Un ambiente de meta-modelado y visualización basado en el paradigma de Zoomable User Interfaces

A meta-modeling and visualization environment based on Zoomable User Interfaces

Jaime A. Pavlich-Mariscal[1] Hernan D. Veliz-Quispe[2]
Steven A. Demurjian[3] Laurent D. Michel[3]

Recibido 3 de febrero de 2012, aceptado 1 de julio de 2014
Received: February 3, 2012 accepted: July 1, 2014

RESUMEN

Un problema importante de la ingeniería de software es la visualización de modelos, para facilitar su comprensión y evolución. *Zoomable User Interfaces* (ZUI) es un paradigma con potencial de mejorar la visualización de modelos, el cual utiliza el zoom como medio para navegar entre diferentes niveles de abstracción. Este artículo, primer paso de una investigación en dicha dirección, describe un prototipo de herramienta meta-case, llamado ZooMEnv, para la definición e instanciación de sintaxis concretas (notación) de lenguajes visuales de modelado. El énfasis de la herramienta es la utilización de un meta-modelo pequeño para representar la notación de manera genérica y, como elemento novedoso, la incorporación de ZUI a través de jerarquías de composición y zoom semántico. Cinco notaciones diferentes son desarrolladas como casos de estudio para probar las capacidades de la herramienta y definir futuras mejoras.

Palabras clave: CASE, ZUI, Modelado de Software, UML, MOF, EMF, DSM.

ABSTRACT

Model visualization is an important problem in software engineering, to assist designers to understand and evolve models. Zoomable User Interfaces (ZUI), which uses zoom as the main mechanism to navigate between different levels of abstraction, has the potential to improve model visualization. This paper, the first step in this research direction, presents a prototype meta-case tool, called ZooMEnv, to define and utilize concrete syntaxes (notations) of visual modeling languages. The focus of the tool is to rely on a small meta-model to represent notation in a generic way and incorporate, as a novel concept, ZUI capabilities through composition hierarchies and semantic zooming. Five different notations are developed as case studies to explore the capabilities of this tool and determine future improvements.

Keywords: CASE, ZUI, Software Modeling, UML, MOF, EMF, DSM.

INTRODUCCIÓN

El desarrollo de software ha evolucionado significativamente durante los últimos años. Paradigmas tales como la programación orientada a objetos, por ejemplo, proveen mecanismos de modularización (clases) para manejar la complejidad. Frameworks de programación

[1] Departamento de Ingeniería de Sistemas. Pontificia Universidad Javeriana. Cra. 7 N° 40-62. Bogotá, Colombia. E-mail: jpavlich@javeriana.edu.co
[2] Departamento de Ingeniería de Sistemas y Computación. Universidad Católica del Norte. Angamos 0610. Antofagasta, Chile. E-mail: hvq001@ucn.cl
[3] Department of Computer Science & Engineering. The University of Connecticut. Storrs, CT 06269-2155. USA. E-mail: steve@engr.uconn.edu; ldm@engr.uconn.edu

implementan funcionalidades comúnmente utilizadas, permitiendo que los programadores se enfoquen en requerimientos del dominio del problema, en lugar de otros requerimientos [3]. Como resultado, los ingenieros de software han podido crear sistemas más grandes y complejos en menos tiempo [33].

A pesar de ello, los requerimientos actuales de software siguen aumentando en complejidad [35], por lo que es necesario explorar nuevas tecnologías para facilitar la evolución de grandes sistemas. Las notaciones visuales son una herramienta muy importante para dicho propósito. Una notación adecuadamente diseñada puede entregar información a ingenieros de software de manera intuitiva, facilitar los cambios y reducir los errores en la definición de los modelos [17]. La popularidad de notaciones como el *Lenguaje de Modelado Unificado* (Unified Modeling Language, UML) [25] se debe, en parte, al hecho que proveen una notación visual común para representar diferentes facetas del diseño [6].

Un elemento altamente relacionado con las notaciones visuales son las herramientas de *Ingeniería de Software Asistida por Computador* (Computer Aided Software Engineering, CASE), las cuales son aplicaciones de software que asisten a los ingenieros de software en la visualización y evolución de los modelos. Para obtener el mayor beneficio de una notación visual, las herramientas CASE deben proveer mecanismos para navegar adecuadamente a través de los modelos y meta-modelos, cambiar fácilmente entre diferentes niveles de abstracción y representar sólo aquella información que es relevante para cada uno de esos niveles. La mayoría de las herramientas CASE utilizan mecanismos comunes para visualizar la información. Cada modelo es mostrado mediante múltiples diagramas en lienzos (canvas) separados. Barras de desplazamiento se utilizan para moverse dentro de cada lienzo. Estructuras jerárquicas de árbol, usualmente visibles en barras laterales, se utilizan para navegar entre diferentes niveles de abstracción y diferentes diagramas del diseño [36-37, 42-43].

Mecanismos como los mencionados con anterioridad presentan importantes oportunidades de mejoramiento, principalmente a través de paradigmas no explotados de manera adecuada por herramientas CASE. Uno de dichos paradigmas es el de *Interfaces de Usuario Aumentables* (Zoomable User Interfaces, ZUI) [2], el cual tiene tres características esenciales: utiliza el zoom como mecanismo primario de navegación entre diferentes niveles de abstracción, utiliza movimiento panorámico (*panning*) para navegar entre entidades a un mismo nivel de abstracción y utiliza el zoom semántico [4] (variación en la representación de información basado en el nivel de zoom) para abstraer detalles en diferentes niveles. La incorporación de ZUI en la visualización de modelos de software podría mejorar sustancialmente su comprensión y evolución por varios motivos:

- ZUI permite explotar el razonamiento espacial del usuario. El despliegue de información en ZUI se asemeja a uso de planos (*blueprints*) utilizados en diseño en otras áreas de la ingeniería, lo cual refuerza la metáfora de interfaz de usuario.

- El zoom semántico permite automatizar la abstracción de detalles no relevantes de los modelos [4]. Esto facilitaría la interacción con los modelos, puesto que elimina la necesidad de ocultar manualmente la información no deseada, además de resaltar la información relevante.

- El uso de ZUI no excluye la utilización de otras técnicas de visualización, como los árboles de navegación, pudiendo coexistir en forma complementaria.

A pesar de estas ventajas, según el leal saber y entender de los autores, las principales herramientas CASE y Meta-CASE existentes no soportan el paradigma ZUI, o bien proveen un soporte muy limitado [11, 36, 37, 39, 42-44].

Este artículo, primer paso en esta línea de investigación, describe un prototipo de una aplicación meta-case que implementa los conceptos anteriores. Esta herramienta, llamada *Zoomable Modeling Environment (ZooMEnv)*, extiende la definición de la sintaxis concreta de lenguajes visuales, a través de un meta-modelo de sintaxis concreta que incluye, además de elementos notacionales básicos (representación de figuras geométricas y texto), mecanismos de abstracción basados en ZUI.

Si bien existen algunas herramientas que comparten ciertas características con ZooMEnv [13, 15, 20-21, 32, 39-41, 47], ninguna de las ellas incluye

las siguientes características simultáneamente: (a) Uso de zoom semántico para visualizar modelos en diferentes niveles de abstracción; (b) Inclusión de conceptos de ZUI en la especificación (meta-modelo) de sintaxis concretas; (b) Representación genérica de modelos y diagramas, lo cual permite una flexibilidad para representar diferentes tipos de lenguajes; (c) La capacidad de crear nuevas sintaxis concretas sin necesidad de modificar o agregar nuevo código a la aplicación.

El propósito de ZooMEnv es servir como prueba de concepto para explorar y entender los principales requerimientos técnicos de una herramienta de esta naturaleza. También se espera que ZooMEnv sirva de base para el diseño (en una versión futura de la herramienta) de un mecanismo de prototipado rápido de notaciones visuales basadas en ZUI. La premisa básica es que lo anterior se puede conseguir a través de un meta-modelo liviano, para la definición de sintaxis concretas, la incorporación de elementos de zoom semántico en el núcleo de la arquitectura del sistema y la utilización de un despliegue gráfico con capacidades de ZUI. Como tal, la validación de la herramienta se realiza en términos de probar su flexibilidad en la creación diferentes tipos de lenguajes de modelado, los cuales incorporan elementos de ZUI. Una validación a gran escala en un entorno de producción está fuera del alcance de este artículo.

El resto de este artículo describe los elementos esenciales alrededor de esta herramienta. Primero se describen conceptos de meta-modelado y ZUI. Después se describe la arquitectura general de ZooMEnv y las características de su diseño; se demuestran las capacidades de la herramienta a través de casos de estudio de diferentes notaciones. Luego, se describen trabajos relacionados. La última sección concluye el artículo y describe el trabajo futuro a realizar en esta línea de investigación.

META-MODELADO Y ZUI

Esta sección describe en detalle los conceptos de modelos y sintaxis concreta, además de los elementos esenciales de ZUI, requeridos para entender el resto del artículo.

Modelos, Meta-modelos y Sintaxis Concreta

A nivel de diseño, el uso de lenguajes de modelado presenta ventajas análogas a las que exhiben los lenguajes de programación de alto nivel: la capacidad de abstraer, en un número reducido de artefactos, una cantidad importante de sentencias de un lenguaje de menor nivel. Esto facilita la comprensión de sistemas de gran complejidad [18].

En el área de modelado existen dos paradigmas principales: *Arquitectura Dirigida por Modelos* (Model-Driven Architecture, MDA) [26] y *Modelado de Dominio Específico* (Domain Specific Modeling, DSM) [17]. El paradigma MDA se centra en definir la estructura y comportamiento de un sistema utilizando lenguajes de modelado, principalmente UML. A través de herramientas de software especializadas, transforma dichos modelos en una implementación en el lenguaje de programación requerido. En forma similar, el paradigma DSM se enfoca en definir modelos de dominio específico que luego serán traducidos a código. La diferencia esencial con MDA es el énfasis de DSM en el uso de lenguajes especializados en el dominio del problema.

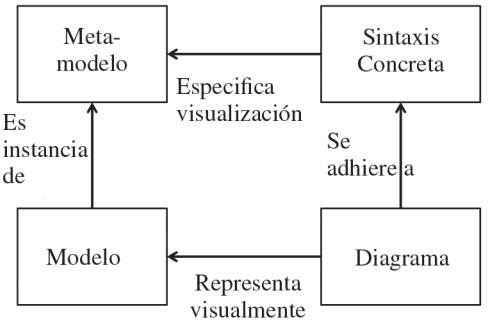

Figura 1. Terminología de Modelos y Meta-modelos. Representa todos los conceptos más importantes sobre modelado y sus relaciones entre sí.

Tanto MDA como DSM poseen conceptos comunes. Para presentar adecuadamente el resto del artículo, la Figura 1 unifica la terminología de ambos paradigmas. Los *Modelos*, especificados en UML (MDA) o en un lenguaje de dominio específico (DSM), representan una abstracción del problema a resolver y pueden incluir información específica del diseño de la solución Un modelo captura los elementos esenciales del problema a resolver y en el caso de MDA y DSM, puede sustituir total o parcialmente al código como elemento central de la solución. Los *Meta-Modelos*, también conocidos como *Sintaxis Abstracta*, representan la estructura

de los elementos y asociaciones que un grupo de modelos debe cumplir. La relación entre un modelo y un meta-modelo es similar a la relación entre una objeto y una clase, en el sentido que un modelo es una instancia de un meta-modelo.

Por ejemplo, el meta-modelo simplificado de UML [25], mostrado en la Figura 2, define la estructura que debe seguir un modelo de clases: las clases de dicho modelo corresponden a instancias de **Class** que contiene atributos (instancias de **Property**), métodos (instancias de **Operation**) y que tienen asociadas conjuntos de superclases. Cualquier modelo de clases creado en UML debe seguir la estructura definida en el meta-modelo de UML.

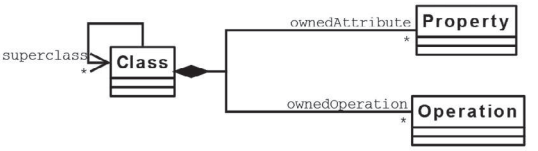

Figura 2. Meta-modelo simplificado de UML [25]. Una clase puede tener asociadas múltiples meta-classes y contener múltiples propiedades (atributos) y operaciones (métodos).

Una librería de uso común en el ámbito de modelado es el Framework de Modelado de Eclipse (Eclipse Modeling Framework, EMF) [7]. EMF provee un lenguaje, llamado *Ecore*, para la definición de la sintaxis abstracta de modelos. Ecore está basado en el Meta-Object Facility de MDA [26], un lenguaje orientado a objetos similar a UML que permite definir meta-modelos de otros lenguajes de modelado, entre ellos UML [25]. Para poder utilizar los meta-modelos creados en EMF, se requiere la creación de un modelo intermedio, llamado *Modelo Generador*, el cual especifica un mapeo desde el meta-modelo Ecore a un conjunto de clases en Java. Las clases Java generadas en este proceso proveen la estructura programática para crear instancias de dichos meta-modelos.

Los modelos y meta-modelos, por sí solos, no requieren explícitamente el uso de ninguna notación particular para su representación. La *Sintaxis Concreta* (ver Figura 1) especifica cómo representar visualmente los modelos a través de los *Diagramas.* La sintaxis concreta consiste en un conjunto de *plantillas*, donde cada plantilla especifica la representación visual de cada clase de un meta-modelo. En general, las herramientas CASE más difundidas han definido sus propias estructuras de representación visual [22, 36-37, 42-43], enfocadas en la representación de figuras geométricas, texto y conexiones entre nodos. Adicionalmente, el estándar de OMG para intercambio de diagramas, UML-DI [24] define un medio para representación de notaciones gráficas, aunque no incluye explícitamente elementos de ZUI.

Para simplificar la nomenclatura, el resto del artículo utilizará las siguientes convenciones:

- El término “clase” se utilizará para referirse a los elementos de un meta-modelo. El artículo evita el uso de la palabra “meta-clase” para no sobrecargar la nomenclatura.
- Un “modelo” es una instancia de un meta-modelo. Asimismo, el término “elemento de modelo” se refiere a la instancia de una clase de un meta-modelo.
- El término “elemento de sintaxis concreta” o, en forma más simple, “elemento de sintaxis”, se refiere a los componentes atómicos de una sintaxis concreta. Las “plantillas” son conjuntos de uno o más elementos de sintaxis que indican la forma de representar instancias de una clase de meta-modelo en un diagrama.
- El término “notación” se utilizará como sinónimo de “sintaxis concreta”.
- Un “diagrama” es una “instancia de una sintaxis concreta”. Asimismo un “elemento de diagrama” es una instancia de un elemento de sintaxis.

Interfaces de usuario aumentable

La Interfaz de Usuario Aumentable, o en inglés, *Zoomable User Interface (ZUI)*, es un paradigma de interacción con el usuario, basado principalmente en el despliegue de información con diferentes niveles de aumento (zoom). ZUI tiene tres características distintivas:

1. Utiliza el zoom como mecanismo primario de navegación a través de diferentes niveles de abstracción
2. Utiliza el zoom semántico (variación en la representación de información basado en el nivel de zoom) [14] para abstraer detalles en diferentes niveles.
3. Utiliza el movimiento panorámico (*panning*) para navegar entre entidades a un mismo nivel

de abstracción. En otras palabras, el lienzo puede ser desplazado hacia arriba, abajo o hacia los lados, sin variar el nivel de aumento, para navegar a través del modelo.

En el ámbito de gráficos estructurados, elemento esencial para la visualización de modelos, existen varios esfuerzos [10, 20, 29-30, 32, 34, 45]. Uno de las más importantes es *Piccolo2d* [45], una librería en Java para la creación de gráficos estructurados en dos dimensiones. Esta librería abstrae procesos de dibujo en pantalla, manejo de eventos, selección de nodos en pantalla, animación y encuadre, entre otros. Estas herramientas, aunque utilizan ZUI en forma intensiva, no resuelven directamente el problema de visualización de modelos de software, sino que proveen la base para implementar herramientas de visualización.

LA HERRAMIENTA ZOOMENV

El elemento central de este artículo es la aplicación ZooMEnv, una herramienta de modelado y meta-modelado con un propósito específico: proveer un mecanismo de creación rápida de editores de modelos con sintaxis concretas basadas en ZUI. Bajo esta perspectiva, el diseño de la aplicación está centrado en los siguientes principios básicos:

1. **Sintaxis concreta con un número mínimo de primitivas.** El supuesto es que, mientras menor el número de primitivas de la sintaxis concreta, más simple es su comprensión y su utilización por parte de los usuarios finales.
2. **Visualización de modelos mediante diferentes sintaxis concretas.** Facilita el prototipado de sintaxis concretas, pues permite probar diferentes notaciones para una misma sintaxis abstracta.
3. **Evitar que la creación de modelos dependan de la generación de código.** Permite que ZooMEnv sea autocontenido y la generación de editores de modelos no requiera compilar código adicional.
4. **Proveer de una arquitectura que facilite la extensión de su funcionalidad en el tiempo.** Permite cubrir futuras necesidades no resueltas en la funcionalidad actual de ZooMEnv.

ZooMEnv está compuesto de dos herramientas: Un *Editor de Sintaxis Concretas* (Figura 3) y un *Editor de Diagramas* (Figura 4). El primero sirve para crear

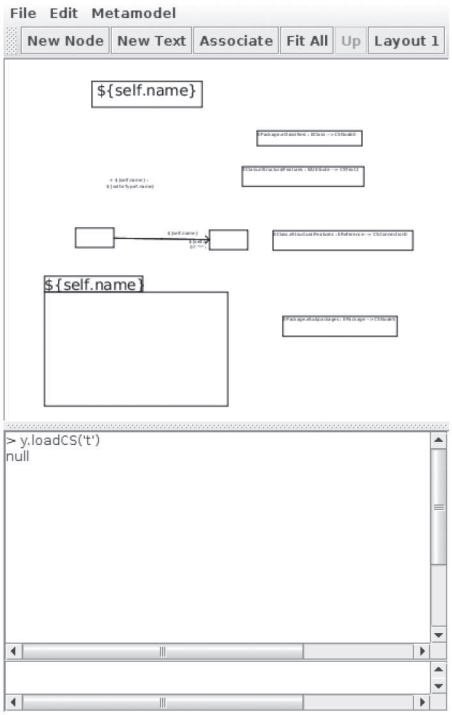

Figura 3. Editor de Sintaxis. Herramienta de ZooMEnv para editar notaciones.

la notación de un meta-modelo. El segundo permite crear instancias de dicho meta-modelo, utilizando dicha notación para representarlas visualmente. Ambas herramientas utilizan Swing [28] para implementar la GUI y proveen un lienzo (canvas) para el dibujo de los diagramas, el cual está basado en la librería Piccolo2D [45]. Para interactuar con los diagramas, las herramientas permiten el uso del mouse, los menús y una interfaz de línea de comandos.

La Figura 5 describe la arquitectura general de la aplicación, la cual comprende un conjunto de módulos principales:

- *Administrador de Modelos*, basado en EMF, administra la creación y evolución de modelos y sus meta-modelos utilizando archivos Ecore.
- *Administrador de Sintaxis Concreta*, basado en EMF, almacena las sintaxis concretas a través de archivos Ecore y asocia los elementos de sintaxis a las clases de los meta-modelos para facilitar su representación en los diagramas.
- *Visualización ZUI*, basado en Piccolo2d, se encarga de visualizar en pantalla los elementos de sintaxis concreta y de diagramas.

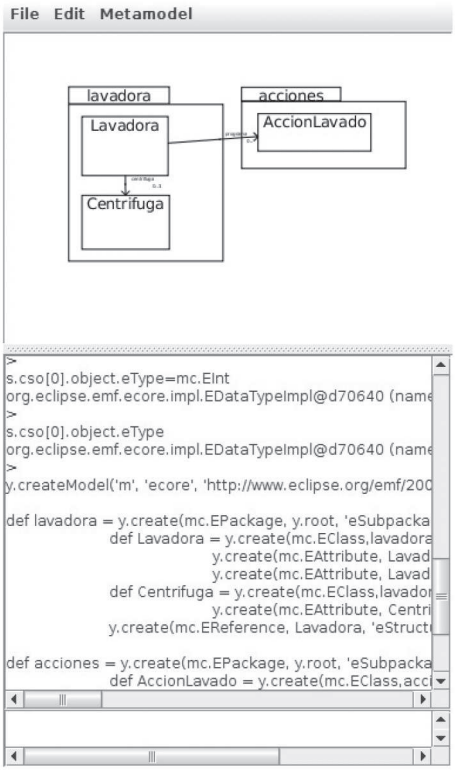

Figura 4. Editor de Modelos. Herramienta de ZooMEnv que permite crear modelos usando la notación definida por el Editor de Sintaxis.

Conectando los tres módulos anteriores hay dos capas intermedias, basadas en el patrón observador [9], que eliminan las dependencias directas entre dichos módulos:

- El *Enlace Sintaxis Concreta ↔ Modelo* se encarga de mantener en sincronía las clases de EMF que almacenan la información de la sintaxis concreta con las clases que representan el modelo que está siendo modificado por el Editor de Modelos.
- El *Enlace Vista ↔ Sintaxis Concreta* se encarga de mantener la sincronía entre las clases que almacenan la sintaxis concreta con las clases que las representan visualmente en Piccolo2D.

En forma transversal, ZooMEnv utiliza Groovy [19] como lenguaje incrustado. Esto permite que el intérprete de la línea de comandos acceda a las APIs de todos los módulos.

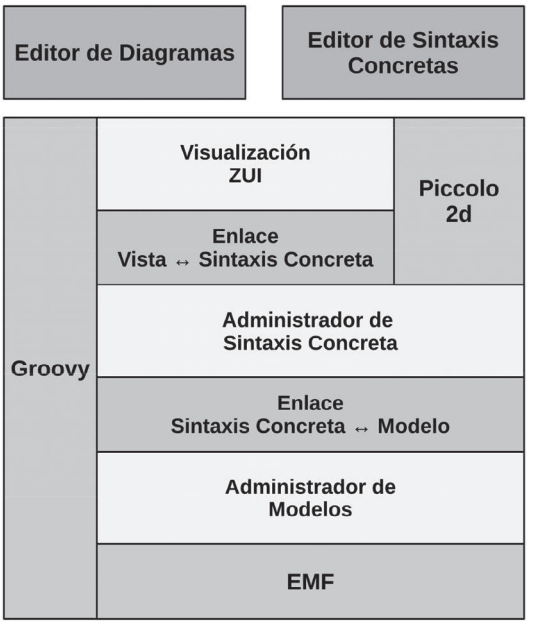

Figura 5. Arquitectura de ZooMEnv. La aplicación comprende una serie de capas de funcionalidad, unificadas a través de un intérprete de lenguaje Groovy.

El resto de esta sección describe el diseño de ZooMEnv en detalle: El Administrador de Modelos, el Administrador de Sintaxis Concreta, la Visualización ZUI, el soporte de ZUI en la sintaxis concreta y la gestión de plantillas del Administrador de Sintaxis Concreta.

Administrador de Modelos

Este módulo provee un conjunto de clases que simplifican el manejo de modelos en EMF. Uno de los objetivos principales de este módulo es evitar la dependencia de EMF hacia la generación de código para la creación de meta-modelos a partir de un archivo Ecore. Para ello, el Administrador de Modelos utiliza intensivamente las capacidades de EMF dinámico [7], un subconjunto de APIs de EMF que permiten la creación e instanciación de meta-modelos en tiempo de ejecución. EMF dinámico no requiere la definición de tipos de datos previos al proceso de compilación, por lo que es posible prescindir de los Modelos Generadores de EMF.

En la práctica, el uso de EMF dinámico significa que ZooMEnv puede cargar meta-modelos en memoria simplemente abriendo los archivos Ecore que los

contienen y creando modelos en forma directa. ZooMEnv permite guardar dichos modelos en formato XMI [27], utilizando las clases que provee EMF para dicho propósito.

Sintaxis concreta y visualización ZUI

Los módulos de Administración de Sintaxis Concreta y Visualización ZUI están altamente relacionados. El primero provee las estructuras que almacenan en forma persistente la estructura de los diagramas y la sintaxis concreta a la cual están adscritos, además de gestionar las plantillas requeridas para representar un modelo en un diagrama. El segundo permite visualizar, en forma uno a uno, los diagramas almacenados en el administrador de sintaxis concreta.

Una parte importante del módulo de Administración de Sintaxis Concreta es el meta-modelo de la Figura 6, el cual describe la estructura que común de todos los diagramas generados con ZooMEnv. La clase **CSElement** representa las características comunes de todos los elementos de sintaxis concreta. Instancias de **CSElement** pueden tener, entre otros atributos, colores (**CSColor**), formas (**CSShape**), transformaciones (**CSTransform**), encuadres (**CSLayout**) y estilos de línea (**CSStroke**).

Las subclases de **CSElement** permiten representar elementos de notación más especializados. **CSNode**, **CSConnection** y **CSConnectionEnd** permiten representar estructuras de grafos. **CSText** representa textos con formato. Las instancias de **CSRoot** son únicas dentro de un diagrama y representan la raíz que contiene a todos los demás elementos.

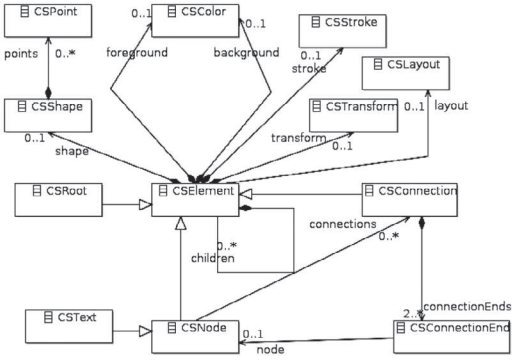

Figura 6. Meta-modelo de sintaxis concretas. Muestra todas las meta-clases esenciales para representar la sintaxis de los elementos de diagramas.

La Visualización ZUI se encarga de desplegar instancias de este meta-modelo, utilizando la librería Piccolo2D [45] para generar tres tipos principales de vistas: la vista de nodos, la vista de conexiones y la vista de texto. Las vistas de nodos despliegan la información de las instancias de **CSNode**, tal como muestra Figura 7. Una vista de nodo puede asumir diferentes formas geométricas y, puede contener otras vistas dentro de ella.

Las vistas de conexiones despliegan la información de **CSConnection** y **CSConnectionEnd** a través de líneas que asocian nodos en un grafo. Estas vistas pueden contener otros elementos en su interior, tales como texto, lo que permite representar nombres de asociaciones, roles y cardinalidades, entre otras cosas. Los extremos de las conexiones pueden desplegar diferentes flechas y son redibujados automáticamente al mover o cambiar el tamaño de los nodos de un grafo.

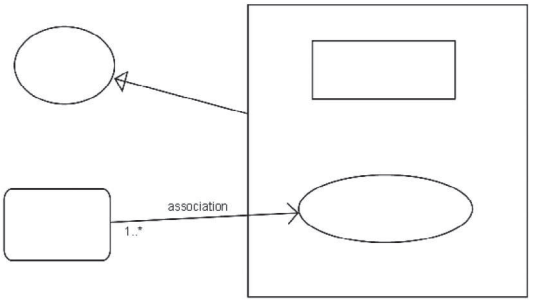

Figura 7. Vistas de nodos y conexiones. ZooMEnv puede representar las vistas de nodos y conexiones mediante diferentes formas geométricas y texto.

Las vistas de texto despliegan la información de **CSText**. Estas vistas incrustan código Groovy para actualizar dinámicamente el texto desplegado a base de la información contenida en los modelos. La Figura 8 muestra una vista de texto dentro de la plantilla de un paquete UML. El nombre del paquete está representado por la vista de texto con el código Groovy **${self.name}**. El motor de Groovy interpreta este código en tiempo de ejecución como el nombre (atributo **name**) del paquete.

Soporte de ZUI en la sintaxis concreta

El meta-modelo de sintaxis concreta soporta el paradigma ZUI a través de dos características. Primero, las instancias de **CSElement** (ver Figura 6)

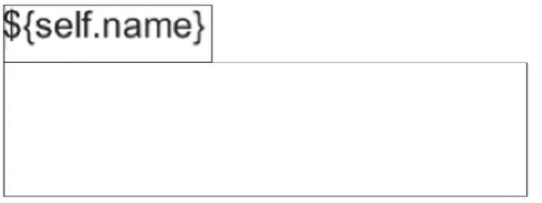

Figura 8. Vista de texto en una plantilla de un paquete UML. Esta plantilla de un paquete UML contiene una etiqueta de texto en lenguaje Groovy, que despliega el atributo **name** del paquete.

forman una jerarquía de composición, donde cada elemento de sintaxis concreta puede contener una serie de elementos hijos. Esto permite, por ejemplo, definir la sintaxis concreta de paquetes de UML, donde el rectángulo que representa el cuerpo del paquete puede contener otros elementos, tales como clases u otros paquetes. Asimismo, este esquema puede representar notaciones como las de las clases de UML, donde el rectángulo principal que representa la clase contiene elementos hijos: el texto con el nombre de la clase y los compartimientos de atributos y métodos.

Adicionalmente **CSElement** incluye capacidades de zoom semántico a través de los atributos **minZoom** y **maxZoom** (no mostrados en la Figura 6). Estos atributos son valores decimales que determinan la visibilidad del correspondiente elemento de diagrama a diferentes niveles de zoom. Dado un valor de zoom en el lienzo donde se dibuja el diagrama, un elemento de diagrama será visible sólo si el valor del zoom actual es mayor o igual que **minZoom** y menor que **maxZoom**.

Estas dos caraterísticas, jerarquías de composición de elementos de sintaxis y zoom semántico, proveen la base para la definición de múltiples notaciones basadas en ZUI. Por un lado, las jerarquías de composición son un elemento inherentemente representable en ZUI, mientras que el zoom semántico permite ocultar y mostrar los elementos de dicha jerarquía, basado en el nivel de aumento en que son visualizados.

Gestión de plantillas

Además de proveer almacenamiento persistente a los diagramas creados en el Editor de Modelos, el meta-modelo de la Figura 6 se utiliza para gestionar las plantillas de la sintaxis concreta de un meta-modelo.

Una plantilla es una jerarquía de instancias de **CSElement** que define la representación visual de una clase de un meta-modelo. ZooMEnv fue diseñado para otorgar flexibilidad en la definición de plantillas, siempre y cuando se cumplan los siguientes requisitos:

- Todos los elementos de un modelo deben estar contenidos dentro de otro a través de una relación de composición [25] (con excepción del elemento raíz del modelo). En la práctica, este requerimiento se cumple siempre, dado que es una restricción impuesta por EMF [7].
- Las instancias de una clase contenida dentro de otro elemento de modelo son representadas siempre con la misma plantilla. En otras palabras, para representar en forma diferente dos instancias de una misma clase en un diagrama (usando plantillas distintas), estas instancias deben estar contenidas por otras instancias de clases que sean diferentes entre sí.

La Figura 9 muestra la parte del meta-modelo de sintaxis concreta encargada de la gestión de plantillas. La clase **CSTemplateDescription** es el puente entre las plantillas y las clases cuyas instancias serán visualizadas en los diagramas. **CSTemplateDescription** comprende cuatro elementos:

- **theClass**, la clase del meta-modelo, cuyas instancias se quiere representar con la plantilla.
- **containerClass**, la clase contenedora, cuyas instancias contienen a instancias de **theClass**.
- **containerStructuralFeature**, el atributo **containerClass** que referencia a la instancia de **theClass**.
- **template**, la plantilla que representará a la instancia de **theClass** en el diagrama de Figura 9.

CREACIÓN DE EDITORES DE MODELOS MEDIANTE ZOOMENV

Esta sección describe varios editores creados para probar el nivel de flexibilidad de ZooMEnv en la creación e instanciación de sintaxis concretas: un editor de mapas conceptuales, un editor de diagramas de componentes en UML, un editor de diagramas

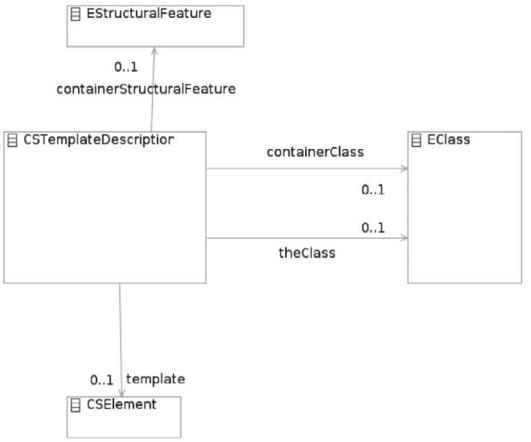

Figura 9. Descripción de plantillas en el meta-modelo de sintaxis concreta. Este es un subconjunto del meta-modelo de sintaxis-concreta que indica las entidades utilizadas para asignar plantillas a elementos de meta-modelos.

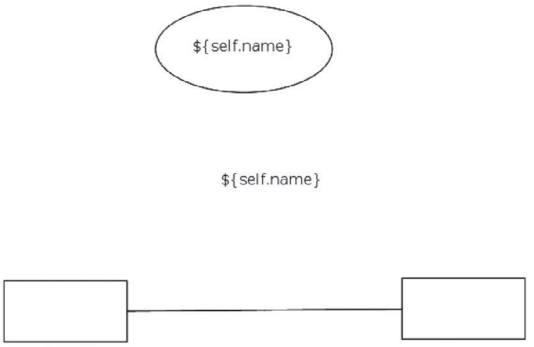

Figura 10. Sintaxis concreta de mapas conceptuales. Las plantillas que describen la vista de ideas principales, secundarias y las asociaciones entre ideas.

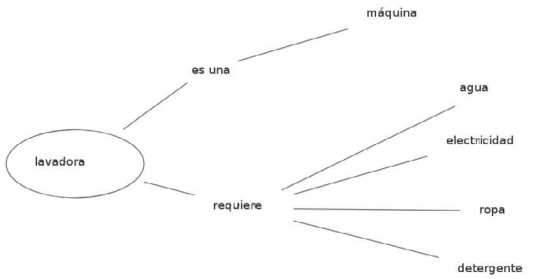

Figura 11. Mapa mental. Describe un diagrama de ejemplo construido a partir de la sintaxis concreta de la Figura 10.

de flujo de datos, un editor de modelos Ecore y un editor de máquinas de transición de estados. Los editores de modelos son presentados en orden de complejidad (de menor a mayor), para ilustrar progresivamente las capacidades de ZooMEnv.

Editor de mapas conceptuales

Para ilustrar las capacidades más básicas de ZooMEnv, se creó un editor sencillo de mapas conceptuales, los cuales contienen ideas principales, representadas como elipses con texto, ideas secundarias, representadas por textos sin figuras geométricas que lo contengan, y asociaciones entre ideas, a través de líneas continuas. La Figura 10 muestra la sintaxis concreta, la cual contiene tres plantillas. La primera plantilla corresponde a la idea principal del diagrama, el texto **${self.name}** dentro de la elipse corresponde a código en Groovy que despliega el nombre de la idea. El texto **${self.name}** a continuación de la elipse es la plantilla para ideas secundarias. La línea que conecta dos rectángulos representa las asociaciones entre ideas. Los rectángulos conectados por la línea no forman parte de la sintaxis concreta; sólo sirven para poder crear la conexión.

La Figura 11 muestra un ejemplo de mapa conceptual construido con la plantilla de la Figura 10. Este ejemplo, aunque no muestra las capacidades de zoom semántico de la herramienta, ilustra cómo las sintaxis concretas se utilizan para generar un diagrama. La idea principal **lavadora** es representada con una elipse (correspondiente a la primera plantilla de la Figura 10). Las ideas secundarias sólo contienen texto (ver segunda plantilla de la Figura 10) y están conectadas por líneas continuas (tercera plantilla de la Figura 10).

Editor de modelos de componentes

Para ilustrar los elementos básicos de zoom semántico de ZooMEnv, la Figura 12 y Figura 13 muestran la sintaxis concreta de un diagrama de componentes de UML [25], a dos niveles distintos de zoom. La Figura 12 muestra la sintaxis concreta a un nivel menor de zoom, donde la primera plantilla representa un componente con su nombre y la segunda plantilla es la relación de dependencia entre componentes.

La Figura 13 muestra la sintaxis concreta a un mayor nivel de zoom. La diferencia principal está en la

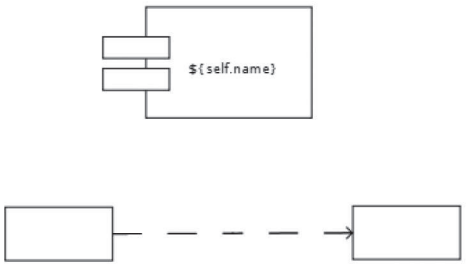

Figura 12. Plantillas con zoom reducido de la sintaxis concreta de modelos de componentes. Las plantillas describen la vista de un componente y una dependencia.

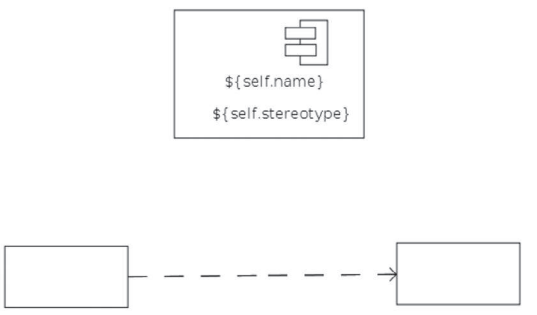

Figura 13. Plantillas con mayor zoom de la sintaxis concreta de modelos de componentes.

representación del componente, el cual agrega un ícono en la parte superior derecha y el estereotipo en la parte inferior.

La Figura 13 describe los mismos componentes que la Figura 12, a un mayor nivel de zoom.

La Figura 14 muestra un ejemplo de diagrama de componentes. La Figura 14a muestra el diagrama a un nivel reducido de zoom, donde los componentes sólo muestran sus nombres. La Figura 14b muestra el diagrama a un mayor nivel de zoom, donde el componente **LavadoraGUI** muestra el estereotipo **«GUI»**. Los demás componentes no poseen estereotipo. Si lo tuviesen, éstos también serían desplegados en la Figura 14b.

La Figura 14a muestra el diagrama a un menor nivel de zoom. La Figura 14b muestra el diagrama a un mayor nivel de zoom.

La Figura 15 muestra la notación a sólo un nivel de zoom, ya que la representación de los elementos de diagrama no varía con el nivel de aumento.

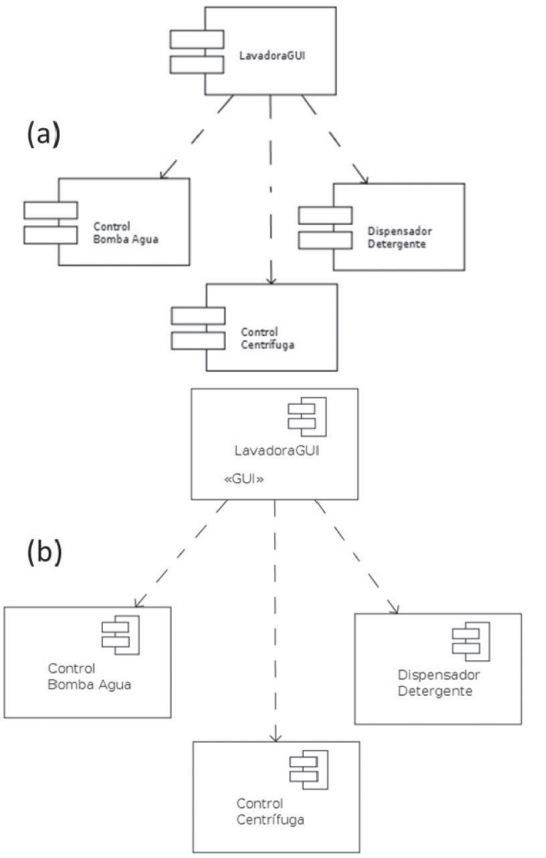

Figura 14. Diagrama de componentes a dos niveles de zoom.

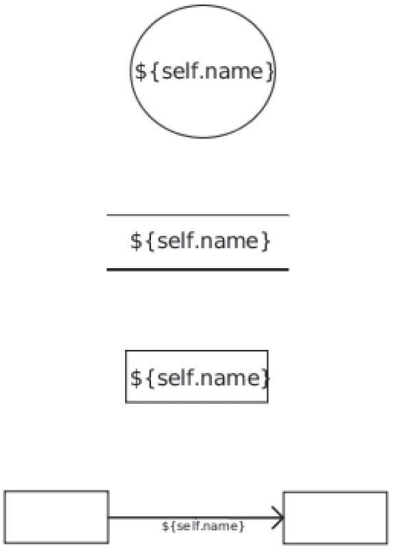

Figura 15. Sintaxis concreta de DFDs.

Editor de diagramas de flujo de datos (DFD)
Otro caso desarrollado para probar ZooMEnv es un editor de DFDs. La Figura 15 muestra la sintaxis concreta de un DFD, según la notación de Yourdon [38] que incluye procesos, almacenes de datos, entidades externas y flujos de datos.

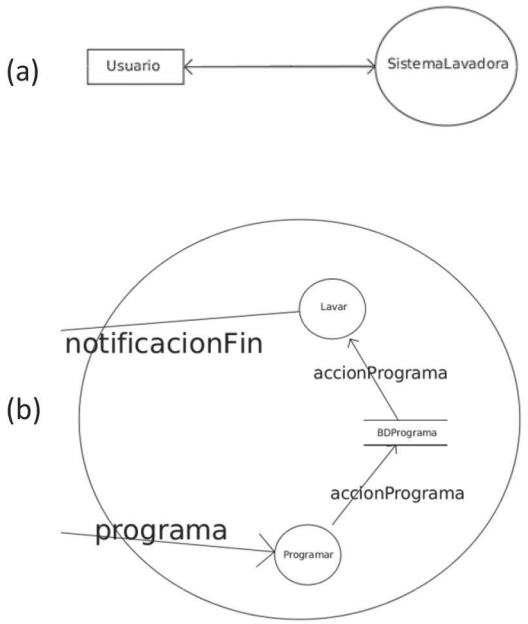

Figura 16. Un diagrama de flujo de datos.

La Figura 16a muestra el diagrama a menor nivel de zoom. La Figura 16b muestra el diagrama a mayor nivel de zoom.

La Figura 16 muestra un DFD de ejemplo. La Figura 16a es el DFD con un zoom reducido. El Editor de Modelos lo muestra automáticamente como un diagrama de contexto. La Figura 16b muestra una vista aumentada del sistema (el nodo **SistemaLavadora**), donde se aprecian mayores detalles de sus procesos y almacenes de datos.

Editor de modelos Ecore
Un elemento clave de EMF es que la sintaxis abstracta de meta-modelos Ecore también está definida en Ecore. Esto significa que las APIs y herramientas que crean modelos basados en Ecore también pueden crear los meta-modelos subyacentes. Aprovechando esta característica, se utilizó ZooMEnv para definir la sintaxis concreta del meta-modelo de Ecore, lo cual permite al Editor de Diagramas crear meta-modelos. La Figura 17 y Figura 18 muestran parte de la sintaxis concreta de Ecore, a dos niveles distintos de zoom. La Figura 17 muestra las plantillas con un nivel menor de zoom, donde el primer rectángulo es la plantilla de una clase Ecore, de la cual se muestra sólo el nombre. La flecha representa asociaciones entre dos clases. El código Groovy

${self.lowerBound}..${self.upperBound<0? "*" : self.upperBound}

despliega la cardinalidad de la asociación.

La última plantilla en la figura representa los paquetes de un meta-modelo Ecore.

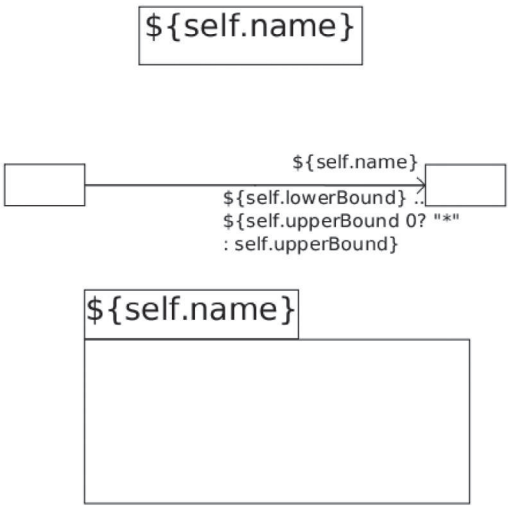

Figura 17. Plantillas con zoom reducido de la sintaxis concreta de Ecore. Las plantillas describen la vista de una clase, una asociación y un paquete en Ecore.

La Figura 18 muestra las plantillas para un nivel mayor de zoom de los mismos elementos que la Figura 17.

La Figura 18 muestra las plantillas con un nivel mayor de zoom. La plantilla de clases Ecore tiene compartimientos para los atributos. Estos últimos son representados usando la plantilla que está a la derecha de la plantilla de clases Ecore.

El código en Groovy

${self.name} : ${self.eType?.name}

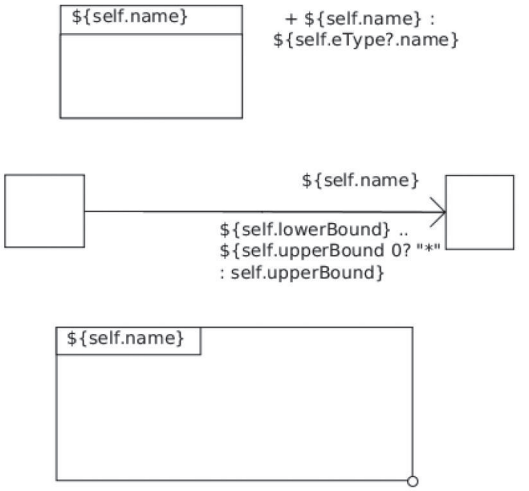

Figura 18. Plantillas con mayor zoom de la sintaxis concreta de Ecore.

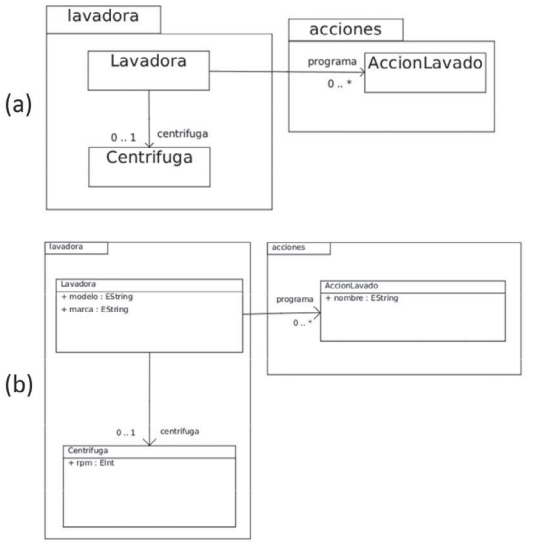

Figura 19. Diagrama Ecore a dos niveles de zoom.

permite desplegar el nombre y el tipo del atributo. El rectángulo de la parte inferior es una vista cercana de los paquetes, donde la etiqueta del nombre está por dentro del rectángulo que contiene los elementos hijos del paquete.

La Figura 19 muestra un ejemplo de diagrama Ecore creado con el Editor de Modelos. La Figura 19a es la vista lejana del modelo, donde los paquetes contienen clases que sólo muestran sus nombres. En la Figura 19b, las clases muestran sus atributos y los paquetes cambian de formato.

La Figura 19a es un diagrama Ecore a menor nivel de zoom. La Figura 19b es un diagrama Ecore a un mayor nivel de zoom.

Editor de Máquinas de Transición de Estados

Por último, se creó un editor de diagramas de transición de estados. La Figura 20 muestra la sintaxis concreta a dos diferentes niveles de zoom. Los nombres de estados y transiciones aparecen con nombres más grandes en la Figura 20a, mientras que en la Figura 20b los estados tienen un compartimiento para mostrar sub-estados.

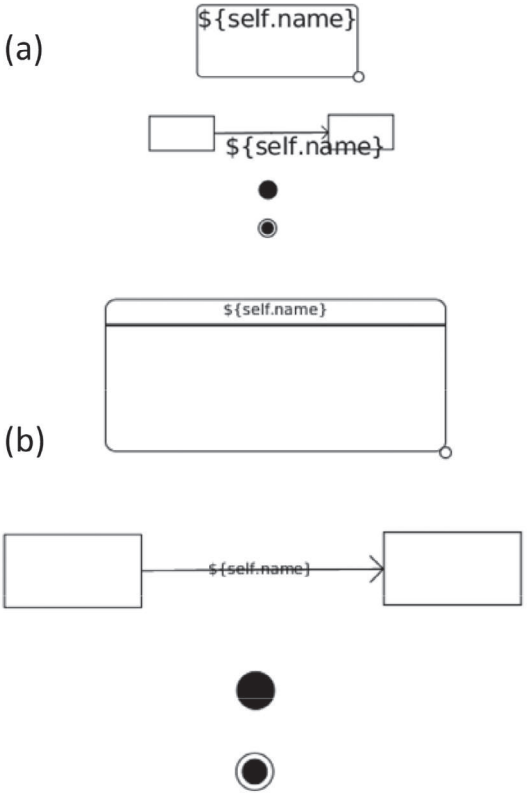

Figura 20. Notación para máquinas de transición de estados.

La Figura 20a es describe la notación para un nivel de zoom menor. La Figura 20b describe la notación para un nivel de zoom mayor.

La Figura 21 refleja esta sintaxis en una máquina de transición de estados. La Figura 21a muestra una vista con menor zoom del diagrama, mientras que la Figura 21b muestra el detalle del estado **funcionando**, el cual contiene otros sub-estados que están conectados con estados principales.

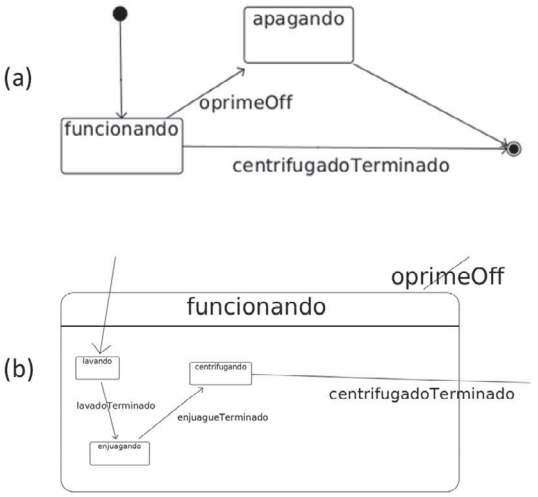

Figura 21. Diagrama de transición de estados.

La Figura 21a muestra el diagrama a menor nivel de zoom. La Figura 21b muestra el diagrama a mayor nivel de zoom.

TRABAJOS RELACIONADOS

En el ámbito de modelado de software existen varias herramientas CASE [11, 22, 36-37, 42-43, 46] que permiten crear diagramas, principalmente en UML, para diseñar sistemas de software. Algunas de estas herramientas ofrecen capacidades para crear meta-modelos y sintaxis concretas [11, 22, 36, 46]. Sin embargo, ninguna de ellas, según el leal saber y entender de los autores, extiende la definición de sintaxis concreta con ZUI en la forma propuesta en este artículo.

En el ámbito de visualización de modelos de software, la herramienta SHRIMP [41] permite visualizar la estructura de código en Java utilizando el paradigma ZUI. A diferencia de ZooMEnv, SHRIMP se enfoca solamente en la visualización de modelos y no provee mecanismos de definición de sintaxis concreta.

La herramienta AVISPA [15] es una herramienta de visualización y análisis de modelos de procesos, basada en la herramienta Moose [48]. A diferencia de AVISPA, ZooMEnv es genérica respecto del tipo de modelo a visualizar, lo cual se refleja en las múltiples diferentes notaciones y modelos que ZooMEnv soporta, mostrados en la sección previa de este artículo. Además, ZooMEnv tiene un énfasis directo en el uso de zoom semántico para navegar en diferentes niveles de abstracción. AVISPA, por otro lado, se enfoca en el análisis de los modelos, aspecto que está fuera del alcance de ZooMEnv.

Otras herramientas similares permiten la visualización de diferentes dominios. IsaViz es una herramienta para visualizar y manipular modelos que usen el estándar RDF [40]. ZGRViewer es una herramienta para visualizar grafos [47]. Squidy es una herramienta de visualización interactiva utilizando hardware especializado y diferentes técnicas de interacción humano computador [20]. Eagle Mode es un entorno de escritorio basado en ZUI, donde la navegación entre aplicaciones se realiza a través de diferentes niveles de aumento [13]. ADORA [32] es una herramienta para visualización jerárquica de modelos que ha sido utilizada principalmente para modelar líneas de productos de software [31]. Aunque todas estas herramientas proveen elementos básicos de ZUI y zoom semántico, ninguna de ellas tiene la capacidad de creación de nuevas sintaxis concretas con el nivel de flexibilidad propuesto por ZooMEnv.

Otra trabajo similar a ZooMEnv es MetaDepth [21], una herramienta de meta-modelado. MetaDepth permite crear nuevas sintaxis abstractas y concretas, pero sin capacidades de zoom semántico.

El trabajo que podría tener la mayor relación con el presente artículo es Pounamu [39], una herramienta meta-case con capacidades de definición de sintaxis concretas. Pounamu incluye soporte parcial de ZUI, a través un plug-in que presenta, en una misma ventana, varias perspectivas con diferentes niveles de zoom [23]. La herramienta ZooMEnv, en contraste, fue diseñada desde cero con un énfasis en ZUI. A nivel de interacción con el usuario, esto se refleja en el uso de ZUI en la interfaz gráfica. Adicionalmente, a nivel de los modelos, ZooMEnv posee capacidades intrínsecas de ZUI en su meta-modelo de sintaxis concreta.

En resumen, las principales diferencias de ZooMEnv con trabajos existentes es que ningún trabajo relacionado incluye las siguientes características simuláneamente: (a) La incorporación de elementos de jerarquías de composición y zoom semántico en la visualización de modelos; (b) El uso de conceptos de ZUI dentro de la especificación (meta-modelo) de sintaxis concretas; (c) El enfoque genérico para

representar modelos y diagramas, lo cual permite una flexibilidad para representar diferentes tipos de lenguajes; (d) La capacidad de la herramienta para crear nuevas sintaxis concretas sin necesidad de modificar o agregar nuevo código a la aplicación.

CONCLUSIONES Y TRABAJO FUTURO

Este artículo describió la herramienta ZooMEnv, un ambiente de modelado y definición de sintaxis concreta basado en el paradigma ZUI. Las notaciones de prueba desarrolladas en este artículo muestran que la arquitectura de esta aplicación, aunque relativamente simple, entrega un grado de flexibilidad en la definición de múltiples sintaxis concretas. En particular la capacidad para almacenar jerarquías de composición de elementos visuales y el zoom semántico permiten abstraer o mostrar detalles de elementos de modelo compuestos. Esta característica ha probado ser útil en la comprensión de modelos complejos [5, 12, 14].

A partir de las sintaxis concretas desarrolladas durante este trabajo, se determinó que la herramienta ZooMEnv puede ser mejorada en varios aspectos. Por un lado, el nivel de zoom en nodos pequeños puede hacer difícil de leer el texto que contienen. La presentación de información podría mejorar significativamente al mantener un tamaño consistente de letra en las etiquetas de los nodos y conexiones.

Otro aspecto importante es la visualización de nodos vecinos. En el caso de la Figura 21b los estados adyacentes al nodo **funcionando** no son visibles, ya que las transiciones salientes se dirigen hacia afuera del campo de visión del lienzo. En forma similar, en la Figura 16b la entidad externa **Usuario** tampoco es visible en ese nivel de zoom. Futuras mejoras en esta área requiere concebir métodos para hacer visible nodos adyacentes dentro del mismo lienzo.

Adicionalmente, un aspecto importante para mejorar la herramienta es la incorporación de organización de nodos (*layout*) y conexiones en el lienzo. Se hace necesario organizar de forma automática los nodos dentro de cada contenedor, como por ejemplo en los modelos de transición de estados (ver Figura 21). Si bien existen algoritmos para dicho propósito [1, 8, 16], el paradigma ZUI introduce una serie de nuevos requerimientos: nodos que contienen grafos en su interior, conexiones desde sub-nodos hacia nodos externos, organización de nodos dependiente del nivel de zoom, entre otros. Estos nuevos requerimientos podrían requerir la creación o adaptación de algoritmos de organización existentes.

Se espera, en un futuro cercano, completar el desarrollo de la versión de producción de la herramienta, la cual debería incorporar las mejoras anteriormente señaladas. Esta versión incorporaría también mejoras en usabilidad, en particular para la creación y edición de modelos, lo cual actualmente depende de manera importante de la interacción con el intérprete Groovy. Una vez esté lista esta nueva versión, parte del trabajo futuro es, a través de experimentos, estudiar con mayor detalle las mejoras en usabilidad entregadas por ZUI de tal forma de identificar nuevas oportunidades de desarrollo e investigación.

REFERENCIAS

[1] G. di Battista. “Graph Drawing: Algorithms for the Visualization of Graphs”. Prentice Hall. 1999. ISBN: 978-0133016154.

[2] B.B. Bederson, J. Grosjean and J. Meyer. “Toolkit design for interactive structured graphics”. IEEE Transactions on Software Engineering. Vol. 30, pp. 535-546. 2004. ISSN: 0098-5589. DOI: 10.1109/TSE.2004.44.

[3] G. Booch. “Object-Oriented Analysis and Design with Applications”. Addison-Wesley Professional. 1993. ISBN: 0805353402.

[4] M.N.K. Boulos. “The use of interactive graphical maps for browsing medical/health Internet information resources”. International Journal of Health Geographics. Vol. 2, pp. 1-14. 2003. DOI: 10.1186/1476-072X-2-1.

[5] T.T.A. Combs and B.B. Bederson. “Does zooming improve image browsing?”. Proceedings of the fourth ACM conference on Digital libraries. Berkeley, California, United States. ACM, pp. 130-137. 1999. DOI: 10.1145/313238.313286.

[6] B. Dobing and J. Parsons. “How UML is used”. Commun. ACM. Vol. 49, pp. 109-113. 2006. ISSN: 0001-0782. DOI: 10.1145/1125944.1125949.

[7] Eclipse Foundation. “Eclipse Modeling Framework (EMF)”. 2013. Date of visit: August 11, 2010. URL: http://www.eclipse.org/modeling/emf/

[8] T.M.J. Fruchterman and E.M. Reingold. "Graph Drawing by Force-Directed Placement". Software - Practice & Experience. Vol. 21, pp. 1129-1164. 1991. DOI: 10.1002/spe.4380211102.

[9] E. Gamma, R. Helm, R. Johnson and J. Vlissides. "Design Patterns: Elements of Reusable Object-Oriented Software". Addison-Wesley. Reading, MA, USA.1995. ISBN: 0-201-63361-2.

[10] C. Geiger, H. Reckter, R. Dumitrescu, S. Kahl and J. Berssenbrügge. "A Zoomable User Interface for Presenting Hierarchical Diagrams on Large Screens". Proceedings of the 13th International Conference on Human-Computer Interaction. Part II: Novel Interaction Methods and Techniques. Springer-Verlag, pp. 791-800. San Diego, CA, USA. 2009.

[11] Gentleware. "Gentleware - model to business: gentleware homepage". 2010. Date of visit: August 17, 2010. URL: http://www.gentleware.com/

[12] C. Gutwin and C. Fedak. "Interacting with big interfaces on small screens: a comparison of fisheye, zoom, and panning techniques". Proceedings of Graphics Interface 2004, pp. 145-152. London, Ontario, Canada. Canadian Human-Computer Communications Society. 2004.

[13] O. Hamann. "Eagle Mode". 2011. Date of visit: January 12, 2011. URL: http://eaglemode.sf.net/

[14] K. Hornbaek, B.B. Bederson and C. Plaisant. "Navigation patterns and usability of zoomable user interfaces with and without an overview". ACM Transactions on Computer-Human Interaction (TOCHI). Vol. 9, pp. 362-389. 2002.

[15] J.A. Hurtado Alegría, M.C. Bastarrica and A. Bergel. "Avispa: a tool for analyzing software process models". Journal of Software: Evolution and Process. Vol. 26, pp. 434-450. 2014. ISSN: 2047-7481. DOI: 10.1002/smr.1578.

[16] T. Kamada and S. Kawai. "An algorithm for drawing general undirected graphs". Information Processing Letters. Vol. 31, pp. 7-15. 1989. DOI: 10.1016/0020-0190(89)90102-6.

[17] S. Kelly and J.-P. Tolvanen. "Domain-Specific Modeling: Enabling Full Code Generation". Wiley-IEEE Computer Society Pr. 2008.

[18] S. Kent. "Model Driven Engineering". Springer Berlin / Heidelberg, pp. 286-298. 2002.

[19] D. Koenig, A. Glover, P. King, G. Laforge and J. Skeet. "Groovy in Action". Manning Publications. 2007. ISBN: 1932394842.

[20] W.A. König, R. Rädle and H. Reiterer. "Interactive design of multimodal user interfaces". Journal on Multimodal User Interfaces. Vol. 3, pp. 197-213. 2010. ISSN: 1783-7677 (Print). 1783-8738 (Online). DOI: 10.1007/s12193-010-0044-2.

[21] J. de Lara, E. Guerra and J. Sánchez-Cuadrado. "Abstracting Modelling Languages: A Reutilization Approach". Advanced Information Systems Engineering. In: J. Ralyté, X. Franch, S. Brinkkemper and S. Wrycza, Eds. Springer Berlin Heidelberg, pp. 127-143. 2012.

[22] A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle and P. Volgyesi. "The Generic Modeling Environment". Budapest, Hungary. 2001.

[23] N. Liu, J. Hosking and J. Grundy. "Integrating a Zoomable User Interfaces Concept into a Visual Language Meta-Tool Environment". Proceedings of the 2004 IEEE Symposium on Visual Languages-Human Centric Computing. IEEE Computer Society, pp. 38-40. 2004.

[24] OMG. "UMLDI". 2006. Date of visit: August 9, 2010. URL: http://www.omg.org/spec/UMLDI/

[25] OMG. "UML 2.2". 2009. Date of visit: February 3, 2012. URL: http://www.omg.org/spec/UML/2.2/

[26] OMG. "Model Driven Architecture (MDA)". 2009. Date of visit: March 1, 2010. URL: http://www.omg.org/mda/

[27] OMG. "XMI 2.4.1". 2011. Date of visit: February 3, 2012. URL: http://www.omg.org/spec/XMI/2.4.1/

[28] Oracle. "Java Platform, Standard Edition 7 API Specification". 2011. Date of visit: February 2, 2012. URL: http://docs.oracle.com7/javase/7/ docs/api/

[29] E. Pietriga. "A toolkit for addressing HCI issues in visual language environments". In Proc. IEEE symposium on visual languages and computing (VL/HCC'05), pp. 145-152. 2005.

[30] E. Pietriga, S. Huot, M. Nancel and R. Primet. "Rapid development of user interfaces on cluster-driven wall displays with jBricks". Proceedings of the 3rd ACM SIGCHI symposium on engineering interactive computing systems. New York, NY, USA. ACM, pp. 185-190. 2011. DOI: 10.1145/1996461.1996518.

[31] M.G. Reinhard Stoiber. "Supporting Stepwise, Incremental Product Derivation in Product Line Requirements Engineering". Fourth International Workshop on Variability Modelling of Software-Intensive Systems, pp. 77-84. 2010.

[32] T. Reinhard, S. Meier, R. Stoiber, C. Cramer and M. Glinz. "Tool Support for the Navigation in Graphical Models". Proceedings of the 30th International Conference on Software Engineering. New York, NY, USA. ACM, pp. 823-826. 2008. DOI: 10.1145/1368088.1368211.

[33] I. Sommerville. "Software Engineering". Addison-Wesley Longman Publishing Co., Inc. 2006. ISBN: 0321313798.

[34] M. Stengel, M. Frisch, S. Apel, J. Feigenspan, C. Kastner and R. Dachselt. "View infinity: a zoomable interface for feature-oriented software development". 33rd International Conference on Software Engineering (ICSE), pp. 1031-1033. 2011. DOI: 10.1145/1985793.1985987.

[35] S.D. Suh and I. Neamtiu. "Studying Software Evolution for Taming Software Complexity". Software Engineering Conference, Australian. Los Alamitos, CA, USA. IEEE Computer Society, pp. 3-12. 2010. DOI: http://doi.ieeecomputersociety. org/10.1109/ASWEC.2010.26.

[36] J.-P. Tolvanen. "MetaEdit+: integrated modeling and metamodeling environment for domain-specific languages". Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications. Portland, Oregon, USA. ACM, pp. 690-691. 2006. DOI: 10.1145/1176617.1176676.

[37] Visual Paradigm. "Visual Paradigm UML". 2013. Date of visit: January 12, 2010. URL: http://www.visual-paradigm.com/product/vpuml/

[38] E. Yourdon. "Just enough structured analysis", pp. 73-75. 2006. Date of visit: February 2, 2011. URL: http://www.yourdon.com/jesa/pdf/JESA_ssbw.pdf

[39] N. Zhu, J. Grundy, J. Hosking, N. Liu, S. Cao and A. Mehra. "Pounamu: A meta-tool for exploratory domain-specific visual language tool development". Journal of Systems and Software. Vol. 80, pp. 1390-1407. 2007. ISSN: 0164-1212. DOI: 10.1016/j.jss.2006.10.028.

[40] "IsaViz". 2007. Date of visit: January 18, 2012. URL: http://www.w3.org/2001/11/IsaViz/

[41] "Shrimp - The CHISEL Group. University of Victoria". 2008. Date of visit: July 21, 2010. URL: http://www.thechiselgroup.org/shrimp

[42] "IBM developerWorks: Rational Rose". 2010. Date of visit: January 12, 2010. URL: http://www.ibm.com/developerworks/rational/products/rose/

[43] "Borland Together". 2010. Date of visit: January 12, 2010. URL: http://www.borland.com/us/ products/together/

[44] "Visio Home Page - Microsoft Office Online". 2010. Date of visit: January 12, 2010. URL: http://office.microsoft.com/en-us/visio/default.aspx

[45] "Piccolo2D - A Structured 2D Graphics Framework". 2010. Date of visit: August 6, 2010. URL: http://www.piccolo2d.org/

[46] "Graphical Modeling Framework". 2010. Date of visit: August 17, 2010. URL: http://www.eclipse.org/modeling/gmp/

[47] "ZGRViewer". 2011. Date of visit: January 18, 2012. URL: http://zvtm.sourceforge.net/zgrviewer.html

[48] "Moose technology". Date of visit: June 24, 2014. URL: http://www.moosetechnology.org/