Arquitecturas dirigidas por modelos aplicadas a aplicaciones sensibles al contexto

Ricardo Tesoriero, José A. Gallud, María D. Lozano, Víctor M. R. Penichet

Universidad de Castilla-La Mancha. Departamento de Sistemas Informáticos
Av. España S/N, Campus Universitario de Albacete, (02071), Albacete, España
[ricardo.tesoriero,jgallud,maria.lozano,victor.penichet]@uclm.es

Abstract. La sensibilidad al contexto es un aspecto muy importante de las aplicaciones actuales, ya que permiten: reconocer los factores humanos y ambientales que las rodean, y actuar en consecuencia. Este trabajo propone una arquitectura dirigida por modelos para desarrollar aplicaciones que explotan la sensibilidad al contexto. A través de tres capas de modelos se pretende pasar de la especificación de las características de la aplicación, a un esqueleto de código fuente que permita implementar un conjunto de artefactos software para soportar la implementación. La primera capa está definida por tres metamodelos que describen los aspectos funcionales (tareas), sociales y espaciales; la segunda capa está definida por otros tres metamodelos que describen los flujos de información entre entidades, los contextos de las entidades que son parte del sistema, y los aspectos relacionados con el espacio referencial en tiempo de ejecución de las mismas; la última capa consta de un único metamodelo que define las plataformas del entorno de despliegue del sistema. Para todos los metamodelos se ha diseñado un lenguaje específico de dominio y su correspondiente herramienta CASE para definirlos.

1 Introducción

En 1997, Martin Weisser describe la evolución de las tendencias de la computación donde vaticina cómo la computación ubicua se posiciona como la computación del futuro, donde muchas personas interactúan con varios ordenadores al mismo tiempo [14].

La tecnología Calm [14] explora cómo los ordenadores pueden ayudar a los usuarios de forma que sean capaces de ayudarles sin requerir su atención sin distraerles de la tarea que estén realizando.

En la actualidad, la computación ubicua es una realidad gracias a la reducción del coste de los dispositivos y la comunicación entre los mismos. Una de las herramientas que tenemos para poder implementar la tecnología Calm es la sensibilidad al contexto.

La sensibilidad al contexto se refiere a la idea de que los ordenadores pueden percibir y reaccionar basados en la información de su ambiente. Por lo tanto, el contexto se define como cualquier información que permite caracterizar la situación de una entidad [4].

Una aplicación sensible al contexto está condicionada tanto por las fuentes de información que definen su contexto, como por los elementos del ambiente que puede manipular. Por lo tanto, podemos definir dos relaciones importantes entre entidades: la relación inter-entidad y la relación intra-entidad, que definen cómo se manipula la información que percibe la entidad, y cómo esta información afecta al entorno.

Finalmente, dada la gran cantidad de plataformas en las que las aplicaciones móviles pueden ser desplegadas, este tipo de sistemas puede estar definido por un conjunto de aplicaciones que se ejecutan en entorno heterogéneo de plataformas tanto de dispositivos como de medios de comunicación.

Como consecuencia de lo expuesto anteriormente, dadas las características de este tipo de sistemas, debemos afrontar, al menos, la definición de elementos en tres niveles de abstracción: un primer nivel independiente de la forma de computación, en el que debemos ser capaces de describir las características del contexto que afectan a las aplicaciones; un segundo nivel independiente de la plataforma de despliegue en el que debemos ser capaces de describir cómo se comporta; y un tercero en el cual especificamos la plataforma en la que se ejecutan las aplicaciones del sistema.

Este artículo define una arquitectura dirigida por modelos de tres capas de modelos que permite desarrollar aplicaciones sensibles al contexto en la que en la primera capa de modelos se definen las características de la aplicación que se desea desarrollar, en la segunda capa se define cómo se definen y relacionan los artefactos de software, y en la tercera capa se define la plataforma en la que se ejecutan los artefactos de software.

El resto del artículo se organiza de la siguiente forma: en la sección 2 se analizan trabajos relacionados con el tema, en la sección 3 se presenta la propuesta de arquitectura dirigida por modelos para el desarrollo de aplicaciones sensibles al contexto, y la sección 4 describe las conclusiones y trabajos futuros planteados.

2 Trabajo relacionado

El objetivo de esta sección es la presentación de los trabajos relacionados con el desarrollo de aplicaciones sensibles al contexto más relevantes en el área. Éste ha sido categorizado en 3 grupos de acuerdo al tipo de propuesta que se aplica para resolver el problema: (a) desarrollo dirigido por ontologías, (b) arquitecturas dirigidas por modelos utilizando la Meta Object Facility (MOF) [8], y (c) las arquitecturas dirigidas por modelos utilizando perfiles de UML.

En cuanto a propuestas de desarrollo dirigidas por ontologías, el trabajo realizado en [6] presenta un caso de estudio que utiliza un mecanismo de transformación de modelos para generar las aplicaciones sensibles al contexto. El desarrollo se basa en seguir una secuencia definida de pasos que introducen conceptos de alto nivel desde el punto de vista del software del sistema. Sin embargo, desde el punto de vista conceptual, características como la dependencia de tareas y roles no son tenidos en

cuenta. Esta relación es un elemento conceptual de alto nivel muy importante a la hora de describir las relaciones sociales entre las entidades del sistema.

En [3] se presenta una ontología básica expresada en Web Onthology Language (OWL) cuya finalidad es el desarrollo de aplicaciones sensibles al contexto utilizando una arquitectura basada en agentes llamada CoBra. Desde el punto de vista conceptual, la ontología es capaz de modelar conceptos básicos como personas, agentes, lugares, eventos, etc. Sin embargo, no expresa la relación que existe entre las tareas, ni la dinámica de roles. Tanto las tareas, como los roles, son elementos de alto nivel muy importantes en la concepción de sistemas altamente interactivos como los sensibles al contexto. Otra limitación de esta propuesta es la utilización de una arquitectura cliente-servidor fija para el desarrollo de aplicaciones.

Respecto a las arquitecturas dirigidas por modelos utilizando MOF, el metamodelo para el desarrollo de aplicaciones sensibles al contexto propuesto en [5] define dos vistas del sistema: Core y Services. Una de las características más relevantes de este modelo es la independencia del contexto respecto del dominio de la aplicación. Sin embargo, no proveen elementos de alto nivel de abstracción tales como, la tarea, la dependencia entre tareas, los roles, los espacios, etc. Como consecuencia, se limita la expresividad del lenguaje en cuanto a las relaciones sociales y espaciales entre las entidades del sistema que forman parte del contexto. Desde el punto de vista de las características de software, el sistema no toma en cuenta aspectos arquitecturales, o de despliegue, ya que se basa en una arquitectura orientada a servicios en la Web.

Una trayectoria de diseños dirigidos por modelos de 3 niveles de abstracción independientes de la plataforma para aplicaciones sensibles al contexto se define en [1]. El trabajo de desarrollo está dividido en dos fases: la fase de preparación y la fase de servicios. A su vez, la fase de servicios está dividida en: la especificación de servicios, el diseño de servicios independiente de la plataforma, y el diseño de servicios específico de la plataforma. Aunque los servicios se especifican a diferentes niveles de abstracción, los conceptos que se manejan son de "bajo nivel", ya que nos encontramos con conceptos como eventos, colas y acciones, dejando de lado elementos de alto nivel de abstracción como la dependencia de tareas, roles, o las características de ubicación que permiten contextualizar la ejecución de la aplicación a un nivel conceptual.

En relación a las arquitecturas dirigidas por modelos utilizando perfiles de UML, en [10] se describe una propuesta basada en la composición dirigida por modelos de servicios Web utilizando la programación orientada a aspectos (AOP). La publicación expone que la interacción entre un usuario final y un servicio puede ser adaptada a parámetros contextuales sin afectar los objetivos generales relacionados con la lógica del servicio. Para llevarlo a cabo, desacopla la lógica del núcleo del servicio, de la que maneja el contexto, adoptando una arquitectura dirigida por modelos en la etapa de diseño, y la AOP en la etapa de codificación. Como en la mayoría de las propuestas que hemos examinado, los conceptos que se manejan son a "nivel computacional", como por ejemplo el de servicio, dejando de lado conceptos de nivel independiente de la computación, tales como el de tarea, entidad, espacio, entre muchos otros. Como en los casos anteriores esto no permite expresar relaciones sociales y espaciales entre

entidades. Además, la solución está dirigida a la Web, no teniendo en cuenta características muy importantes como la plataforma de despliegue.

3 Propuesta de arquitectura dirigida por modelos para el desarrollo de aplicaciones sensibles al contexto

Para poder solventar las deficiencias expresadas en la sección 2, proponemos la definición de una arquitectura dirigida por modelos basada en tres puntos de vista que describen el sistema en tres niveles de abstracción.

El primer punto de vista está relacionado con el análisis del sistema. En él se define un modelo independiente de la computación (CIM) del sistema en el que se describen: las características de las entidades, los espacios, y las tareas que llevan a cabo las entidades.

El segundo punto de vista está relacionado con el diseño del sistema. En él se define un modelo independiente de la plataforma (PIM) que captura las características esenciales del diseño de software del sistema en artefactos de software que estará caracterizado por la descripción del CIM definido en el nivel anterior.

Finalmente, el tercer punto de vista está relacionado con el entorno de despliegue del sistema. En él se define un modelo específico de la plataforma (PSM) donde se alojarán los artefactos de software que se han definido en el PIM.

La arquitectura define dos transformaciones: una transformación multi-modelo entre los modelos del CIM y el PIM, y otra definida de modelo a texto, que combina el PIM y el PSM para generar el código fuente de la aplicación.

Todos los metamodelos para definir el CIM, el PIM y el PSM se definieron en ECORE (dialecto de EMOF [8]). Aunque por motivos de espacio no trataremos las transformaciones, las transformaciones modelo a modelo fueron definidas en Atlas Transformation Language (ATL) [2], y las transformaciones modelo a texto en MOFScript [7]. Ésta transformaicón genera código en Java que compila sobre un framework abstracto a modo de adaptador a un framework concreto. Un video de todo el proceso puede verse en [13].

3.1 El modelo independiente de la forma de computación (CIM)

El objetivo de esta capa es la representación del sistema en términos de sus características sensibles al contexto. Estas características fueron estudiadas y modeladas en [12] donde se agrupan en: factores humanos y el entorno físico.

Para modelar estas características a nivel conceptual, hemos definido 3 lenguajes específicos de dominio (DSLs) a través de 3 metamodelos: el social, el espacial y el de tareas.

El metamodel social. El metamodelo social (ver **Fig. 1**) está basado en el Organization Structural Diagram [9] y permite clasificar e identificar las entidades que son parte del sistema en función de sus características. La clasificación se realiza a través de *Roles*. Los *Roles* pueden ser *Specializations* de otros *Roles* indicando que

el *Role* especializado tiene las características del que especializa, más algunas propias. Las *Instances* identifican una entidad particular con ciertas características a través de *Realizations* de *Roles*. La **Fig. 2** muestra un ejemplo en el que se define el *Role* "Sensor de luz" como una *Specialization* del *Role* "Sensor". Además, podemos observar ejemplos de las *Realizations* del *Role* "Sensor de luz" en las *Instances* "Sensor de luz del hall de entrada" y "Sensor de luz garage".

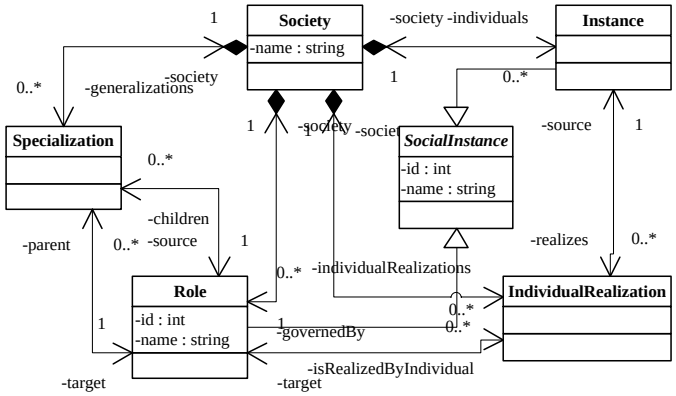

Fig. 1. Metamodelo social del CIM

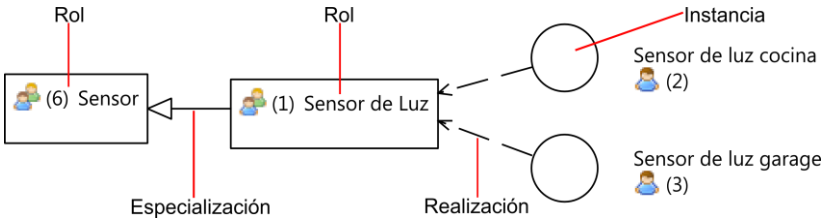

Fig. 2. Lenguaje específico de dominio del modelo social

El modelo de espacios. El metamodelo espacial (ver **Fig. 3**) se utiliza para representar los distintos tipos de espacios que forman parte del contexto de la aplicación. El *Universe* contiene dos tipos de *Spaces*: los *PhysicalSpaces* (que representan una posición, un volumen, una superficie en una pantalla, etc.), y los *VirtualSpaces* (que representan agrupaciones de espacios de acuerdo a su funcionalidad, como por ej.: las habitaciones, los halls, las cocinas, los baños, etc.). Además, el *Universe* define tres tipos de relaciones entre espacios: las *Contentions* que definen una relación entre dos *PhysicalSpaces* en la uno está contenido en otro, y que, dado que son físicos, un espacio puede estar contenido en un solo espacio (por ej. "Habitación 1" está contenida en "Casa"); las *Grouping* que permiten agrupar tanto *PhysicalSpaces* como *VirtualSpaces*, en un *VirtualSpace* de forma tal que se puede hacer referencia a un espacio que está compuesto de otros como si fuera un *Space* único (por ej. dentro de una "Casa", podemos decir que, tanto el "Baño" como la "Concina" son "Espacios comunes", y que si alguien está en el "Baño", o en la

"Cocina", está en un "Espacio común"); finalmente tenemos *Generalizations* que agrupan espacios de acuerdo a una funcionalidad común (por ej. tanto la "Habitación 1", como la "Habitación 2" son "Habitaciones"). Los ejemplos se basan en el modelo espacial de la **Fig. 4**.

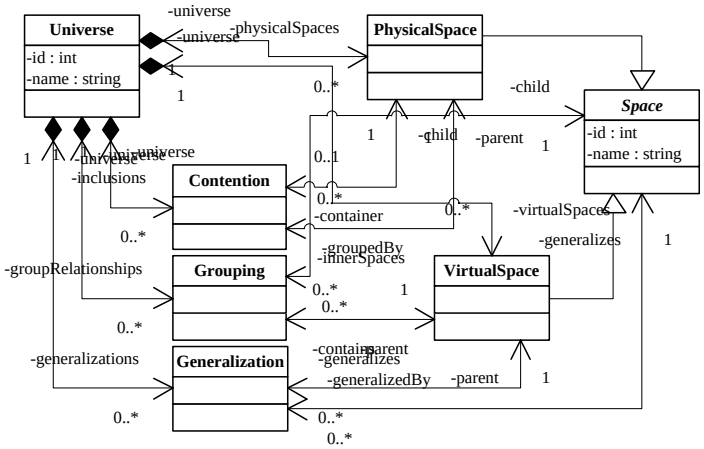

Fig. 3. Metamodelo espacial del CIM

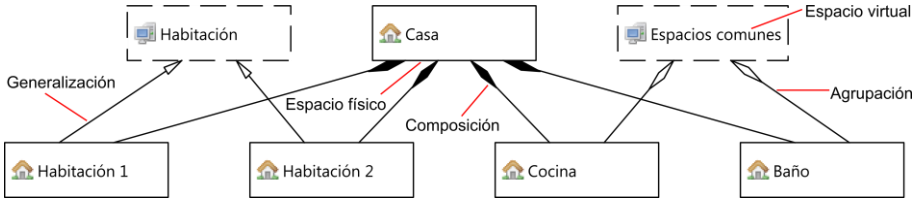

Fig. 4. Lenguaje específico de dominio del modelo de espacios

El modelo de tareas. El metamodelo de tareas (ver **Fig. 5**) está basado en el modelado de workflows [15] donde el token representa una entidad. Éste, además de expresar las relaciones temporales entre las tareas, se encarga relacionarlas con los modelos sociales y espaciales. Así se define un *ContextAwareSystem* como un conjunto de tareas (*Tasks*) y condiciones (*Conditions*) relacionadas. Existen dos tipos de *Tasks*: las *SystemTasks* que representan tareas que son ejecutadas dentro del sistema, y las *RouteTasks* que representan relaciones entre *SystemTasks* (alternativas entre tareas (*Join*) o paralelismo de tareas (*Split*)). Toda *Task* is preceded by a *Condition*. Así, toda *SystemTask* es precedida por una *PreCondition* o *InitialCondition*, y sucedida por una *PostCondition* o *FinalCondition*. Cada *Condition* representa un estado relevante al sistema a la *SystemTask* que será/fue ejecutada. El estado se define en función de un conjunto de expresiones que tienen en cuenta los distintos aspectos del sistema: el social (*socialExpression*) donde se introducen expresiones regulares para indicar sincronización (,) o alternativa (+), el espacial

(*spaceExpression*) donde también se usan expresiones regulares para denotar bifurcaciones (+) o secuencias (,) de eventos, el temporal (*taskExpression*) donde utilizamos expresiones regulares para expresar secuancias de tareas (,) o alternativas de ejecución (+), el lógico (*logicExpression*) donde se usan expresiones lógicas para establecer las condiciones, y el de intercambio de datos (*dataExpression*) donde utilizamos expresiones declarativas. De esta forma, quedan definidos los diferentes estados contextuales que son relevantes para la ejecución de las tareas. El lenguaje específico de dominio se mostrará en la sección 3.2.

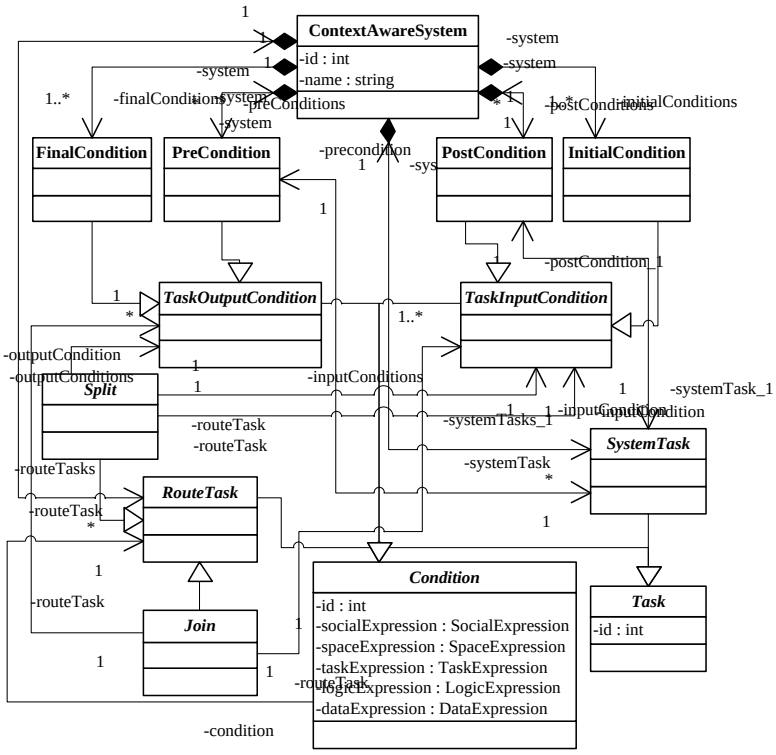

Fig. 5. Metamodelos de tareas CIM

3.2 Patrones de construcción de modelos independientes de la computación

En esta sección describiremos un conjunto de patrones de construcción de CIM para las situaciones de modelado más comunes. . Expresaremos que una entidad representada por una instancia IX es capaz de ejecutar una tarea TN mediante el operador =>.

Patrón Alternativa de Tareas. El patrón Alternativa de Tareas describe las alternativas de ejecución de tareas en función del rol de que está jugando la entidad.

Por ej., tomando de referencia el modelo social de la **Fig. 6.a**, en la **Fig. 6.b** observamos que IA => T1; IB => T1, T2; IC => T1.

Patrón Tareas Comunes. El patrón Tareas Comunes describe la posibilidad de ejecutar tareas que son comunes a varios roles. Por ej., tomando de referencia el modelo social de la **Fig. 6.a**, en la **Fig. 6.c** observamos que IA, IB, IC => T1.

Patrón Sincronización de Tareas. El patrón Sincronización de Tareas permite la ejecución de una tarea por parte de dos entidades de manera sincrónica. Por ej., tomando de referencia el modelo social de la **Fig. 6.a**, en la **Fig. 6.e** observamos que (IB e IC) => T. Este patrón es muy utilizado para intercambiar datos. Por ej., si la *dataExpression* de la *PreCondition* de T es "IB (data)", y la *dataExpression de la PostCondition* (no dibujada por razones de espacio) es "IC (data)", entonces "data" ha pasado de IB a IC.

Patrón Tareas Paralelas/Concurrentes. El patrón Tareas Paralelas/Concurrentes describe la posibilidad de ejecución de tareas en forma paralela asíncrona de tareas. Por ej., tomando de referencia el modelo social de la **Fig. 6.a**, en la **Fig. 6**.**f**, IB=>T1 e IV=>T2. Para expresar concurrencia, por ej., sobre IB, las *socialExpressions* de tanto la *PostCondition*, como de las *PreConditions*, deberían hacer referencia a IB. De esa forma IB=>T1 y T2 concurrentemente.

Patrón de Sensibilidad a la Ubicación. El patrón de Sensibilidad a la ubicación expresa de manera conceptual cómo la ubicación es tenida en cuenta a la hora de contextualizar la ejecución de una tarea. Por ej., tomando de referencia el modelo social de la **Fig. 6.a** y el modelo de espacios de la **Fig. 4**, en la **Fig. 6.d** IC => T1 si está en "Habitación 1", IC => T2 (si está en el "Espacio Común", o sea en la "Cocina" o el "Hall"), IC => T3 (si está en la "Cocina" o en la "Habitación 2").

3.3 El modelo independiente de la plataforma (PIM)

El modelo independiente de la plataforma está inspirado [4, 11]. El modelado muestra el sistema desde dos puntos de vista: el inter-entidad y el intra-entidad. Mientras la primera vista se centra en el modelado de los flujos de información entre entidades mediante el metamodelo de flujos de información, y la referencia existencial entre entidades mediante el metamodelo de espacios referenciales; la segunda vista se centra en cómo actúa sobre el ambiente en función de la percepción del mismo mediante el metamodelo de contexto de entidad.

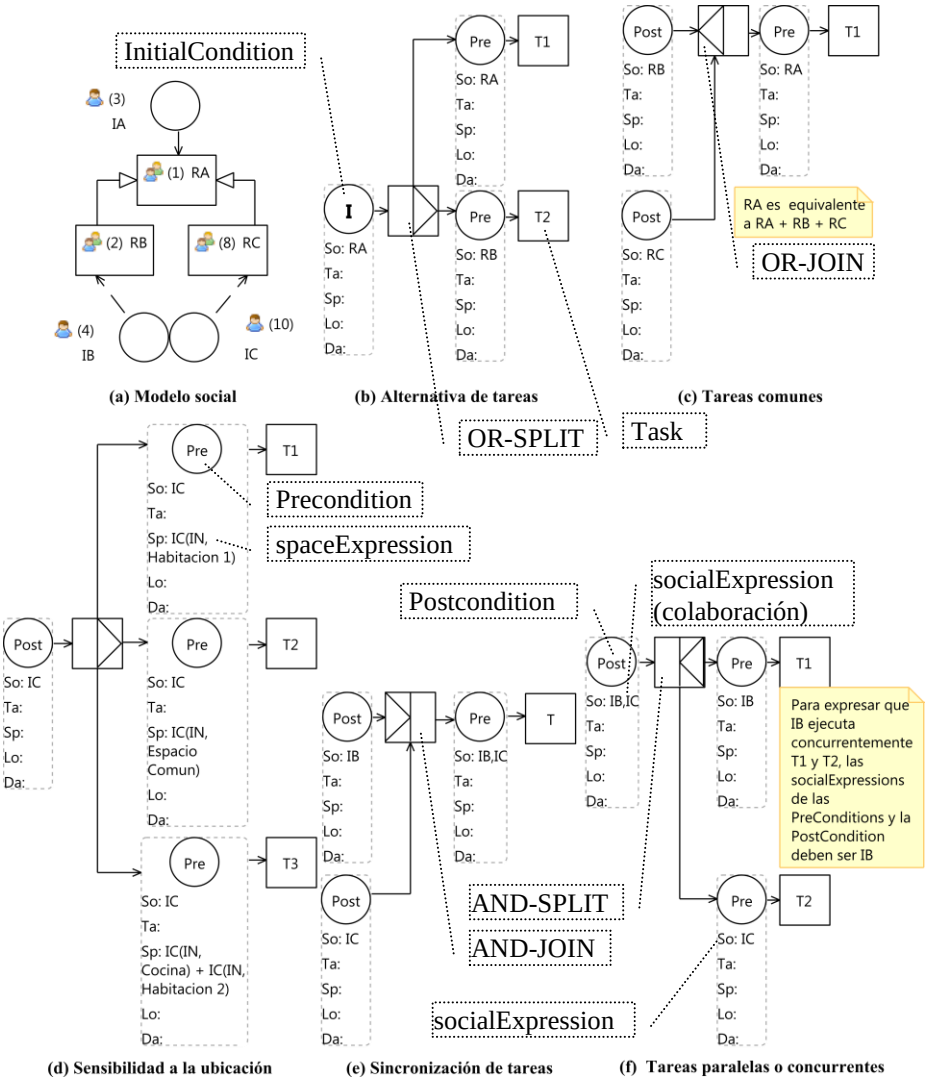

Fig. 6. Patrones de construcción de modelos CIM

Modelos de flujos de información. El metamodelo de flujos de información (ver **Fig. 7**) fue inspirado por [16] donde todo flujo de información es generado por una entidad, y consumido por otra. El entorno es considerado tanto productor (fuente de percepción), como consumidor (actuaciones de las entidades sobre él) de flujos de información. El *SystemInformationFlow* define los *InformationFlows* (tanto *PointToPont* como de *Broadcasting*) entre las *ContextAwareEntities*. Los *PointToPointInformationFlows* definen *Data Input* (información de retorno) o Data *Output* (enviar información), pero los *BroadcastInformationFlows* solo pueden definir *Data Output*. La *Data* se define en términos de colecciones de

VaraibleDataDefinitions. En la sección 3.4 mostramos la notación utilizada en el DSL.

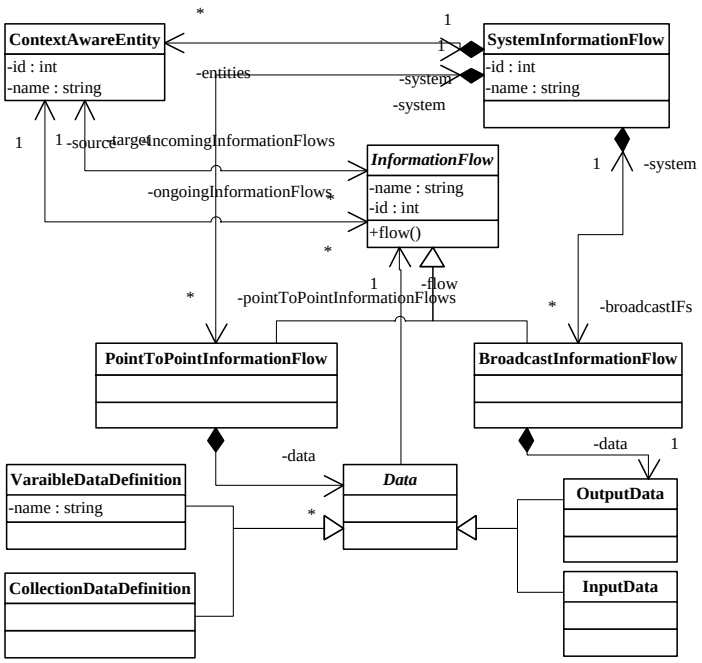

Fig. 7. Metamodelo de flujos de información

Modelo de espacios referenciales. El objetivo de este metamodelo (ver **Fig. 8**) es mostrar cuál es la dependencia existencial entre las diferentes entidades que son parte del sistema, y la cardinalidad de cada una de ellas. Un ejemplo de dependencia existencial existe entre el carro de compras de una página Web, y la página en sí. No tiene sentido hablar del carro de compras si no existe la aplicación Web a la que está asociada. Así, un *ReferentialSpace* agrupa un conjunto de *ContextAwareEntityReferences* (que representan referencias a las entidades definidas en el modelo de flujos de información) y otros *ReferentialSpaces*. Las relaciones entre los *ReferentialSpaces* y las *ContextAwareEntityReferences*. Así, si un *ReferentialSpace* desaparece, entonces desaparecen todos los *ReferentialSpaces* y las *ContextAwareEntityReferences* que dependen de él. Un ejemplo de una aplicación compartida para dibujar se ve en la **Fig. 9** donde del *SharedApplicationSpace* dependen las *WorkSessions*, de la que a su vez dependen las *UserSessions*. Por lo tanto, cuando desaparece una *WorkSession*, desaparecen las*UserSessions* asociadas.

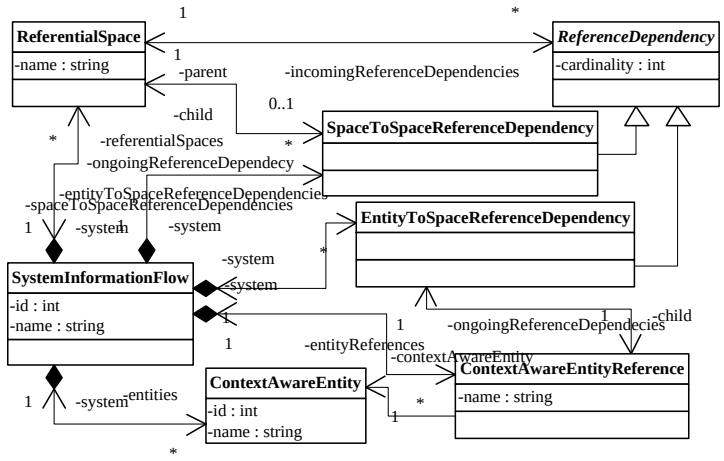

Fig. 8. Metamodelo de espacios referenciales

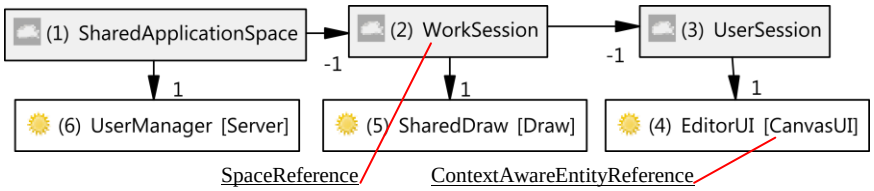

Fig. 9. Lenguaje específico de dominio del modelo de flujo de información

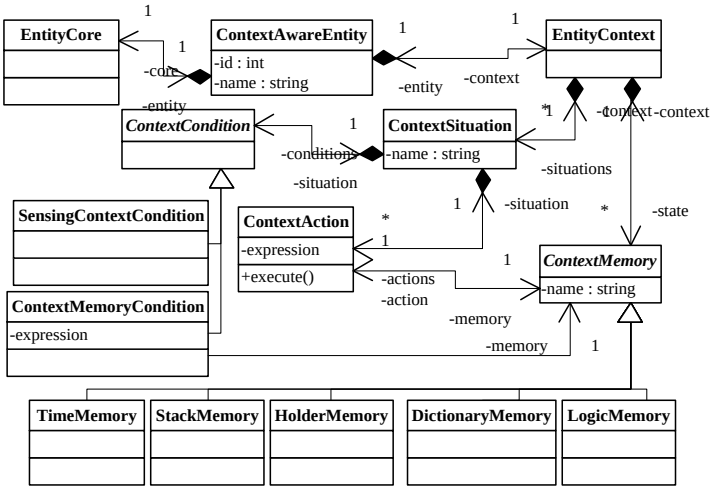

Fig. 10. Metamodelo de contextos de entidad

Modelo de contexto de entidad. El metamodelo de contexto de entidad (ver **Fig. 10**) tiene por objetivo especificar el comportamiento del contexto a partir de la percepción de su entorno a través de los flujos de información definidos en el modelo de flujos de información. Así, una *ContextAwareEntity* es capaz de definir un conjunto de *ContextMemories* (que mantienen el estado del contexto visto desde el punto de vista de la entidad) y un conjunto de *ContextSituations*. Cada situación define un conjunto de *ContextConditions* que describen un estado relevante del contexto. Cuando esas *ContextConditions* se satisfacen, entonces se dice que ha ocurrido una situación que es procesada desencadenando un conjunto de *ContextActions* que son la respuesta a ese estado relevante del contexto.

3.4 Patrones de construcción de modelos independientes de la plataforma

En esta sección se muestran patrones de modelos para situaciones frecuentes.

Patrón Evento. El objetivo de este patrón es la notificación de un evento por parte de una entidad a otras. Por ej., en la **Fig. 11.a** y **b** se definen un *BroadcastInformation Flow* entre un *CardReader* y un conjunto de *Applications*, mediante el cual se propaga el *id* leído.

Patrón Polling. El objetivo de este patrón es la recolección, por parte de una entidad, de información de contexto contenida en otra entidad. Por ej., en la **Fig. 11.c** y **d** se define un *PointToPointInformationFlow* entre un *CardReader* y una *Application*. El flujo parte de la *Application*, se procesa en el *CardReader*, y devuelve a la *Application* el *id*.

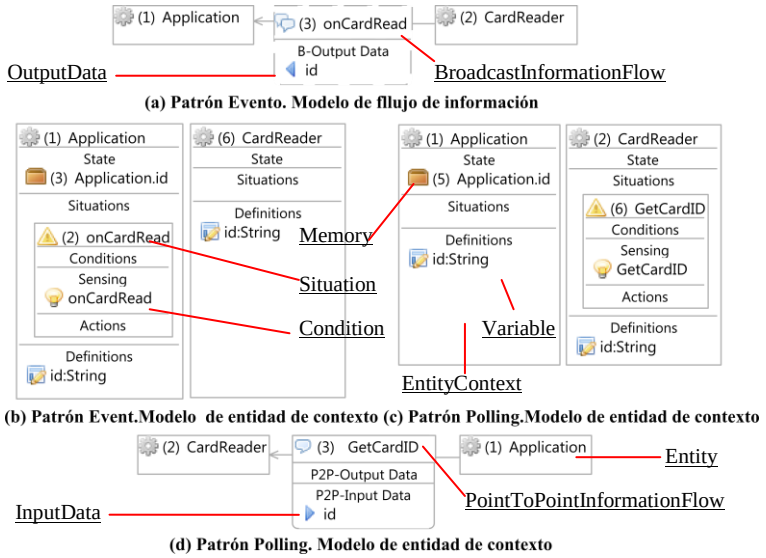

Fig. 11. Patrones de construcción de modelos independientes de la plataforma

3.5 Modelo específico de la plataforma

El metamodelo describe un *DeploymentEnvironment*, donde se definen las *Platforms* de dos elementos principales: *Devices* y *Connections*. Mientras los *Devices* (nodos de procesamiento) agrupan *EntitiesEntities,* las *Connectios* (canales de comunicación) agrupan *InformationFlows*. Para cada *Device* definimos el *ProgrammingLanguage*, y el *OperatingSystem*; para cada *Conenction* definimos el *Medium* y el *Protocol.* Un ejemplo se puede ver en la **Fig. 12**. A partir de la combinación este modelo con el PIM se genera código fuente en Java compatible con un framework abstracto que se adaptará a uno concreto que soporte la funcionalidad requerida.

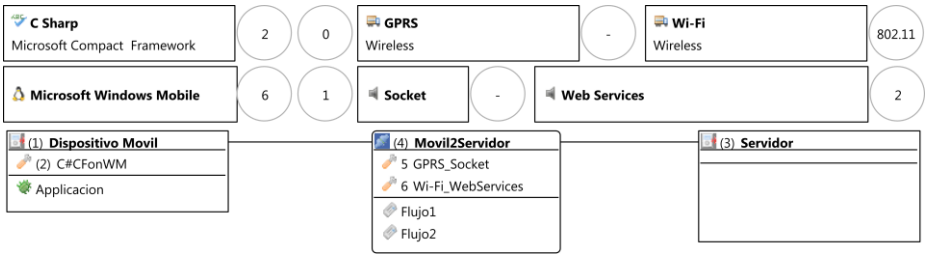

Fig. 12. Lenguaje específico de dominio de modelo específico de dominio

4 Conclusiones

El trabajo presenta una arquitectura dirigida por modelos dividida en tres capas que permite definir de forma independiente aplicaciones sensibles al contexto en tres niveles de abstracción.

La primera capa permite describir cómo el entorno afecta la funcionalidad de la aplicación de forma independiente de los artefactos de software que se utilicen para definirla. El modelo está definido por tres metamodelos que describen las tareas, el entorno social y espacial del sistema.

La segunda capa permite definir los artefactos de software que definirán el sistema en la capa anterior sin especificar la plataforma. Este modelo describe al sistema desde dos puntos de vista: el de las relaciones inter-entidad e intra-entidad.

Finalmente, la última capa permite especificar la/s plataforma/s de despliegue de los artefactos de software definidos en la capa anterior.

Así, el artículo cubre el modelado de las aplicaciones sensibles al contexto desde su concepción hasta su despliegue. Además, el artículo presenta un conjunto de patrones de modelado útiles para la definición de situaciones recurrentes que existen en este tipo de aplicaciones.

Como trabajo futuro, estamos trabajando en la implementación de un framework concreto que soporte el código generado a partir del PSM. También estamos trabajando en la generación de código de verificación (casos de prueba) para las clases generadas a partir de las post condiciones definidas en el modelo de tareas.

5 Referencias

1. Almeida, J. P. A., Iacob, M.-E., Jonkers, H. and Quartel, D.: “Model-Driven Development of Context-Aware Services”; Proc. Distributed Applications and Interoperable Systems, Lect. Notes Comp. Sci. 2025, Springer, 2006, 213-227.
2. Bzivin, J., Jouault, F. and Valduriez, P.: “The Atlas Transformation Language ATL”, 2008, URL: http://ralyx.inria.fr/2003/Raweb/atlas/uid10.html.
3. Chen, H., Finin, T. W. and Joshi, A.: “Using OWL in a Pervasive Computing Broker”; Proc. the Workshop on Ontologies in Agent Systems at the 2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems, 73, Melbourne, Australia, July 2003, 9-16.
4. Dey, A., Abowd, D., G. and Salber, D.: “A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications”; Human-Computer Interaction, 16 (2/4), 2001, 97-166.
5. Farias de, C. R. G., Leite, M. M., et al.: “A MOF metamodel for the development of context-aware mobile applications”; Proc. ACM Symposium on Applied Computing, ACM, New York USA, 2007, 947-952.
6. Georgalas, N., Ou, S., Azmoodeh, M. and Yang, K.: “Towards a Model-Driven Approach for Ontology-Based Context-Aware Application Development: A Case Study”; Proc. 4th International Workshop on Model-Based Methodologies for Pervasive and Embedded Software, IEEE, Washington DC, USA, 2007, 21-32.
7. MOFScript: The Eclipse Foundation. URL: http://www.eclipse.org/gmt/mofscript/.
8. OMG. The Object Management Group: “Meta Object Facility (MOF) 2.0 Core Final Adopted Specification”; (Mar. 2004).
9. Penichet, V. M. R.: “TOUCHE: Task-Oriented and User-Centred Process Model for Developing Interfaces for Human-Computer-Human Environments”. PhD thesis, University of Castilla-La Mancha, Spain, 2007.
10. Prezerakos, G. N., Tselikas, N. D. and Cortese, G.: “Model driven Composition of Context-aware Web Services Using ContextUML and Aspects”; Proc. International Conference on Web Services, IEEE Computer Society, Utah, USA, July 2007, 320-329.
11. Schilit, B., Adams, N. and Want, R.: “Context-Aware Computing Applications”; Proc. Workshop on Mobile Computing Systems and Applications, IEEE, Santa Cruz, US, 1994.
12. Schmidt, A.: “Ubiquitous Computing - Computing in Context”. PhD thesis. Lancaster University, UK, 2002.
13. Tesoriero, R. CAUCE MDA. URL: http://www.youtube.com/watch?v=gp-oLnM3GWY&playnext=1&list=PL06B4EF11544CF008. 2011.
14. Weiser, M. and Brown, J. S.: “The Coming Age of Calm Technology”; Beyond Calculation: The Next Fifty Years of Computing, Denning, P. J. and Metcalfe, R. M. (eds.), Copernicus, 1997, 75-85.
15. WfMC: The Workflow Management Coalition: “Workflow Management Coalition Terminology and Glossary (WFMC-TC-1011)”; Brussels, 1996.
16. Zimmer, T.: “Towards a Better Understanding of Context Attributes”; Proc. PerCom Workshops, IEEE Computer Society, 2004, 23-27.