaciones Científicas de la Provincia de Buenos Aires (CICPBA). Desde 2010 es Directora del Centro de Altos Estudios en Tecnología Informática (CAETI) de la Universidad Abierta Interamericana (UAI), Ciudad Autónoma de Buenos Aires, Argentina. Su área de interés es la ingeniería de software, en particular el desarrollo de software dirigido por modelos, los lenguajes y metodologías de modelado de software y la verificación formal de programas. Es miembro del Centro de investigación LIFIA y ejerce la docencia en carreras de grado y posgrado en la Facultad de Informática de la Universidad Nacional de La Plata y en la UAI. Actualmente dirige proyectos de investigación y desarrollo de software y es autora de varios artículos científicos referidos al tema.

Indice de Figuras: