INTERACCION
Revista Digital de AIPO
Asociación Interacción Persona-Ordenador
Vol. 3, No 2 (2022)
ISSN 2695-6578

Co-CIAT: Un Entorno para el Modelado Colaborativo en Notación CIAN

Co-CIAT: An Environment for Collaborative Modeling in CIAN Notation

Yoel Arroyo
Departamento de Tecnologías
y Sistemas de la Información
Facultad de Ciencias Sociales/UCLM
Talavera de la Reina, Toledo
España
Yoel.Arroyo@uclm.es

Ana Isabel Molina
Departamento de Tecnologías
y Sistemas de la Información
Escuela Superior de Informática/UCLM
Ciudad Real, España
AnaIsabel.Molina@uclm.es

Miguel Ángel Redondo
Departamento de Tecnologías
y Sistemas de la Información
Escuela Superior de Informática/UCLM
Ciudad Real, España
Miguel.Redondo@uclm.es

Recibido: 07.10.2022 | Aceptado: 30.11.2022

Palabras Clave

CSCW
MDD
groupware
modelado
diseño
awareness

Resumen

Este artículo presenta Co-CIAT, un editor gráfico colaborativo para la especificación de sistemas de trabajo en grupo usando la notación CIAN. En el artículo se describe el proceso de creación de Co-CIAT, como ejemplo de aplicación del método de desarrollo dirigido por modelos Learn-CIAM, que ha permitido la generación semiautomática de este editor colaborativo. El proceso de generación no sólo soporta la obtención del editor gráfico, sino también la incorporación de los mecanismos de *awareness* y otros componentes de coordinación y comunicación, necesarios para soportar este tipo de tareas grupales. Co-CIAT, al ser una herramienta *online* permite, a cualquier equipo multidisciplinar, trabajar sin la necesidad de reservar espacios, realizar desplazamientos, etc., lo cual redunda en una reducción de costes.

Keywords

CSCW
MDD
groupware
modeling
design
awareness

Abstract

This article introduces Co-CIAT, a collaborative graphical editor for the specification of group work systems using the CIAN notation. The article describes the creation process of Co-CIAT, as an example of the application of the Learn-CIAM *model-driven development* method, which has allowed the semi-automatic generation of this collaborative editor. The generation process not only supports obtaining the graphical editor, but also the incorporation of *awareness* mechanisms and other coordination and communication components, necessary to support this kind of group tasks. Co-CIAT, being an *online* tool, allows any multidisciplinary team to work without the need to reserve spaces, to commute, etc., which results in costs reduction.

1. Introducción

En 1984, se introduce por primera vez el término Trabajo Cooperativo Asistido por Computador (*Computer Supported Cooperative Work*, CSCW), por Irene Greif y Paul Cashman (Greif, 2019). Desde entonces y hasta la fecha, el enfoque CSCW ha permitido estudiar el funcionamiento básico del trabajo cooperativo, así como implantar las bases teóricas y métodos de soporte al diseño de los sistemas que le dan soporte (*groupware*) (Bullinger-Hoffmann et al., 2021). Son muchas las ventajas reconocidas que ofrece el uso del enfoque CSCW y de los sistemas *groupware*. Los equipos, que pueden encontrarse físicamente dispersos, pero que trabajan en proyectos en los que la comunicación y colaboración es esencial, necesitan conectar, comunicarse y compartir información con el resto de miembros del equipo. Gracias al uso de herramientas *groupware*, los integrantes de un equipo de trabajo pueden asistir a las sesiones de debate y trabajo en grupo desde su propio espacio físico de trabajo. El empleo de herramientas de soporte a la comunicación y colaboración distribuida reduce o elimina el tiempo y el coste de los desplazamientos, así como el coste de alquiler o compra de un área física de trabajo. A su vez, el espacio de almacenamiento de documentos, los costes de mantenimiento, y el tiempo de búsqueda de documentos, pueden reducirse significativamente (Hedjazi, 2018).

Los sistemas *groupware* resultarán eficientes en la medida en que sean capaces de reproducir y soportar de forma virtual las necesidades de comunicación, coordinación y compartición de

información que se dan en las sesiones presenciales de trabajo en grupo. Para ello, es necesario incorporar, a través de diferentes mecanismos visuales (*widgets* y componentes de la interfaz), información sobre el resto de los miembros del equipo y su actividad durante el desarrollo del trabajo grupal. La información suministrada a los equipos se apoya en la denominada consciencia de grupo (*awareness*) (Collazos et al., 2019), que resulta clave para la toma de decisiones, así como para la comunicación, colaboración y coordinación de los integrantes del grupo de trabajo. La aplicación eficiente del *awareness* requiere de mecanismos de visualización que faciliten la generación y la compartición de información, sin provocar una carga cognitiva adicional que dificulte y afecte negativamente el desarrollo de la actividad (Schnaubert & Bodemer, 2022).

Por su parte, el propio proceso de diseño de los sistemas *groupware* resulta complejo. Con el objetivo de dar soporte a la especificación de este tipo de aplicaciones se propuso la notación gráfica CIAN (Molina et al., 2013), soportada por la herramienta CIAT (Giraldo et al., 2009). CIAT es un editor gráfico que permite crear modelos en notación CIAN, y especificar así los requisitos de colaboración e interacción de los sistemas *groupware*. Sin embargo, el editor gráfico CIAT no es colaborativo. Teniendo en cuenta que el propio proceso de creación de modelos en esta notación puede realizarse de forma colaborativa, y posiblemente distribuida, se propone hacer evolucionar CIAT hacia una versión colaborativa, a la que se ha llamado Co-CIAT. En el proceso de desarrollo de la versión colaborativa de CIAT se aplicará un proceso de desarrollo dirigido por modelos para la obtención de sistemas de aprendizaje colaborativo, llamado Learn-CIAM (Arroyo et al., 2021). Learn-CIAM es una metodología creada con el objetivo de dar soporte a diseñadores, ingenieros y docentes, en la creación y generación semiautomática de sistemas colaborativos de aprendizaje (*Computer-Supported Collaborative Learning*, CSCL) (Hmelo-Silver & Jeong, 2021). Consta de dos fases diferenciadas: una primera fase, que da soporte a la generación de distintos editores gráficos indepen-dientemente del dominio de entrada (consistente en un conjunto de nodos y enlaces, que componen los diagramas); y una segunda fase, que genera e incorpora componentes para obtener los sistemas colaborativos de aprendizaje finales.

El objetivo de este trabajo es el de dotar a los distintos implicados (*stakeholders*) en el diseño de sistemas *groupware*, que hagan uso de la notación CIAN, de una herramienta (Co-CIAT) de creación colaborativa de modelos en esta notación. En este artículo se describe el proceso de obtención de esta aplicación, que se beneficia del enfoque CSCW, e incorpora una serie de características de *awareness* que han sido evaluadas y utilizadas en trabajos anteriores. En su creación se han seguido las diferentes etapas de las que consta la primera fase de la metodología Learn-CIAM, la cual, tal y como se comentó anteriormente, permite obtener editores gráficos independientes del dominio de entrada (y no necesa-riamente de aprendizaje, que se aborda en la segunda fase de dicha metodología).

Este artículo se estructura como sigue. En el apartado 2 se describen algunos trabajos relacionados o herramientas existentes que permiten trabajar o generar editores gráficos colaborativos. El apartado 3 describe el proceso de creación del editor gráfico Co-CIAT, introduciendo, en primer lugar, la metodología, la notación, así como las tecnologías y herramientas que han servido de base a su desarrollo. Por último, se enumeran las conclusiones extraídas de este trabajo, y las líneas de continuación del mismo.

2. Trabajos relacionados

Contar con un entorno colaborativo de modelado facilita a los miembros de un equipo de trabajo compartir información y coordinar sus actividades en el contexto de un proyecto de diseño conjunto (Bullinger-Hoffmann et al., 2021; Di Ruscio et al., 2017), pudiendo, además, interactuar de manera distribuida en la construcción de un modelo o artefacto, de acuerdo con la especificación de un objetivo o de una tarea concreta (Bucchiarone et al., 2021).

En la literatura existen numerosas aportaciones y herramientas colaborativas, de distinta índole, que abordan el modelado en grupo. Nuestro interés se centrará en aquellas que incorporen un conjunto amplio de características de soporte al *awareness*, así como de componentes de comunicación y coordinación y que, a su vez, admitan diferentes dominios gráficos, es decir, permitan el diseño de diferentes tipos de diagrama. En concreto, y puesto que se pretende soportar el trabajo colaborativo en tiempo real (al mismo tiempo), se estaría hablando de *herramientas de modelado colaborativo síncrono*. Debido a que existe una gran variedad de este tipo de herramientas, se pasa a describir, a continuación, algunas de las más relevantes encontradas en la literatura y que se alinean con la consecución de nuestros objetivos.

HAMSTERS (*Human-centered Assessment and Modeling to Support Task Engineering for Resilient Systems*) (Martinie et al., 2015) es un editor de modelado gráfico para la edición de diagramas de tareas basados en la notación CTT (*ConcurTaskTrees*) (Paternò, 2003), que incluye, sobre la base de esta notación, nuevos conceptos relacionados con la identificación de posibles errores humanos y como prevenirlos. Los autores, Koller et al. (Koller et al., 2016), han trabajado en una versión colaborativa, que incorpora la funcionalidad necesaria para que los usuarios puedan comunicarse, como es la inclusión de un *chat*, así como algunos mecanismos de *awareness*, tales como la posibilidad de añadir notas o realizar comentarios sobre las tareas (indicando el usuario que las añade), además de un historial de cambios en el que quedan

registradas las modificaciones realizadas por cada usuario en el diagrama. Sin embargo, la versión colaborativa de esta aplicación se encuentra en desarrollo y sólo existe un prototipo de la misma. Por otra parte, permite trabajar de forma colaborativa en un único dominio específico, el de los diagramas de tareas CTT ampliado.

Obeo Designer[1] es la versión colaborativa profesional (no libre) de *Sirius*[2], un entorno de modelado gráfico de sistemas complejos (*software*, actividades empresariales, etc.). Su versión colaborativa aplica un enfoque basado en la nube para la compartición de los modelos y toda la información relacionada con ellos. Sin embargo, este sistema no incluye un adecuado soporte al *awareness*, y se limita a proporcionar cierto control sobre el contexto compartido, mediante el bloqueo de regiones, evitando conflictos a la hora de modificar y compartir los modelos.

AToMPM[3] (*A Tool for Multi-Paradigm Modeling*) (Syriani et al., 2013) es un entorno de modelado colaborativo de código libre, que se puede utilizar desde el navegador *web*, lo cual evita problemas de portabilidad entre plataformas o dispositivos. AToMPM es una evolución, y sucesor directo, de su homónimo monousuario, AToM3 (De Lara & Vangheluwe, 2002). Se trata de un entorno que permite generar diferentes sistemas de modelado para dominios específicos, además de facilitar la transformación, manipulación y mantenimiento de los modelos, e incluso la generación de código a partir del diseño realizado por el usuario en las etapas finales del desarrollo. Para ello se apoya en la nube, donde se almacena toda la información para la compartición de información entre los usuarios. Sin embargo, al igual que *Obeo Designer*, también basada en la nube, únicamente da soporte a la compartición de los modelos, pero no incorpora mecanismos adecuados de *awareness.*

SpacEclipse (Gallardo et al., 2012) es una evolución de la herramienta de modelado *Space-Design* (Duque et al., 2008), creada como *plug-in* de *Eclipse*, que permite la generación semiautomática de sistemas colaborativos síncronos de modelado. Este sistema es una de las herramientas más interesantes y potentes de las analizadas, ya que facilita la generación de editores gráficos colaborativos independientes del dominio. Además, incorpora un gran número de características y mecanismos de *awareness*, así como otros componentes para la comunicación y la coordinación síncrona, que no incluyen las herramientas anteriormente comentadas. De hecho, *SpacEclipse* ha sido adaptada, mejorada y ampliada en trabajos más recientes (Arroyo et al., 2019, 2021), como parte de la metodología Learn-CIAM, lo cual pone de manifiesto que se trata de una herramienta potente y muy flexible.

En la Tabla 1 se muestra una comparativa de algunas de las características más relevantes de cada una de las herramientas analizadas, indicando: 1) la plataforma de destino; 2) el dominio gráfico con el que permite trabajar (que podrá ser uno en concreto o personalizado), y si permite crear dominios de modelado según la necesidad del diseñador; 3) si contempla características de *awareness* (en mayor o menor medida, es decir, si incluye un soporte a esta característica bastante rico o completo, o por el contrario, escaso o parcial); y 4) si incorpora *widgets* o componentes de soporte a la coordinación y la comunicación (como *chats*, paneles de sesión, etc.).

Tabla 1: Herramientas de modelado colaborativas y principales características

Entorno colaborativo	Plataforma	Dominio	*Awareness*[4]	Colab. (*Widgets*)
HAMSTERS	NetBeans	CTT ampliado	√	Sí
Obeo Designer	Web	Personalizado	~	No
AToMPM	Web	Personalizado	~	No
SpacEclipse	Eclipse	Personalizado	√	Sí

Pese a que la tendencia actual es la generación de sistemas multiplataforma basados en la nube, como ocurre con *Obeo Designer* o AToMPM, tal y como se ha visto, existen propuestas validadas e implementadas sobre la plataforma *Eclipse*. El uso de plataformas modulares y basadas en *plug-ins*, como *Eclipse*, juega un papel importante en la creación de herramientas de modelado colaborativas, gracias a los *frameworks* de modelado y generación de editores gráficos que incorpora. A su vez, el uso de entornos integrados de desarrollo (*Integrated Development Environment*, IDE) flexibles y ágiles, como *Eclipse*, en los que el usuario pueda añadir, personalizar o eliminar funcionalidades y vistas del entorno de modelado según sus necesidades, facilita la adaptación del entorno colaborativo a los requisitos específicos del proyecto y de la organización. Además, este IDE es un entorno de desarrollo multiplataforma contrastado, profesional y de largo recorrido, que permite a los diseñadores trabajar sobre distintos sistemas operativos.

Por tanto, de los entornos descritos, el que mejor se adapta a nuestras necesidades es la versión más reciente de *SpacEclipse* (Arroyo et al., 2019, 2021), ya que está implementada sobre la plataforma *Eclipse*, lo cual lo hace flexible y extensible, e incluye un conjunto de *widgets* de soporte a la colaboración y al *awareness* bastante completo.

[1] https://www.obeodesigner.com/en/
[2] https://www.eclipse.org/sirius/
[3] https://atompm.github.io/

[4] *Leyenda:* (√) Soporte bastante rico o completo; (~) Soporte escaso o parcial

3. Co-CIAT: editor gráfico colaborativo

A continuación, se describe cada uno de los elementos, así como el proceso de creación de las distintas versiones o evoluciones, que constituyen la base del editor gráfico colaborativo final (Co-CIAT): (1) la notación gráfica soportada (CIAN) y la primera versión del editor gráfico (CIAT); (2) una nueva versión del editor gráfico monousuario, de soporte a la creación de modelos en dicha notación, pero que hace uso de tecnologías más actuales; (3) el editor gráfico colaborativo final (Co-CIAT), el cual ha sido generado utilizando como dominio de entrada la herramienta descrita en el apartado 3.2 (la versión más reciente del editor gráfico monousuario CIAT); y para finalizar, (4) una comparativa de los procesos y los aspectos metodológicos que han permitido generar cada una de las versiones.

3.1 Obtención del editor gráfico de partida (CIAT)

CIAM (*Collaborative Interactive Applications Methodology*) (Molina et al., 2008) es una metodología propuesta para facilitar el diseño de sistemas de trabajo en grupo. Surge con el objetivo de guiar a diseñadores e ingenieros *software* en la especificación conceptual de la capa de presentación (interfaz de usuario) de sistemas *groupware*. Dicha metodología es soportada por una herramienta, CIAT (*Collaborative Interactive Applications Tool*) (Giraldo et al., 2009), que permite crear modelos basados en la notación CIAN (*Collaborative Interactive Applications Notation*) (Molina et al., 2013). Ambas, notación y herramienta, se encuentran enmarcadas dentro de la metodología CIAM.

En el proceso de diseño de la capa de presentación de un sistema *groupware*, según la metodología CIAM, se ha de seguir una serie de fases/etapas (Molina et al., 2009): 1) *Especificación del sociograma*: en esta fase se debe modelar la estructura de la organización (sus actores, roles, agrupaciones, etc.); 2) *Modelado del Proceso*: en esta fase se detallan las principales tareas (o procesos) individuales y en grupo, así como los participantes en cada una de ellas; 3) *Modelado de responsabilidades*: en esta etapa se modelan las responsabilidades individuales y compartidas entre los implicados; 4) *Modelado de las tareas en grupo*: en esta fase se describen, con mayor nivel de detalle, las tareas en grupo (colaborativas y cooperativas) identificadas en la etapa anterior; y 5) *Modelado de la interacción*: en esta última etapa se modelan los aspectos más interactivos del sistema, haciendo uso de la notación CTT (*ConcurTaskTrees*) (Paternò, 2003).

En cada una de estas fases o etapas se crean distintos modelos o especificaciones gráficas, haciendo uso de la notación CIAN (Molina et al., 2013). El diseñador podría crear los modelos a mano, o utilizar alguna herramienta de dibujo convencional (*Microsoft Paint*, *PowerPoint*, etc.). Sin embargo, para facilitar esta labor, se creó un editor gráfico que permitiera (y a la vez guiara) a los diseñadores en la creación de las especificaciones que propone CIAM y en el uso de su notación. Este editor gráfico es CIAT (Giraldo et al., 2009).

La primera versión de CIAT se creó siguiendo el paradigma de desarrollo dirigido por modelos (*Model-Driven Development*, MDD), apoyándose en el marco de desarrollo EMF/GMF (*Eclipse/Graphical Modeling Frame work*). EMF/GMF es un marco que facilita la creación y mantenimiento de los modelos necesarios para la generación de editores gráficos en Eclipse. Para ello, incorpora un diagrama de bloques predefinido, que sirve de guía en todo el proceso. Este diagrama fue utilizado para la generación semiautomática del editor gráfico CIAT (Figura 1), que pasó por los siguientes pasos (en cada uno de los cuales se generaron distintos ficheros):

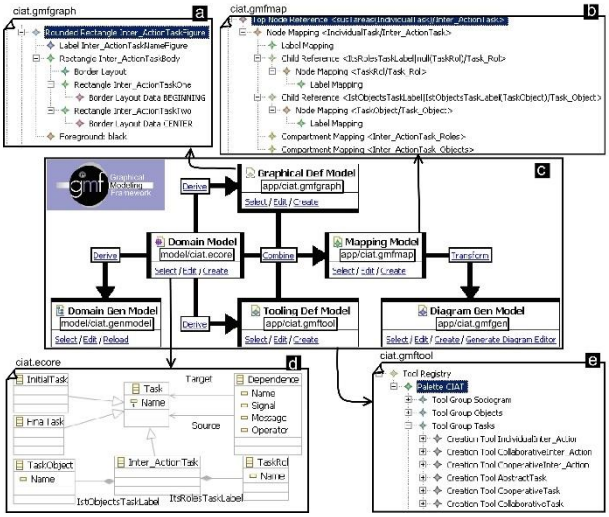

Figura 1: Proyecto EMF/GMF y su uso para generar el editor gráfico CIAT (extraído de (Giraldo et al., 2009))

1. *Especificación del modelo gráfico*: modelo que sirve para definir la estructura gráfica de cada uno de los nodos y enlaces, basados en la notación CIAN. Se corresponde con un primer boceto del aspecto visual que debería tener cada uno de los nodos y enlaces que soportará el editor gráfico.
2. *Especificación del modelo de dominio*: modelo de dominio que describe cada uno de los nodos y enlaces de la notación soportado por el editor gráfico, atendiendo al boceto creado en 1. Se corresponde con la creación de dos ficheros: el modelo de dominio, con extensión *'.ecore'* (Figura 1.d); y el modelo gráfico, con extensión *'.gmfgraph'* (Figura 1.a).
3. *Especificación de la paleta del editor*: modelo necesario para la especificación del conjunto de nodos y enlaces que deben estar disponibles en la paleta del editor gráfico. Se corresponde con un fichero con extensión *'.gmftool'* (Figura 1.e).
4. *Implementación del modelo de mapeo*: modelo necesario para relacionar cada uno de los nodos y

enlaces especificados en los modelos anteriores: dominio, gráfico y paleta del editor; de manera que se pueda, por ejemplo, arrastrar y crear el nodo o enlace correspondiente al hacer *drag & drop* desde la paleta hacia el área de edición (*canvas*). Se corresponde con la creación de un fichero con extensión *'.gmfmap'* (Figura 1.b).

Una vez creados todos los modelos enumerados, es suficiente con lanzar el proceso de generación automático para obtener el editor gráfico en forma de *plug-in* de *Eclipse*. Como se puede apreciar, son varios los modelos y especificaciones a crear, con la complejidad que este proceso conlleva, no solo por su definición, si no por posibles necesidades de mantenimiento y/o actualización de los mismos. La nueva versión de CIAT, y que sirve como dominio de entrada para el editor gráfico colaborativo Co-CIAT, ha sido generada utilizando un proceso simplificado, más actual, y que se describe en el siguiente apartado.

3.2 Creación de la nueva versión del editor gráfico monousuario CIAT

La nueva versión de CIAT se creó siguiendo también un proceso de desarrollo dirigido por modelos, pero utilizando un conjunto de tecnologías, herramientas y lenguajes que simplifican significativamente el proceso de generación automática. Tanto para la generación como posterior uso del nuevo editor gráfico, es necesario disponer de las siguientes tecnologías y herramientas en el entorno de desarrollo *Eclipse*:

1. *Paquete Eclipse Modeling Tools*[5]*:* entorno de desarrollo necesario para la generación y uso del editor gráfico en forma de *plug-in* de *Eclipse*. Incluye un conjunto predefinido de *plug-ins* de soporte al paradigma de desarrollo dirigido por modelos.
2. *Epsilon*[6]*:* familia de lenguajes y de herramientas de soporte a la automatización de tareas comunes dentro del paradigma de desarrollo dirigido por modelos en *Eclipse*, tales como la generación de código, la transformación entre modelos, y la validación de los mismos (a partir de modelos EMF, UML, *Simulink* o XML, entre otros). Incluye a su vez la herramienta *Eugenia*, que sustituye al diagrama de bloques descrito en el apartado anterior. Esta herramienta es capaz de generar, a partir de un único fichero de dominio, cada uno de los ficheros mencionados anteriormente (*.ecore, .gmfgraph, .gmftool, .gmfmap*).
3. *Editor Emfatic*[7]*:* editor de texto de soporte a la navegación, edición, y conversión de modelos *Ecore*, que utiliza una sintaxis compacta y más legible, muy similar a la del lenguaje de programación Java (y en concreto, al lenguaje *Emfatic*[8]).
4. *QVTo* (*Query/View/Transformation) Operational*[9]: *plug-in* que, junto a los lenguajes de *Epsilon* y la herramienta *Eugenia*, es capaz de realizar transformaciones *modelo-a-modelo* complejas.

Una vez se dispone del conjunto de tecnologías y herramientas anterior, el proceso de diseño y generación puede comenzar. En la elaboración del nuevo editor gráfico se han seguido los pasos descritos en la primera fase de la metodología Learn-CIAM (Arroyo et al., 2021). Pese a que Learn-CIAM es una metodología enfocada en el desarrollo de sistemas colaborativos de aprendizaje, fue diseñada en dos fases diferenciadas al detectarse que la primera de ellas podría ser independiente del dominio. Esta fase se compone de varias etapas (Figura 2), que se siguieron en la generación del nuevo editor gráfico:

1. *Especificación informal:* modelo que sirve para diseñar el aspecto de cada uno de los nodos y enlaces que debe soportar posteriormente el editor. Se corresponde con un boceto inicial del aspecto visual de cada uno de estos elementos. En este trabajo, se corresponde con el boceto de los componentes gráficos de la notación CIAN que se podrán crear usando el editor.
2. *Modelado del dominio:* modelo de dominio que incluye cada uno de los nodos y enlaces, así como sus relaciones, atendiendo al diseño especificado en 1. En este trabajo, se corresponde con un único fichero en lenguaje *Emfatic* (*CIAT.emf*), que recoge toda esta información.
3. *Personalización del editor gráfico:* una vez que se dispone del modelo de dominio, es posible generar el editor gráfico de manera automática. En el caso de que el resultado final no sea el deseado, se puede aprovechar la potencia que ofrece el paquete *Epsilon* para crear un fichero de personalización adicional (con extensión *'.eol'*, en lenguaje *EOL*), que permite modificar el aspecto por defecto del editor. En este trabajo, se ha generado un fichero de personalización, denominado *Ecore2GMF.eol*, para mejorar el aspecto de la paleta, así como de algunos de los nodos, que requerían una estructura visual diferente, y más compleja, a la proporcionada por defecto.

[5] *Paquete Modeling Tools*: https://www.eclipse.org/downloads/packages/release/2022-03/r/eclipse-modeling-tools
[6] [Eclipse URI] http://download.eclipse.org/epsilon/updates/2.4/
[7] [Eclipse URI] https://download.eclipse.org/emfatic/update/
[8] Lenguaje *Emfatic*: https://www.eclipse.org/modeling/emft/emfatic/
[9] [Eclipse URI] https://download.eclipse.org/mmt/qvto/updates/releases/3.4.0

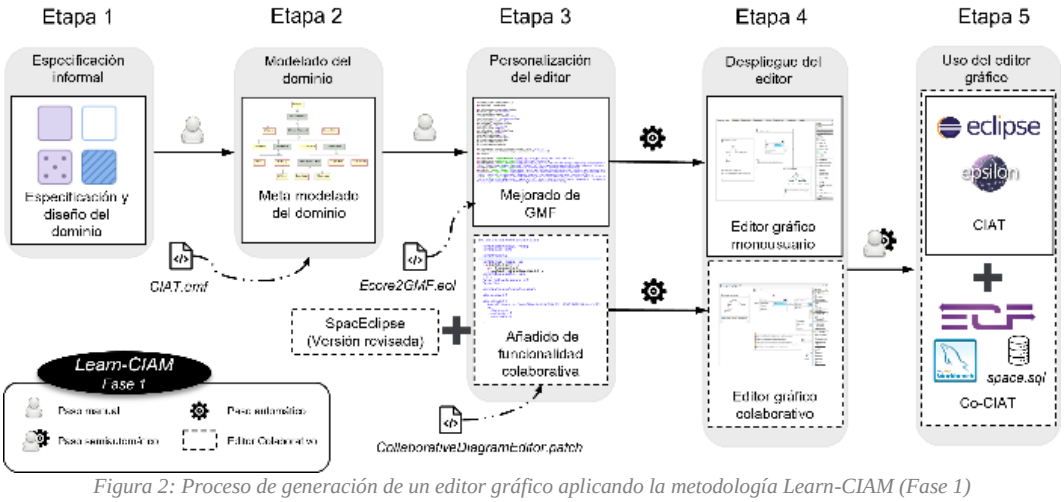

Figura 2: Proceso de generación de un editor gráfico aplicando la metodología Learn-CIAM (Fase 1)

4. *Despliegue del editor gráfico:* etapa correspondiente al despliegue del editor gráfico generado (ejecutable para su instalación en diferentes instancias del entorno Eclipse).
5. *Uso del editor gráfico:* como su propio nombre indica, se corresponde con el uso del editor gráfico generado, previa instalación del mismo.

La Figura 3 muestra una comparativa de las tecnologías utilizadas en cada uno de los procesos de generación, EMF/GMF y *Eugenia*, respectivamente. Si atendemos al proceso EMF/GMF, se distinguen dos procesos internos, uno correspondiente al *framework* EMF, encargado de generar e implementar la sintaxis abstracta, y otro correspondiente a GMF, encargado de generar la sintaxis gráfica concreta del editor.

Para comenzar el proceso, el desarrollador debe definir la sintaxis abstracta a través de un metamodelo *Ecore*. Inmediatamente después, se realiza una primera transformación *modelo-a-modelo* para obtener el modelo generador EMF (*GenModel*). Este modelo contiene información de bajo nivel que especifica cómo debe ser implementado finalmente el metamodelo en código Java (p. ej., el paquete Java sobre el que el código debe ser generado, información sobre *copyright* que deba ser ligada a los ficheros generados, etc.). Finalmente, este modelo sufre una última transformación *modelo-a-texto* en la que se genera el código Java final y los ficheros de configuración. Si el metamodelo *Ecore* es modificado en mitad del proceso, EMF incorpora un generador que puede detectar los cambios y propagarlos a su correspondiente modelo generador *GenModel* sin sobrescribir las personalizaciones definidas por el usuario. Sin embargo, este resulta efectivo sólo cuando se producen cambios simples sobre el metamodelo (p. ej., añadir un nuevo atributo a una clase ya definida en el metamodelo).

Cuando se producen cambios de mayor complejidad, como mover una clase a un paquete diferente, el modelo *GenModel* deberá ser regenerado desde cero, siendo necesario volver a realizar las personalizaciones manualmente. Esto supone un proceso de mantenimiento complejo para los desarrolladores, que además nunca terminan de tener claro qué cambios se sobrescribirán sobre los metamodelos automáticamente sin dar problemas y cuales, por el contrario, deteriorarían todo el trabajo realizado hasta ese momento. Por ello, en la práctica, es habitual que los desarrolladores tengan que escribir documentos de mantenimiento de apoyo, en los que dejar constancia de los cambios que causan una u otra situación para saber cómo reaccionar en el futuro. Además, una vez el metamodelo *Ecore* ha sido definido y el código EMF ha sido generado, el desarrollador debe construir tres modelos adicionales, específicos de GMF, necesarios para implementar el lenguaje de soporte al editor gráfico (sintaxis concreta). Estos modelos se corresponden con los mencionados previamente (Figura 1), es decir: el modelo de gráficos (*.gmfgraph*), el modelo de herramientas (*.gmftool*), y el modelo de mapeo (*.gmfmap*).

Como se puede observar en la Figura 3, el modelo de mapeo sufre una transformación *modelo-a-modelo* y genera el modelo *GMFGen*, que contiene la información de bajo nivel que necesita el generador de código GMF para producir finalmente la sintaxis concreta que define al editor gráfico (código Java y ficheros de configuración).

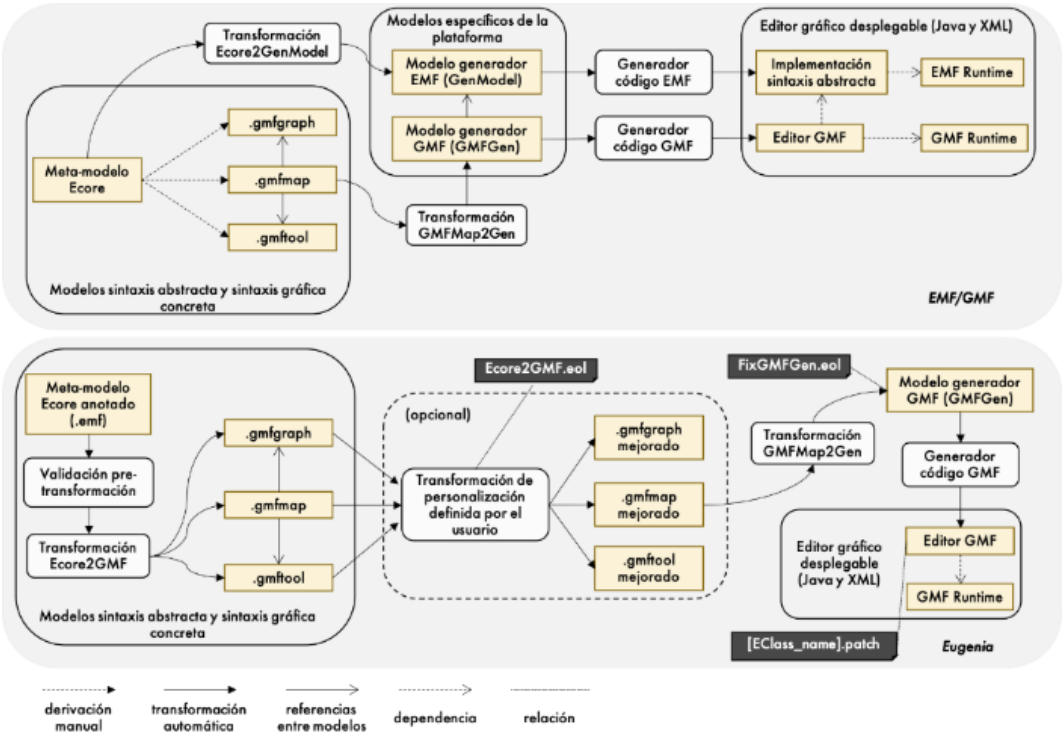

Figura :. Comparativa de los procesos de desarrollo de editores gráficos al usar las opciones tecnológicas EMF/GMF y Eugenia

Pese a que, en términos de automatización, GMF incorpora el diagrama de bloques como asistente (Figura 1), se ha demostrado que los modelos generados no resultan demasiado eficientes, incluso para dominios simples. Como resultado, el desarrollador debe crear a mano estos modelos de bajo nivel usando unos editores en forma de árbol (Figura 1). Además, a la complejidad de crear estos modelos a mano, hay que añadir: 1) la poca información relacionada con los errores que GMF arroja en la consola; y 2) que al contrario que EMF, GMF no aporta ni siquiera un generador que permita realizar cambios a mitad del proceso y sobrescriba el editor generado, por lo que obliga a los desarrolladores a empezar desde cero y añadir los cambios manualmente, incluso para generar editores con los dominios más simples. Por tanto, implementar y mantener un editor gráfico usando EMF/GMF resulta un proceso bastante laborioso, principalmente para aquellos desarrolladores sin experiencia previa con esta tecnología. En ocasiones, incluso los desarrolladores expertos podrían llegar a desechar el uso de esta tecnología ante un proceso tan manual y repetitivo.

Eugenia surge con la clara intención de automatizar aún más este proceso, y hacerlo más transparente, sencillo y directo para los desarrolladores. Si atendemos nuevamente a la Figura 3, *Eugenia* supone un proceso de desarrollo simplificado que facilita la labor de los desarrolladores usando varios lenguajes de alto nivel y un único metamodelo anotado como artefacto de entrada. A partir de este metamodelo, se produce una serie de transformaciones automáticas *modelo-a-modelo* con las que es posible obtener los modelos específicos de la plataforma requeridos por los generadores de código EMF/GMF, de una manera consistente y repetible, para finalmente, acabar generando el editor gráfico deseado.

La principal novedad de *Eugenia* es la incorporación de nuevas transformaciones y la posibilidad de realizar personalizaciones usando lenguajes de alto nivel sobre los artefactos principales del proceso anterior (EMF/GMF). Para ello, el metamodelo *Ecore* debe extenderse con las anotaciones propias de *Eugenia* para poder iniciar el proceso de generación automática. Estas anotaciones se escriben en código *Emfatic*, y se organizan en seis categorías principales:

- *@gmf.diagram:* anotaciones que sirven para especificar las propiedades del diagrama, tales como el tipo del elemento raíz del modelo, la extensión del fichero del editor gráfico, y si el editor debe ser exportado como *plug-in* o aplicación Java independiente (*standalone*).
- *@gmf.node:* anotaciones utilizadas para indicar qué elementos de la sintaxis abstracta deben representarse como nodos (vértices) en la sintaxis gráfica, y especificar su forma, color, tamaño, etiqueta, etc.
- *@gmf.link:* anotaciones que sirven para indicar qué elementos de la sintaxis abstracta deben ser representados como enlaces en la sintaxis gráfica, y especificar su grosor, color, estilo, flechas, etiquetas, etc.
- *@gmf.compartment:* anotaciones usadas para indicar qué nodos pueden contener otros nodos de la sintaxis gráfica de forma anidada (p. ej., en la sintaxis

concreta de un diagrama UML, los atributos se encuentran dentro de una clase).

- *@gmf.affixed:* anotaciones que sirven para indicar qué nodos pueden adjuntarse al borde de otros nodos de la sintaxis gráfica (p. ej., en un diagrama BPMN 2.0. los eventos suelen adjuntarse a los bordes de las actividades).
- *@gmf.label:* anotaciones utilizadas para especificar etiquetas adicionales para un nodo concreto.

```
@gmf.diagram()
class CIATDiagram {
      val Sociogram[*] itsSociograms;
      val ProcessDiagram[*] itsProcessDiagrams;
      val DomainDiagram[*] itsDomainDiagrams;
      val CTTDiagram[*] itsCttDiagrams;
      val Link[*] itsLinks;
}

......

@gmf.node(label="name",
tool.small.bundle="org.chico.ciat.figures",
tool.small.path="images/Sociogram.gif")
class Sociogram {
      attr String name;
      @gmf.compartment(layout="free", collapsible="false")
      val SociogramNode[*] itsSociogramNodes;
}

......

abstract class DependencyElement {}

class Dependency extends Link {
      ref DependencyElement[1] source;
      ref DependencyElement[1] target;
}

@gmf.link(label="eLabel", source="source", target="target",
style="solid", width="1", color="0,0,0",
tool.small.bundle="org.chico.ciat.figures",
tool.small.path="images/EnablingTransition.gif",
source.constraint="self <> oppositeEnd", target.constraint="self
<> oppositeEnd")
class Enabling extends Dependency {
      readonly attr String eLabel = ">>";
}
```

Listado 1: Porción de código extraída de CIAT.emf (Figura 2)

Una vez anotado, y tras ser validado previamente en busca de errores, el metamodelo principal sufre una primera transformación (*ECore2GMF*). Esta transformación es la encargada de generar automáticamente, a raíz de la conjunción de la sintaxis abstracta (metamodelo) y concreta (anotaciones), los modelos de gráficos, herramientas y mapeo correspondientes.

En aquellas ocasiones en las que el editor gráfico posea un dominio más complejo, *Eugenia* ofrece la posibilidad de realizar una transformación adicional, con la que modificar manualmente, de una manera más sencilla, los modelos anteriores. Para ello, el desarrollador deberá crear un fichero con el nombre exacto '*ECore2GMF.eol'*. Básicamente, lo que el desarrollador debe procurar es, usando el lenguaje *EOL*, definir una serie de instrucciones que permitan modificar los modelos anteriores y mejorar así el aspecto del editor gráfico final. Dicho de otra forma, el desarrollador puede realizar modificaciones directamente sobre los modelos cuando las anotaciones básicas de *Eugenia* no resulten ser suficiente para generar el aspecto del editor gráfico deseado. A su vez, *Eugenia* añade la posibilidad de, a partir de la definición de otro fichero escrito en lenguaje *EOL*, denominado *FixGMFGen.eol*, incluir código externo y enlazarlo al editor. Este fichero suele ser útil, por ejemplo, para enlazar de una manera rápida y sencilla el proyecto que contenga las figuras que serán utilizadas en el editor gráfica. Por último, *Eugenia* también ofrece la posibilidad de, al editor ya generado en código Java y XML (ficheros de configuración), aplicar parches (*.patch*) de manera automática que modifiquen el código. Por ejemplo, puede ser necesario acercar el texto de la etiqueta externa de los nodos del editor o, como se describirá más adelante en este mismo texto, incorporar código fuente capaz de incorporar la funcionalidad colaborativa.

Eugenia supone, por tanto, una opción más intuitiva y flexible que sus predecesores. Permite trabajar con lenguajes de alto nivel de abstracción de una manera más sencilla y transparente que la tecnología EMF/GMF, aporta más opciones de personalización para la generación automática del editor, y facilita el proceso de mantenimiento al ser capaz de, a través de un único fichero anotado y una serie de ficheros *EOL* más manejables, generar editores gráficos de mayor calidad y complejidad. Además, *Eugenia* ha resultado validada (Kolovos et al., 2017) y bien recibida por la comunidad *MDE* de *Eclipse*, existiendo evidencias de su uso en entornos académicos como la Universidad de Cádiz y la Universidad de York en cursos de máster de Ingeniería Informática, e incluso en entornos industriales y de investigación.

Esta simplificación puede apreciarse aún más en sistemas como el nuevo editor gráfico generado. El único fichero que se ha tenido que crear, de manera adicional, ha sido el fichero de personalización *Ecore2GMF.eol*. Como se ha descrito anteriormente, al no ser necesario crear cada uno de los modelos de manera manual, es suficiente con revisar el elemento que se desee personalizar en el modelo correspondiente para, a continuación, especificar las modificaciones visuales necesarias en el fichero de personalización. En las Figuras 4 y 5 se muestra, a modo de ejemplo, cómo se ha modificado el aspecto de uno de los nodos del editor gráfico (Figura 4), como el orden de los elementos de la paleta del editor gráfico (Figura 5), haciendo uso del fichero de personalización.

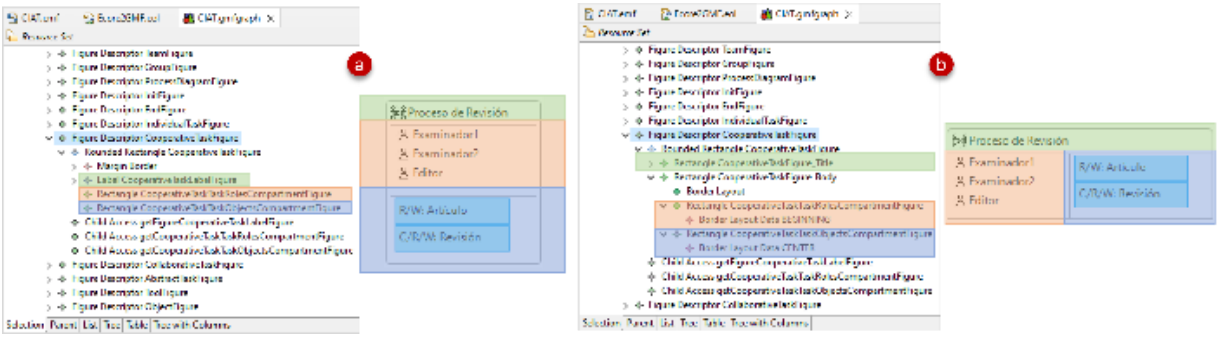

Figura 4: Nodo de una actividad cooperativa (CIAN) sin personalizar (a) y personalizado (b)

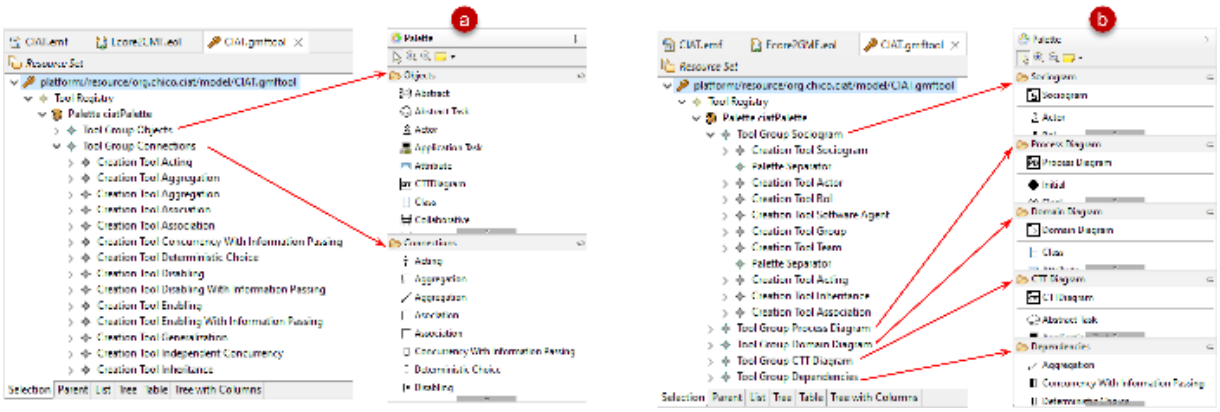

Figura 5: Paleta del editor CIAT sin personalizar (a) y personalizada (b)

3.3 Proceso de generación del editor colaborativo síncrono (Co-CIAT)

Aplicando la primera fase de la metodología Learn-CIAM, fue posible incorporar el soporte colaborativo a la última versión del editor gráfico monousuario (obtenido como resultado de los pasos indicados en el apartado 3.2). Para ello, de nuevo, fue necesario disponer de una serie de tecnologías y herramientas instaladas en el entorno:

1. *SpacEclipse (versión revisada)*: *plug-in* de *Eclipse* de soporte a la metodología de idéntico nombre (Gallardo et al., 2013), que da soporte a la generación de editores gráficos colaborativos independientes del dominio. Los editores gráficos colaborativos generados incluyen una serie de mecanismos de *awareness* y *widgets* de soporte al diseño colaborativo.
2. ECF (*Eclipse Communication Framework*)[10] : conjunto de *frameworks* que permite configurar las comunicaciones entre un servidor y los distintos clientes de un sistema o servicio determinado. Es el *plug-in* necesario para soportar la comunicación entre distintas instancias cliente de Co-CIAT, y así, permitir trabajar de forma colaborativa.
3. *MySQL Workbench*[11] y *MySQL Server*[12] : herramientas para arquitectos y diseñadores de bases de datos. Proporcionan capacidad para el modelado de datos y consultas en lenguaje SQL (*Structured Query Language*), así como una serie de herramientas para la configuración de servidores, administración de usuarios, copias de seguridad, y otras funcionalidades de soporte adicionales. Es, por tanto, el conjunto de herramientas necesario para la implementación y configuración del servidor de soporte a la base de datos utilizada por Co-CIAT.

El proceso de generación de Co-CIAT se llevó a cabo siguiendo, nuevamente, la primera fase de Learn-CIAM (Arroyo et al., 2021) (Figura 2). En concreto, en la etapa de personalización, y como se ha mencionado previamente, existe la posibilidad de utilizar parches genéricos (ficheros con extensión *'.patch'*), con una sintaxis idéntica a los parches utilizados por *Git* para el control de versiones. En este trabajo se creó un parche (*CollaborativeDiagram Editor.patch*) (Listado 2) (Figuras 2 y 3), para incorporar la funcionalidad colaborativa necesaria durante el proceso de generación automática. La ventaja de este procedimiento es que supone un "camino de ida y vuelta", ya que es posible

[10] [Eclipse URI] https://download.eclipse.org/rt/ecf/3.13.2/site.p2/

[11] MySQL Workbench: https://downloads.mysql.com/archives/workbench/

[11]2 MySQL Server: https://dev.mysql.com/downloads/windows/installer/5.7.html

también (gracias a *Eugenia*) aplicar o deshacer estos cambios, pulsando tan solo un botón (Figura 6). De esta forma, el usuario final puede escoger entre utilizar la versión de la herramienta de modelado individual (CIAT) o la versión colaborativa (Co-CIAT), en cualquier momento.

```
diff --git
org.chico.ciat.diagram/src/ciat/diagram/part/CiatDiagramEditor.java
org.chico.ciat.diagram/src/ciat/diagram/part/CiatDiagramEditor.java
index f4c32c5..266a093 100644
---
org.chico.ciat.diagram/src/ciat/diagram/part/CiatDiagramEditor.java
+++
org.chico.ciat.diagram/src/ciat/diagram/part/CiatDiagramEditor.java

@@ -61,13 +80,18 @@
import ciat.diagram.navigator.CiatNavigatorItem;
+import spaceclipse.collab.CUtilities;
+import spaceclipse.collab.interfaces.ICollaborativeEditor;
+import spaceclipse.herramientas.UsuarioPanel;
+import spaceclipse.space.SpacEclipse;

-public class CiatDiagramEditor extends DiagramDocumentEditor
implements IGotoMarker {
+public class CiatDiagramEditor extends DiagramDocumentEditor
implements ICollaborativeEditor, IGotoMarker {

/**
 * @generated
@@ -89,6 +113,8 @@
 */
    public CiatDiagramEditor() {
        super(true);
+       CUtilities.setEditor(this);
    }

 /**
@@ -297,6 +323,19 @@
    getDiagramGraphicalViewer());
    getDiagramGraphicalViewer().setContextMenu(provider);
    getSite().registerContextMenu(ActionIds.DIAGRAM_EDITOR_
        CONTEXT_MENU, provider, getDiagramGraphicalViewer());
+
+ org.eclipse.gmf.runtime.diagram.ui.parts.DiagramCommandStack stack
=
(org.eclipse.gmf.runtime.diagram.ui.parts.DiagramCommandStack)getComm
andStack();

......
```

Listado 2: Porción de código extraída de CollaborativeDiagramEditor.patch

Para llevar a cabo este proceso de generación, tal y como se muestra en la Figura 6, es necesario disponer en el espacio de trabajo de *Eclipse* de: 1) el conjunto de *plug-ins* que compone la nueva versión monousuario de CIAT; 2) la versión revisada de *SpacEclipse*; y 3) el parche colaborativo, listo para ser aplicado o no, a conveniencia del desarrollador.

Como se puede apreciar, el proceso es muy sencillo, una vez que el parche ha sido generado de manera correcta, pues es suficiente con seleccionar la opción deseada para incorporar o eliminar la funcionalidad colaborativa sobre el código fuente del editor monousuario. Por supuesto, si se quisiera o se considerara necesario lanzar el proceso de generación automático (completo), el parche sería aplicado durante dicho proceso de manera automática.

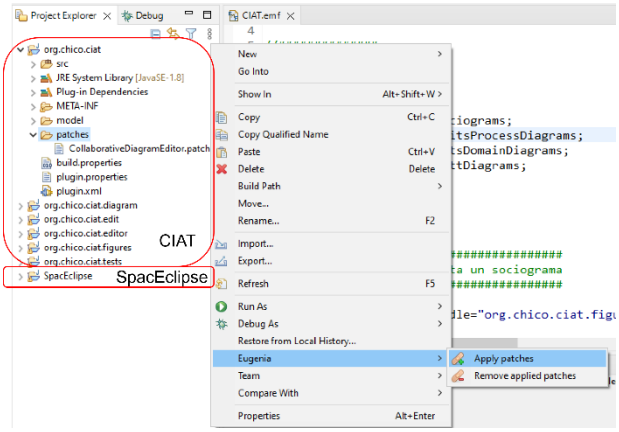

Figura 6: Co-CIAT y aplicación de parches

Una vez que se dispone del editor gráfico colaborativo, es necesario configurar un servidor de soporte a la base de datos (*script 'space.sql'*), que permita a los usuarios finales realizar acciones en Co-CIAT, tales como la creación de sesiones, la creación y actualización de usuarios, o conectarse a una sesión colaborativa de modelado, entre otras.

En la Figura 7 se muestra el aspecto final de la herramienta Co-CIAT. Tal y como se puede ver en la imagen, en la sesión colaborativa de modelado intervienen tres usuarios que hacen uso del editor gráfico de la notación CIAN. En el ejemplo se muestra el diagrama en esta notación (en concreto, el modelo de proceso) de un sistema de soporte a la gestión de los Trabajos Fin de Grado (TFG) de una Facultad. En el instante capturado los usuarios están discutiendo sobre la composición y elementos de este diagrama. En esta figura se pueden ver los principales componentes de la interfaz gráfica final de Co-CIAT, provenientes tanto del editor gráfico CIAT como del *plug-in SpacEclipse*. En el área central está el *canvas* o área de dibujo (Figura 7.a), en el que se puede ver los telepunteros de cada uno de los usuarios, identificados mediante un color distintivo. El resto del editor lo componen *widgets* y componentes colaborativos provenientes de *SpacEclipse*: panel de sesión (Figura 7.b), en el que aparecen los usuarios conectados a la sesión colaborativa; el chat estructurado (Figura 7.c), en el que se puede ver la conversación que están manteniendo los usuarios durante el proceso de diseño; y el panel de turnos (Figura 7.d), en el que los usuarios de la sesión pueden solicitar el turno para modificar el diagrama. Por último, es necesario indicar que también se incluyen distintas características y mecanismos *awareness*, que provienen de la versión revisada de *SpacEclipe*, tales como: 1) el uso de notificaciones textuales, para indicar quién está modificando el diagrama en un momento determinado (en este ejemplo se puede ver, desde la instancia de Co-CIAT del usuario Miguel, que el usuario Ana está *"editando…"* el diagrama); 2) sonidos, para avisar a los usuarios de que se ha producido un cambio de turno, en el cual se realiza además el correspondiente bloqueo del editor al resto de usuarios, evitando así posibles sobreescrituras; 3) uso de telepunteros y colores identificativos

para cada usuario conectado a la sesión; 4) uso de mensajes predeterminados en el *chat*; y 5) avatares únicos para cada usuario (que pueden modificar accediendo a su perfil, antes de unirse a la sesión colaborativa).

3.4 Discusión

La metodología Learn-CIAM (Arroyo et al., 2021) ha sido creada siguiendo el enfoque de desarrollo dirigido por modelos. Después de haber descrito cada uno de los procesos para la generación de los editores gráficos monousuario y colaborativo, se considera importante discutir, también, algunas diferencias metodológicas.

Al seguir un enfoque MDD, se ha tomado como base la arquitectura MDA (*Model-Driven Architecture*) en la generación de cada uno de los editores gráficos. Así, la Figura 8 presenta: a) la arquitectura MDA (Figura 8.a); b) la relación entre los modelos MDA y los modelos utilizados en la versión monousuario inicial (Figura 1); c) la relación entre los modelos MDA y los modelos y artefactos utilizados en la nueva versión monousuario (Figuras 2 y 3); y d) la relación entre los modelos MDA y los modelos y artefactos que intervienen en la generación de la versión colaborativa (Figuras 2 y 3).

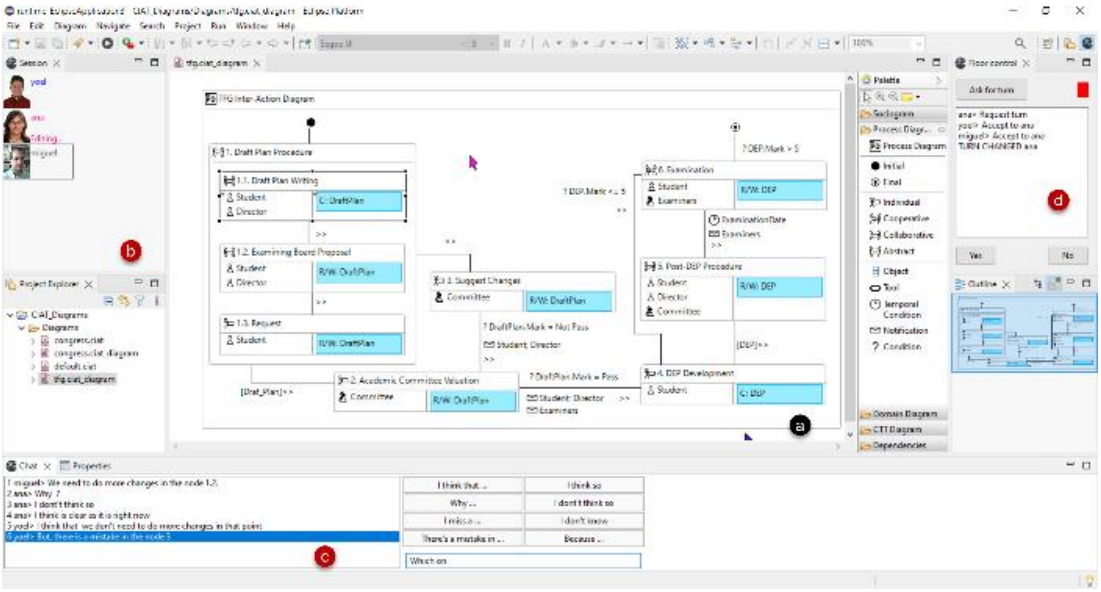

Figura 7: Interfaz de usuario colaborativa del editor gráfico Co-CIAT

Observando estos modelos, se pueden apreciar algunas diferencias:

- En la primera versión monousuario (Figura 8.b), el modelo abstracto (.ecore) y los modelos específicos de la plataforma (.gmfgraph, .gmftool, y .gmfmap) no sufren transformación alguna (concreción y/o abstracción). Esto se debe a la tecnología utilizada (EMF/GMF).
- En la nueva versión monousuario (Figura 8.c), sí se produce una primera transformación M2M (*Model-To-Model*), en la que se concreta el modelo abstracto (.emf) en tres modelos específicos de la plataforma (.gmfgraph, .gmftool, y .gmfmap). Esto confirma que la nueva metodología estaría simplificando este proceso (el usuario final debe crear y mantener menos modelos), provocado por la elección de la nueva tecnología (Eugenia).
- Por último, los procesos MDD de la nueva versión monousuario y la versión colaborativa son idénticos, a excepción de la incorporación de la funcionalidad colaborativa a partir de un parche (.patch), encargado de unir el código final generado (.java) con el *plug-in* SpacEclipse (también en Java) (Figura 8.d).

En términos metodológicos, por una parte, se debe destacar el primer punto, es decir, la simplificación conseguida gracias a la elección de la nueva tecnología. A su vez, es necesario advertir que la metodología utilizada está diseñada para la generación semiautomática de editores gráficos independientemente del dominio. En este trabajo, enfocado en la presentación de una nueva herramienta colaborativa (Co-CIAT), y en su comparación con una herramienta existente previa (CIAT), resulta evidente que los lenguajes y entornos objetivo finales son los mismos, es decir, Java y Eclipse. Sin embargo, la metodología (Learn-CIAM) ha sido creada con el objetivo de permitir generar sistemas en lenguajes y entornos de distinta naturaleza. Por ejemplo, sería posible generar sistemas *web* multiplataforma. Para conseguir este hito, y observando nuevamente la Figura 8.d., sería necesario cambiar los artefactos utilizados en la última transformación M2T (*Model-*

To-Text), es decir, el parche y la restricción impuesta por el uso del *plug-in* SpacEclipse. Este trabajo, por tanto, también pretende sentar las bases para poder alcanzar un mayor número de usuarios finales y dispositivos finales en trabajos futuros.

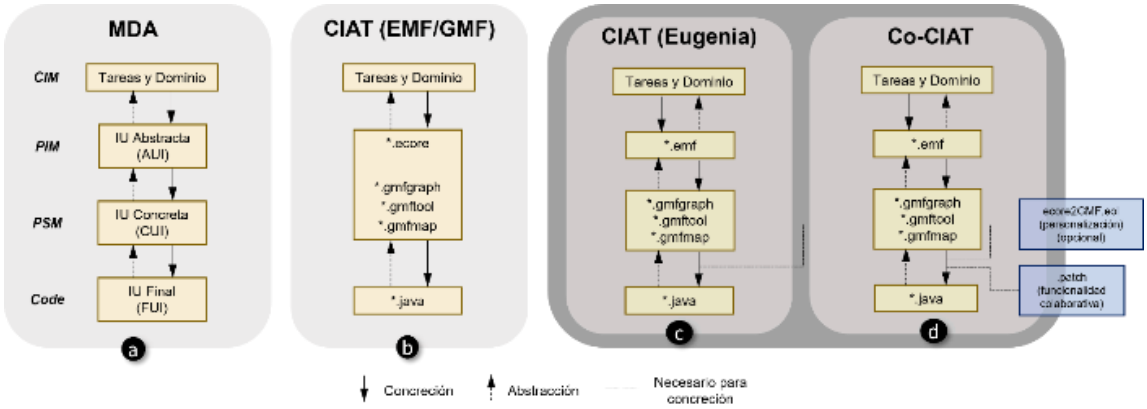

Figura 8: Comparativa MDD para la generación de los editores gráficos monousuario (CIAT) y colaborativo (Co-CIAT)

4. Conclusión

En este trabajo se ha descrito la evolución y proceso de generación de un editor gráfico colaborativo de soporte a la especificación de sistemas de trabajo en grupo en notación CIAN. Para ello, ha sido necesario generar una nueva versión del editor gráfico monousuario de partida (CIAT), utilizando un conjunto de tecnologías y lenguajes de soporte al desarrollo dirigido por modelos.

Y es que, en este artículo también se realiza una comparativa entre ambos procesos de generación, así como de la tecnología y lenguajes necesarios para la creación de este tipo de editores, permitiendo comprobar cómo se ha visto simplificado el proceso de obtención de la versión colaborativa de esta herramienta final (Co-CIAT).

A su vez, el proceso de generación de esta aplicación ha servido de ejemplo de aplicación de la metodología Learn-CIAM. Este ejemplo ha permitido constatar que la división de esta metodología en dos fases bien diferenciadas fue acertada. Tal y como se ha indicado con anterioridad, la primera de las fases de Learn-CIAM permite generar editores independientes del dominio, es decir, permite la generación de editores gráficos de notaciones no orientadas al modelado de requisitos de aprendizaje; mientras que la segunda de las fases sí que se enfoca en la generación del soporte necesario para obtener este tipo concreto de sistemas colaborativos (CSCL).

También, cabe indicar que, por tratarse de una herramienta de trabajo en grupo *online* permite, a cualquier equipo multidisciplinar, trabajar sin la necesidad de reservar espacios, realizar desplazamientos, etc. y, por tanto, permite reducir los costes derivados.

Como trabajo futuro, se plantea realizar varias evaluaciones y validaciones de satisfacción con distintos *stakeholders* que pudieran estar implicados en el diseño colaborativo de sistemas *groupware* y que, para ello, quieran hacer uso de la notación CIAN. Como escenarios de uso/evaluación de partida, se podría contemplar el diseño colaborativo de sistemas *groupware* para la gestión de Trabajos Fin de Grado (TFG) de una Facultad (como en el ejemplo incluido en este artículo), para la gestión y planificación de un Congreso, o para la planificación de una EvAU (Evaluación para el Acceso a la Universidad), entre otros.

Referencias

Arroyo, Y., Molina, A. I., Redondo, M. A., & Gallardo, J. (2021). Learn-CIAM: A Model-Driven Approach for the Development of Collaborative Learning Tools. *Applied Sciences* 2021, Vol. 11, Page 2554, 11(6), 2554. https://doi.org/10.3390/APP11062554

Arroyo, Y., Molina, A. I., Redondo, M. A., Gallardo, J., & Lacave, C. (2019). Collaborative Modeling of Group Learning Applications Using Eclipse Technology. *13th International Conference on Ubiquitous Computing and Ambient Intelligence*, 31(1), 21. https://doi.org/10.3390/proceedings2019031021

Bucchiarone, A., Ciccozzi, F., Lambers, L., Pierantonio, A., Tichy, M., Tisi, M., Wortmann, A., & Zaytsev, V. (2021). What is the future of modeling? *IEEE Software*, 38(2), 119–127. https://doi.org/10.1109/MS.2020.3041522

Bullinger-Hoffmann, A., Koch, M., Möslein, K., & Richter, A. (2021). Computer-Supported Cooperative Work - Revisited. I-Com, 20(3), 215–228. https://doi.org/10.1515/ICOM-2021-0028/MACHINEREADABLECITATION/RIS

Collazos, C. A., Gutiérrez, F. L., Gallardo, J., Ortega, M., Fardoun, H. M., & Molina, A. I. (2019). Descriptive theory of awareness for groupware development. *Journal of Ambient Intelligence and Humanized Computing*, 10(12), 4789–4818. https://doi.org/10.1007/s12652-018-1165-9

De Lara, J., & Vangheluwe, H. (2002). Atom3: A tool for multi-formalism and meta-modelling. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics*), 2306, 174–188. https://doi.org/10.1007/3-540-45923-5_12

Di Ruscio, D., Franzago, M., Malavolta, I., & Muccini, H. (2017). Envisioning the future of collaborative model-driven software engineering. *39th International Conference on Software Engineering Companion, ICSE-C 2017*, 219–221. https://doi.org/10.1109/ICSE-C.2017.143

Duque, R., Gallardo, J., Bravo, C., & Mendes, A. J. (2008). Defining tasks, domains and conversational acts in CSCW systems: the SPACE-DESIGN case study. *Journal of Universal Computer Science*, 14(9), 1463–1479. https://doi.org/10.3217/jucs-014-09-1463

Gallardo, J., Bravo, C., & Redondo, M. A. (2012). A model-driven development method for collaborative modeling tools. *Journal of Network and Computer Applications*, 35(3), 1086–1105. https://doi.org/10.1016/j.jnca.2011.12.009

Gallardo, J., Molina, A. I., Bravo, C., & Redondo, M. A. (2013). A model-driven and task-oriented method for the development of collaborative systems. *Journal of Network and Computer Applications*, 36(6), 1551–1565. https://doi.org/10.1016/j.jnca.2013.03.016

Giraldo, W. J., Molina, A. I., Collazos, C. A., Ortega, M., & Redondo, M. A. (2009). CIAT, A Model-Based Tool for Designing Groupware User Interfaces Using CIAM. In Computer-Aided Design of User Interfaces VI (pp. 201–212). Springer London. https://doi.org/10.1007/978-1-84882-206-1_18

Greif, I. (2019). How we started CSCW. *Nature Electronics* 2019 2:3, 2(3), 132–132. https://doi.org/10.1038/s41928-019-0229-y

Hedjazi, D. (2018). Constructing collective competence: A new CSCW-based approach. *International Journal of Information and Communication Technology*, 12(3–4), 393–416. https://doi.org/10.1504/IJICT.2018.090418

Hmelo-Silver, C. E., & Jeong, H. (2021). An Overview of CSCL Methods. 65–83. https://doi.org/10.1007/978-3-030-65291-3_4

Koller, M., Bogdan, C., & Meixner, G. (2016). Collaborative task modeling: A first prototype integrated in Hamsters. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 9856 LNCS*, 366–373. https://doi.org/10.1007/978-3-319-44902-9_24

Kolovos, D. S., García-Domínguez, A., Rose, L. M., & Paige, R. F. (2017). Eugenia: towards disciplined and automated development of GMF-based graphical model editors. *Software and Systems Modeling*, 16(1), 229–255. https://doi.org/10.1007/s10270-015-0455-3

Martinie, C., Navarre, D., Palanque, P., & Fayollas, C. (2015). A generic tool-supported framework for coupling task models and interactive applications. *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems - EICS '15*, 244–253. https://doi.org/10.1145/2774225.2774845

Molina, A. I., Gallardo, J., Redondo, M. A., Ortega, M., & Giraldo, W. J. (2013). Metamodel-driven definition of a visual modeling language for specifying interactive groupware applications: An empirical study. *Journal of Systems and Software*, 86(7), 1772–1789. https://doi.org/10.1016/j.jss.2012.07.049

Molina, A. I., Redondo, M. A., & Ortega, M. (2009). A methodological approach for user interface development of collaborative applications: A case study. *Science of Computer Programming*, 74(9), 754–776. https://doi.org/10.1016/j.scico.2009.03.001

Molina, A. I., Redondo, M. A., Ortega, M., & Hoppe, U. (2008). CIAM: A Methodology for the Development of Groupware User Interfaces. *Journal of Universal Computer Science*, 14(9), 1435–1446. https://doi.org/10.3217/jucs-014-09-1435

Paternò, F. (2003). ConcurTaskTrees: An Engineered Notation for Task Models. In The Handbook of Task Analysis for Human-Computer Interaction (pp. 483–503). http://giove.cnuce.cnr.it/ctte.html

Schnaubert, L., & Bodemer, D. (2022). Group awareness and regulation in computer-supported collaborative learning. *International Journal of Computer-Supported Collaborative Learning*, 1–28. https://doi.org/10.1007/s11412-022-09361-1

Syriani, E., Vangheluwe, H., Mannadiar, R., Hansen, C., Mierlo, S. V., & Ergin, H. (2013). AToMPM: A web-based modeling environment. MODELS '13: Invited Talks, Demos, Posters, and ACM SRC, 1115.