CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL INSTITUTO POLITÉCNICO NACIONAL

UNIDAD ZACATENCO

DEPARTAMENTO DE COMPUTACIÓN

Marco conceptual para la administración del contexto en sistemas colaborativos

Tesis que presenta

Ismael González Martínez

Para obtener el grado de

Maestro en Ciencias

en Computación

Directora de Tesis: Dra. Sonia Guadalupe Mendoza Chapa

RESUMEN

Debido a la naturaleza dinámica de los ambientes colaborativos, el software destinado a ayudar a grupos de trabajo debe adaptarse por sí mismo a las diferentes situaciones que podrían ocurrir. Este requisito de adaptabilidad puede satisfacerse mediante sistemas colaborativos (*groupware*) con un soporte adaptativo, capaz de hacer frente a los cambios en el contexto de uso, que ha sido considerado como un aspecto importante en el diseño de sistemas interactivos. Sin embargo, la construcción de tal soporte no es una tarea sencilla de lograr, debido a dos problemas principales que han sido identificados por la investigación actual en la computación consciente del contexto: 1) la mayoría de los estudios se han enfocado principalmente en el contexto de un único usuario y 2) el soporte adaptativo de los sistemas contextuales generalmente considera un número reducido de variables contextuales, principalmente la ubicación del usuario y la plataforma de interacción. Esta tesis se enfoca en el modelado y la administración del contexto de uso desde el punto de vista de los ambientes colaborativos, en los cuales la adaptabilidad del sistema depende de múltiples variables. Se propone un marco conceptual compuesto de un modelo semántico y una arquitectura para el desarrollo de sistemas colaborativos conscientes de contexto. El modelo semántico representa a los componentes de las aplicaciones colaborativas creadas bajo el paradigma MVC, a fin de que los miembros de un grupo de trabajo puedan interactuar en un ambiente colaborativo para compartir servicios y recursos. Para el desarrollo de sus clases y propiedades, se utilizaron los escenarios del marco de desarrollo XARE para sistemas colaborativos conscientes de contexto. Éstas complementan las desarrolladas en la arquitectura RAMS, que ofrece un soporte computacional para la compartición de los recursos de una organización. Así mismo, se describen las etapas del ciclo de adaptación, que fundamentan la arquitectura propuesta y se detalla su funcionamiento y su implementación. Tanto el modelo semántico como la arquitectura conceptual han sido validados mediante la implementación de la herramienta de administración de contenidos vía *NFC*, que permite visualizar los datos y los documentos asociados a una reunión, considerando el contexto del grupo de usuarios que utilizan la aplicación.

ABSTRACT

Owing to the dynamic nature of collaborative environments, the software intended to assist working groups should adapt itself to the different situations that might occur. This adaptability requirement can be satisfied by providing groupware systems with an adaptability support able to cope with changes in the context of use, which has been considered as an important aspect in the design of interactive systems. Nevertheless, the building of such a support is not an easy task to accomplish, as two main problems have been identified by current research in context-aware computing: 1) most of the studies have mainly focused on a single user's context, thus the context of multiple users involved in a common project remains an unexplored subject, and 2) the adaptability support of context-aware systems generally considers a reduced number of contextual variables, mainly the user's location an the interaction platform. This thesis is focused on modeling and administration of the context of use from the point of view of collaborative environments, in which system adaptability depends on several variables. This thesis is focused on the modeling and the administration of the context of use from the point of view of collaborative environments, in which system adaptability depends on multiple variables. A conceptual framework is proposed consisting of a semantic model and an architecture for the development of context-aware collaborative systems. The semantic model represents the components of the collaborative applications created under the MVC paradigm, so that members of a working group can interact in a collaborative environment to share services and resources. For the development of its classes and properties, the scenarios of the XARE framework for the development of context-aware collaborative systems were used. These complement those developed in the RAMS architecture, which offers a computational support for the sharing of the resources of an organization. Also, the stages of the adaptation cycle are described, which provide details of the proposed architecture and its operation and its implementation are detailed. Both, the semantic model as the conceptual architecture have been validated through the implementation of the content management tool via NFC, that allows to visualize the data and the documents associated with a meeting, considering the context of the group of the users of the application.

Agradecimientos

Conacyt y Cinvestav:

Gracias por su apoyo para la realización de esta tesis.

Dra. Sonia:

Gracias por su confianza, paciencia, apoyo y comprensión en el desarrollo de esta tesis.

A mis padres:

Gracias por creer en mí, por dedicar su vida a mí y enseñarme que el deber, la responsabilidad y el amor son la base para mi crecimiento. Ustedes me han enseñado que los actos son más valiosos que las palabras y por ello, esta tesis está dedicada a ustedes.

A mi sobrino:

Gracias por todas tus ocurrencias y compañía, por las cuales nunca me faltó una sonrisa.

A mis amigos:

Gracias Laiphel, Daniel y Mary. Su tiempo, apoyo e ideas durante este camino son tesoros invaluables que llevo conmigo.

Gracias Dr. José Matías Mentado Alvarado, Dr. Sergio Víctor Vergara Chapa y Sofía Reza Cruz por su apoyo en la realización de esta tesis.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

INTRODUCCIÓN

Este capítulo ofrece una visión general al lector del trabajo realizado. Así, la Sección 1.1 presenta la motivación que da origen a esta tesis, seguido se presenta el planteamiento del problema en la Sección 1.2. En tanto, la Sección 1.3 presenta los objetivos planteados y la Sección 1.4 muestra la hipótesis de investigación. Finalmente, en la Sección 1.5 se detalla la organización de esta tesis.

1.1. Motivación

Debido a la naturaleza dinámica de los ambientes colaborativos, el software destinado a asistir a los grupos de trabajo deberá adaptarse por sí mismo a las diferentes situaciones que pudiesen ocurrir. Esta necesidad de adaptación puede satisfacerse ofreciendo sistemas colaborativos con un soporte de adaptabilidad capaz de hacer frente a los cambios en el contexto de uso.

Los ambientes de trabajo colaborativo se caracterizan por las diferentes situaciones que pueden ocurrir en distintos momentos. Estas situaciones son el producto de la interacción entre varias personas, quienes poseen habilidades diversas y, en consecuencia, juegan diferentes roles en proyectos comunes. A su vez, estos proyectos están regidos por fechas de

entrega bien definidas y requieren recursos de disponibilidad variable que utilizan los colaboradores, de manera oportunista, durante la realización de cada tarea. El software que da soporte a la colaboración (*groupware*) debería ser lo suficientemente flexible para adaptarse a cada situación, con la intención de incrementar su usabilidad y capacidades para impactar positivamente el desempeño general del grupo.

Para ilustrar este requerimiento, considere que varias personas trabajan en el mismo proyecto. En este caso, el sistema colaborativo debería estar consciente de las necesidades de cada usuario para adaptarse convenientemente, e.g., destacando los documentos que le son más útiles a cada uno. Otra situación común en ambientes de trabajo colaborativo ocurre cuando la fecha de entrega de un proyecto está próxima; en este caso el sistema colaborativo debería adaptarse para dar prioridad al proyecto en cuestión, e.g., facilitando herramientas para agilizar su desarrollo y eliminando los elementos que causan distracciones. Un ejemplo más se observa cuando se está llevando a cabo una reunión importante; en esta situación el sistema colaborativo podría adaptarse para no admitir interrupciones, e.g., posponiendo la entrega de mensajes irrelevantes a un momento más apropiado [Olivares Toledo, 2011].

Este requerimiento de adaptación se relaciona con el concepto de "contexto de uso", el cual es un elemento importante en el diseño de sistemas interactivos. Sin embargo, dos cuestiones relativas a este concepto han sido identificadas en las investigaciones actuales sobre cómputo consciente de contexto: 1) la mayoría de los estudios se han centrado en el contexto de un sólo usuario, así que el contexto de múltiples usuarios involucrados en un mismo proyecto permanece prácticamente inexplorado y 2) la adaptabilidad de los sistemas contextuales considera un número reducido de variables (principalmente la ubicación del usuario y la plataforma).

1.2. Planteamiento del problema

El comportamiento humano se adapta a las diferentes situaciones de la vida diaria, e.g., las personas se apresuran cuando se les hace tarde para ir a una reunión, alzan la voz para

ser escuchadas en ambientes ruidosos o evitan distracciones cuando tienen mucho trabajo que realizar. Por el contrario, la mayoría de los sistemas computacionales no se ajustan al contexto del usuario, mientras que el resto lo hace en una forma elemental, e.g., algunas características de los teléfonos celulares pueden ser modificadas de acuerdo al perfil activo del usuario.

Muchos trabajos de investigación sobre sistemas interactivos están orientados a definir, modelar y redactar cambios en el contexto de uso. Sin embargo, la mayoría se enfoca en sistemas mono-usuario que sólo consideran un número limitado de variables contextuales, como la plataforma (más específico el tamaño de pantalla) y la ubicación del usuario.

1.3. Objetivos

1.3.1. Objetivo general

Desarrollar un marco conceptual compuesto de un modelo semántico del espacio colaborativo global y una arquitectura conceptual para la administración del contexto en sistemas colaborativos, con el fin de ofrecer a los desarrolladores de aplicaciones un soporte para integrar la adaptabilidad a las diferentes situaciones que se presenten en tiempo de ejecución, a través de la composición de los contextos individuales de los colaboradores.

1.3.2. Objetivos particulares

1. Integrar las ontologías de la arquitectura RAMS [García García, 2013] en el marco conceptual para la administración del contexto en sistemas colaborativos.

2. Identificar y modelar semánticamente los elementos clave del marco conceptual XARE [Sánchez Morales, 2013] e integrarlos en las ontologías de la arquitectura RAMS.

3. Modelar semánticamente el espacio de colaboración global para sistemas colaborativos,

con el fin de integrar aplicaciones bajo el paradigma Modelo-Vista-Controlador a las ontologías de la arquitectura RAMS.

4. Establecer las directrices para el desarrollo de las capas de la arquitectura conceptual propuesta: 1) percepción de eventos contextuales, 2) detección de la situación, 3) comunicación de la situación, y 4) adaptación del sistema.

5. Desarrollar la herramienta *Administración de contenidos vía NFC* que demuestre la viabilidad del modelo semántico y de la arquitectura conceptual propuestos, a través de una serie de pruebas para comprobar su desempeño ante distintas condiciones del dispositivo.

1.4. Hipótesis de investigación

Mediante un marco conceptual del contexto para sistemas colaborativos, es posible integrar un soporte de adaptabilidad a cambios contextuales.

1.5. Organización del documento

El Capítulo 2 introduce el contexto de investigación de este trabajo y se analizan los trabajos relacionados. Tales trabajos han ayudado a definir el alcance de esta tesis, analizando sus características, con el fin de detectar las áreas de oportunidad que la presente propuesta puede investigar y satisfacer. Se han analizado sistemas conscientes al contexto, con el fin de identificar sus adaptaciones, las entidades y sus respectivas variables contextuales que consideran para dicho propósito.

El Capítulo 3 introduce la arquitectura RAMS, que es un soporte computacional de compartición de recursos físicos (e.g., salas de juntas, salones de clases y auditorios), humanos (e.g., trabajadores y visitantes) y virtuales (e.g., scanners, plotters y dispositivos móviles) ubicados en una organización, siendo que éstos pueden estar distribuidos en diferentes oficinas, edificios o dispositivos de almacenamiento. Para ello, la arquitectura RAMS utiliza

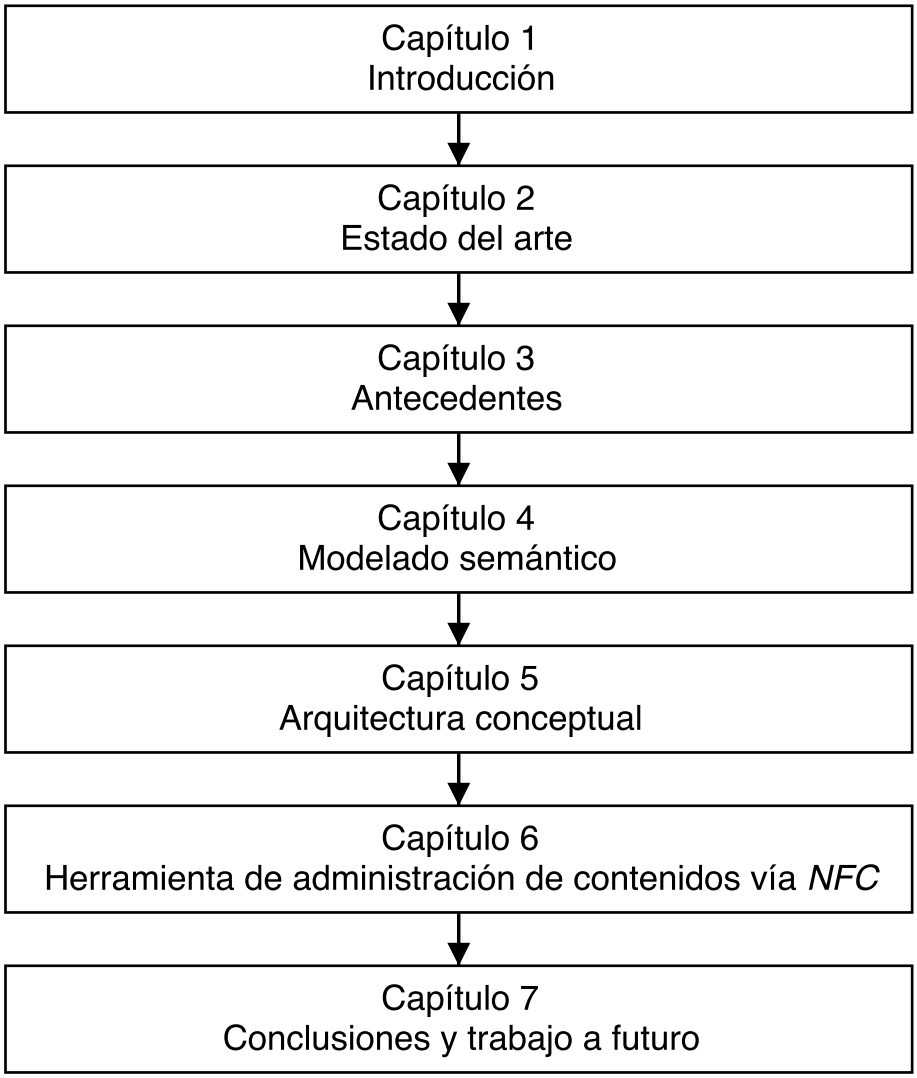


Figura 1.1: Organización del documento

un modelo semántico que es enriquecido con la representación de un espacio colaborativo global. También se describe el marco de desarrollo XARE (*conteXt-Aware groupwaRE*) para aplicaciones colaborativas conscientes de contexto, en cuyo desarrollo se identificaron los requerimientos para algunas aplicaciones en siete escenarios. Tales requerimientos se toman en cuenta para el modelado del espacio colaborativo global y para el desarrollo de la arquitectura conceptual.

En el Capítulo 4 se propone un modelo semántico para el espacio colaborativo global, desarrollado a partir de las clases identificadas en los escenarios descritos en el Capítulo 3 y que integra las clases pertenecientes a las ontologías de la arquitectura RAMS. Su uso se ejemplifica mediante la representación de las aplicaciones correspondientes a los escenarios estudiados en el desarrollo del marco XARE.

En el Capítulo 5 se analiza el ciclo de adaptación al contexto de un sistema para el desarrollo de la arquitectura conceptual propuesta. Se explican cada una de las capas (percepción de eventos contextuales, detección de la situación, comunicación de la situación y adaptación del sistema) y se dan las directrices para su implementación.

Posteriormente, en el Capítulo 6, detalla la implementación y pruebas de la herramienta *Administración de contenidos* para validar el marco conceptual propuesto. Finalmente, en el Capítulo 7, se presenta un resumen de las contribuciones de este trabajo, así como el camino previsto que este trabajo de investigación puede seguir, con el fin de ser enriquecido.

Capítulo 2

ESTADO DEL ARTE

Las Secciones 2.1 y 2.2 definen una serie de términos fundamentales sobre los sistemas colaborativos conscientes de contexto y sobre el modelado semántico. Particularmente, la Sección 2.1 define qué es un sistema consciente de contexto, mientras que la Sección 2.2 describe el contexto de uso, los sistemas mono-entidad y multi-entidad, el contexto de uso de los sistemas mono-usuario y multi-usuario. Posteriormente, la sección 2.3 introduce a las ontologías, sobre las cuales se fundamenta el modelo semántico del espacio colaborativo global propuesto. Finalmente, la Sección 2.4 presenta un breve resumen de varios sistemas conscientes de contexto representativos y se realiza un análisis comparativo de ellos.

2.1. Sistemas conscientes de contexto

En 1984, los científicos de la computación Irene Greif y Paul Cashman del Instituto de Tecnología de Massachusetts (MIT, por sus siglas en inglés) acuñaron el término "Trabajo Cooperativo Asistido por Computadora" (CSCW, por sus siglas en inglés). Esta área centra su investigación en el entendimiento de las características del trabajo cooperativo, a fin de diseñar la tecnología computacional que le proporcione soporte.

Dicho entendimiento requiere ser abordado desde las perspectivas de la Sociología, Psi-

cología Organizacional, Tecnología de la Información y la Comunicación, ya que su unión tiene un gran impacto en la forma en la cual las personas realizan sus actividades en el trabajo [Dourish, 1996]. Sin embargo, en la primera década de su existencia, el desarrollo de esta disciplina estuvo dividido. Mientras que la Tecnología de la Información y la Comunicación desarrollaron los principios de diseño y las estrategias de implementación para sistemas dedicados, la Psicología Organizacional y la Sociología estudiaron los grupos colaborativos sin tener en cuenta el soporte tecnológico.

Se entiende como **cooperación** a un proceso en una organización que consiste en la realización conjunta de una tarea, por parte de la gente, con el fin de generar un producto final. Al término del proceso, las contribuciones individuales no pueden ser aisladas porque el resultado final es la suma de todas ellas [Miao and Haake, 1999].

Actualmente existe un gran interés en el diseño de sistemas colaborativos conscientes del contexto, con el fin de solventar las implicaciones de la compartición de una amplia gama de medios de comunicación, herramientas y recursos de información entre un grupo de trabajo.

2.2. Contexto de uso

El **contexto** es cualquier información que puede usarse para caracterizar el estado de una entidad. Una **entidad** es una persona, lugar u objeto relevante entre un usuario y una aplicación, incluyendo al usuario y a la aplicación [Dey et al., 2001].

Tal información corresponde a las **variables contextuales**, definidas como las variables relevantes que los sistemas consideran para llevar a cabo la adaptación [Ghiani et al., 2009].

Así, la **adaptabilidad al contexto** es un conjunto de métodos y técnicas orientados a permitir a los usuarios realizar sus tareas con menor esfuerzo y mejores resultados considerando el contexto actual del usuario.

El **enfoque representacional** define el contexto como un conjunto enumerable y previamente conocido de distintos factores que representan la situación actual [Dourish, 2004].

Por lo cual, una **situación** es una configuración de las variables contextuales que es propicia para realizar una adaptación en el sistema colaborativo.

2.2.1. Sistemas mono-entidad y multi-entidad

Los sistemas mono-entidad consideran la situación de una sola entidad que participa a lo largo de la ejecución de un sistema; en cambio, los sistemas multi-entidad consideran la situación de varias entidades de forma simultánea [Olivares Toledo, 2011].

De forma similar, los sistemas mono-usuario consideran el contexto de un solo usuario que utiliza un sistema y los sistemas multi-usuario consideran el contexto de varios usuarios de forma simultánea.

2.2.2. Contexto de uso de los sistemas mono-usuario

El contexto de uso de los sistemas mono-usuario consiste en la terna de componentes (**usuario**, **plataforma**, **ambiente**) [Vanderdonckt et al., 2008] donde:

1. **Usuario**: corresponde al arquetipo humano que utilizará el sistema interactivo [Vanderdonckt and González Calleros, 2008]. Agrupa los elementos *perfil*, *actividad*, y *rol*, que son propios del usuario y por ello son factores externos al sistema;

2. **Plataforma**: compuesto por todos los dispositivos de hardware y el software que están disponibles para sostener la interacción del usuario con el sistema. Este componente se modela en función de los recursos computacionales que determinan la manera en la cual la información se procesa, se transmite, se muestra, y es manipulada por el usuario; y

3. **Ambiente**: se refiere a las condiciones físicas y sociales en las cuales se desarrolla la interacción. Así, entidades periféricas a la tarea en curso pueden alterar el comportamiento del sistema (e.g., durante una conferencia, las notificaciones de los dispositivos

de los asistentes pueden ser cambiadas a hápticas) y/o del usuario (e.g., dependiendo de su ubicación física, el sistema puede mostrar u ocultar información relevante para el usuario).

2.2.3. Contexto de uso de los sistemas multi-usuario

En el desarrollo de un sistema colaborativo deben considerarse las actividades de los colaboradores. Se denomina *conciencia de grupo* al conocimiento de las actividades de los demás colaboradores, que provee un contexto para las propias. Ésta puede clasificarse en cuatro tipos [Dourish and Bellotti, 1992]:

1. **Conciencia informal**: se refiere a la ubicación y al estado de disponibilidad de los colaboradores;
2. **Conciencia de grupo estructural**: consiste de toda aquella información de los roles, responsabilidades, posición social o profesional de los colaboradores;
3. **Conciencia social**: corresponde a los contextos social y conversacional de los colaboradores; y
4. **Conciencia de espacio del trabajo**: consiste de toda aquella información sobre la interacción de los colaboradores en el espacio de trabajo compartido.

Con base en la definición de *conciencia de grupo*, se extiende la definición de contexto de uso para sistemas multi-usuario a la terna de componentes (**grupo de colaboradores**, **conjunto de plataformas**, y **ambiente común**) [Decouchant et al., 2013a], de forma similar:

1. **Grupo de colaboradores**: corresponde a los arquetipos humanos que utilizarán el sistema colaborativo y que están involucrados en un proyecto común.
2. **Conjunto de plataformas**: compuesto por todos los dispositivos de hardware y soft-

ware que alojan y permiten un mejor funcionamiento del sistema colaborativo. Éstos pueden ser usados de forma individual por cada colaborador (e.g., una computadora personal) o de forma compartida por varios colaboradores (e.g., un pizarrón compartido).

3. **Ambiente común**: se refiere a las condiciones físicas y sociales que son externas al grupo de trabajo, pero que tienen una influencia en los colaboradores y/o en el sistema colaborativo. También, incluye un ambiente virtual compuesto por [Vanderdonckt et al., 2008]:

   - Un **escritorio**: que es un conjunto de aplicaciones disponibles para un colaborador dependiendo de su rol [Haake et al., 2010],
   - **Espacios de trabajo** públicos y privados de las aplicaciones colaborativas, y
   - **Objetos compartidos**: los archivos o parte de ellos (e.g., texto, imágenes, audio, vídeo) que son compartidos por los colaboradores a través del espacio de trabajo público de una aplicación colaborativa.

2.3. Ontología

En la década de los ochentas la Inteligencia Artificial concibió a las ontologías como la teoría de un mundo modelado y un componente de los sistemas de conocimiento [Guarino, 1995]. Particularmente, [Gruber, 2009] define las ontologías como un conjunto de representaciones primitivas para modelar un dominio de conocimiento o discurso. Una década después, [Guarino, 1995] definió una ontología como "una especificación explícita de una conceptualización", por lo cual, en los sistemas basados en conocimiento lo que existe es exactamente lo que puede ser representado.

[Horrocks, 2011] definió una ontología como un modelo del mundo, o al menos de algunos de sus aspectos. Por ello, es necesario tanto modelar el vocabulario relevante al dominio como especificar el significado de sus términos.

Cada una de las definiciones anteriores sitúan a las ontologías a un nivel semántico, puesto que los conceptos, las relaciones, y cualesquiera otros elementos (e.g., restricciones) definidos en un modelo ontológico tienen como fin representar la interacción entre tales elementos. Así, el objetivo de una ontología es dar significado a los elementos que se definen de forma independiente de las estructuras de datos disponibles en los lenguajes de programación utilizados para implementar una aplicación.

Es por este poder semántico que el uso de las ontologías se ha incrementado a través de los años. Proporcionan una conceptualización común para elementos que pueden parecer diferentes, pero que en realidad poseen características comunes que son percibidas mediante un motor de inferencia para ontologías. Esto permite modelar y unificar los diferentes aspectos del contexto de uso, es decir, los ambientes físico y computacional, los recursos y el modelo de usuario.

Sin embargo, hasta la fecha su diseño no se ha estandarizado y la creación de ontologías es todavía un proceso que puede realizarse de diferentes maneras. Sólo existen una serie de buenas prácticas, e.g., los consejos generales para solucionar los posibles problemas en la formalización de conceptos en el proceso de diseño propuestos por Noy y Hafner [Noy and Hafner, 1997] y una serie de principios de diseño sintetizados a partir de un exhaustivo estudio de los propuestos por varios autores realizados por [Morbach et al., 2009]. No obstante, dichos principios de diseño están sujetos a la interpretación de los mencionados autores del concepto de ontología.

Los principales conceptos del vocabulario ontológico, y su representación gráfica en este documento, son:

- **Individuo**: es un objeto del dominio de interés. Los individuos corresponden a las instancias de las clases, y se denotan por un óvalo punteado (véase Figura 2.1(a)).
- **Propiedad**: es una relación binaria entre dos individuos o entre un individuo y un valor. Se distinguen tres tipos:

1. **Propiedades de datos**: son aquellas propiedades que relacionan un individuo de una clase del dominio de interés a un dato de un esquema XML o a una literal RDF, i.e., una relación individuo-valor, y se denotan por un rectángulo con las esquinas redondeadas (véase Figura 2.1(b)).

2. **Propiedades de objeto**: son aquellas propiedades que relacionan dos individuos sin importar si pertenecen a una misma clase o a dos diferentes, i.e., una relación individuo-individuo, y se denotan por una línea (véase Figura 2.1(c)).

3. **Propiedades de anotación**: son aquellas propiedades que proporcionan información adicional sobre los individuos, propiedades y clases.

- **Clase**: es la representación concreta de un concepto. Así, las clases representan individuos similares entre ellos de alguna forma, y se denotan por un óvalo (véase Figura 2.1(d)).

- **Restricciones cuantificadores**: se definen sobre propiedades de objetos y pueden ser existenciales o universales. Una restricción existencial determina que un individuo pertenece a una clase determinada sólo si se relaciona con un individuo de tal clase; mientras que, las restricciones universales restringen el tipo de individuos con los cuales un individuo puede relacionarse a través de una propiedad de objeto.

- **Axioma de cerradura**: se define sobre una propiedad de objeto mediante una restricción universal, que delimita los tipos de individuos que se relacionan a través de esa propiedad a los considerados en las restricción existencial.

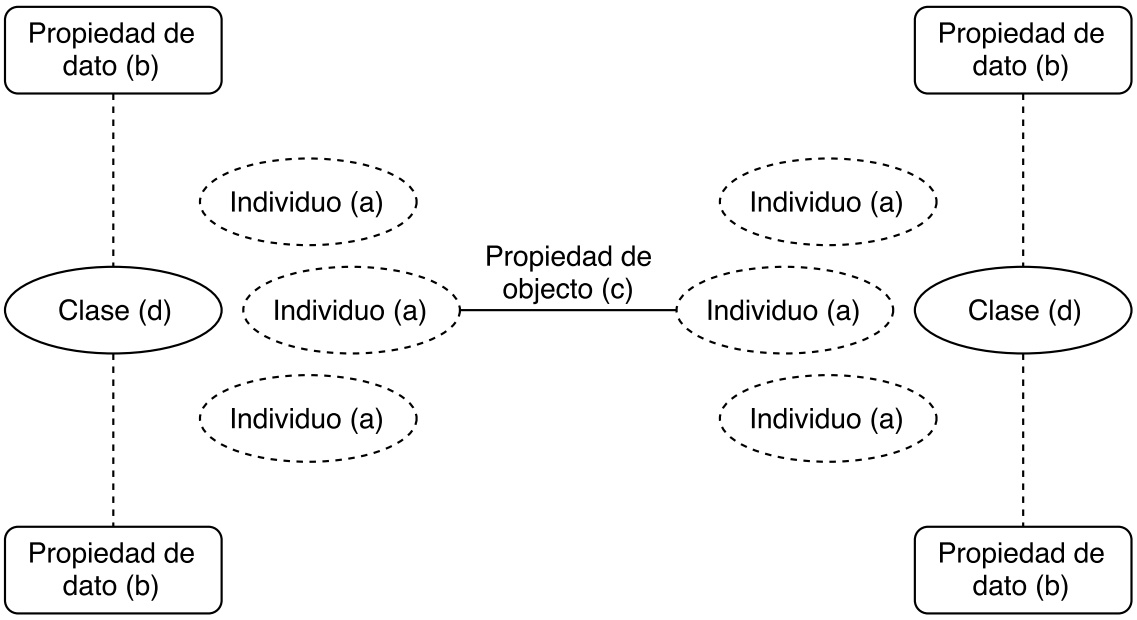


Figura 2.1: Representación gráfica de los elementos de una ontología

Como el objetivo principal de esta tesis es proponer un marco conceptual que sea capaz de detectar cambios en el contexto de uso de los sistemas colaborativos, en esta sección primero se describe brevemente varios sistemas conscientes de contexto y se ofrece un análisis comparativo de ellos. Entonces, se subrayan algunas características, ventajas, e inconvenientes de varios marcos propuestos para facilitar el desarrollo de sistemas conscientes de contexto.

2.4. Trabajo relacionado

Los sistemas conscientes de contexto generalmente pueden adaptar su operación al contexto de uso actual sin la intervención explícita del usuario, con el fin de incrementar su usabilidad y efectividad [Baldauf et al., 2007].

2.4.1. Aplicaciones

Call Forwarding [Want et al., 1992] proporciona a una recepcionista información sobre la ubicación del personal, con el fin de facilitar la tarea de dirigir las llamadas al lugar más

cercano en donde sus destinatarios están ubicados. Tal información incluye el nombre y ubicación de la respectiva persona, la extensión más cercana, y la probabilidad de hallarlos en tal ubicación.

Conference Assistant [Dey et al., 1999] es una aplicación que ofrece información útil a la gente que asiste a una conferencia. En particular, esta aplicación muestra las actividades de la conferencia y una tabla de tiempo, además subraya aquellas actividades que puedan ser de interés para cada usuario dependiendo de sus preferencias. Cuando un usuario ingresa en un cuarto, Conference Assistant le muestra en su laptop, el nombre y la biografía del orador, junto con el título de la lectura. Además, se notifica a cada usuario de las actividades actuales de sus colaboradores cercanos.

El sistema de control de temperatura para hogares de la French Electricity Company [Calvary et al., 2001] permite al usuario controlar la temperatura de varios cuartos. Esta aplicación adapta su interfaz de usuario al tamaño de la pantalla del dispositivo actual del usuario: 1) en una PC, las temperaturas de todos los cuartos son mostradas simultáneamente; 2) en un PDA, la temperatura de cada cuarto es mostrada en pestañas diferentes; y 3) en un teléfono móvil, la interfaz gráfica de usuario es completamente remodelada hacia una forma textual.

El sistema ConnecTables [Tandler et al., 2001] ofrece transiciones fluidas desde un espacio de trabajo individual a uno colaborativo y viceversa, permitiendo a los colaboradores acoplar y desacoplar dos tabletas personales, con el fin de crear dinámicamente un espacio compartido y uno privado, respectivamente. Los colaboradores pueden duplicar, arrastrar y soltar objetos (e.g., imágenes) de una tableta a otra cuando éstas están en el modo acoplado.

El sistema Roomware [Prante et al., 2004] añade las habilidades de cómputo y comunicación a objetos reales (e.g., paredes, mesas, y sillas) con el fin de explorar nuevas formas de interacción entre colaboradores. Particularmente, las aplicaciones de Roomware pueden ejecutarse en tres dispositivos especiales: 1) DynaWall, que es una gran superficie táctil, que sirve como un dispositivo de interacción integrado uniformemente en la pared por tres pi-

zarrones interactivos; 2) InteracTable que proporciona una pantalla de plasma táctil en una tabletop; y 3) CommChair, que combina la movilidad y la comodidad de los sillones con la funcionalidad de una computadora con un apuntador.

El sitio Sedan-Bouillon [Balme et al., 2005] promueve los sitios turísticos de las ciudades francesas de Sedan y Bouillon. Su principal función adaptativa permite al usuario tomar parte en la redistribución de la página principal del sitio entre un PC y un PDA, con el fin de facilitar la navegación. Este sitio Web soporta todos los tipos de redistribución, e.g., replicación total o distribución parcial de sus componentes gráficos (el título, el contenido y la barra de navegación de la página web) entre tales dispositivos.

UbiDraw [Vanderdonckt and González Calleros, 2008] es una aplicación de dibujo vectorial que adapta su interfaz de usuario mediante el despliegue, ocultamiento, redimensionado y reordenamiento de las barras de herramientas e iconos conforme a la tarea actual del usuario (e.g., el último icono presionado) y las preferencias para alguna tarea (e.g., un rango que representa la necesidad del usuario para un icono) y el espacio disponible de la pantalla. Se supervisa el estado de estas variables contextuales, con el fin de mostrar el mayor número de herramientas. La adaptabilidad en UbiDraw siempre resulta de la iniciativa del usuario.

UbiCicero [Ghiani et al., 2009] es una guía de museo consciente de la ubicación de los visitantes a quienes les proporciona servicios e información sobre las obras de arte. De esta manera, cuando un visitante ingresa a un nuevo cuarto, UbiCicero le proporciona un resumen de las obras de arte ahí exhibidas, y cuando éste se aproxima a una de ellas, le pregunta si desea información adicional. UbiCicero además soporta interacción multi-usuario mediante un juego colaborativo, que genera dos tipos principales de vistas. Así, los visitantes pueden observar en sus dispositivos móviles, el mapa de cada cuarto en el cual se muestran las obras de arte visitadas por cada equipo. Adicionalmente, en las pantallas públicas que están ubicadas en partes estratégicas, los visitantes pueden observar el mapa del museo completo con la ubicación de cada jugador y las obras de arte visitadas por cada equipo.

Nokia Situations[1] es una aplicación móvil que adapta el comportamiento del teléfono del

usuario al contexto actual de uso, que se detecta dependiendo de las situaciones especificadas por el usuario mediante una serie de reglas basadas en los sensores internos del teléfono. Una situación puede ser definida de acuerdo al día, tiempo, ubicación GPS, así como la disponibilidad de redes WiFi y dispositivos Bluetooth. Una vez que las situaciones han sido definidas, el usuario configura el comportamiento deseado del teléfono.

2.4.2. Marcos de desarrollo

CoCaMAAL (*Cloud-oriented Context-Aware Middleware in Ambient Assisted Living [AAL]*), [Forkan et al., 2014] es un marco especializado en escenarios de vida asistida por el ambiente (*AAL*) y consiste en una red amplia de sensores biomédicos y dispositivos embebidos. Para compensar su escaso poder de procesamiento, la gran cantidad de información generada es procesada a través de cómputo en la nube. Así, el principal objetivo de este marco es simplificar el flujo entre la recolección y el procesamiento de *Big Data*.

BDCaM (*Big Data for Context-aware Monitoring*), [Forkan et al., 2015] es una extensión de CoCaMAAL y consiste en un middleware consciente de contexto. Utiliza el cómputo en la nube para obtener los patrones y tendencias en la información individual de cada paciente, mediante probabilidades asociadas establecidas previamente. Además BDCaM realiza aprendizaje supervisado con la información contextual obtenida e inferida. Esta información se utiliza en la toma de decisiones por parte del paciente en procesos conscientes de contexto y en la detección eficiente de anomalidades.

EgoSENSE [Milić and Stojanović, 2017] es un middleware para Android destinado a soportar la implementación de servicios móviles conscientes de contexto, especifícamente para el monitoreo de la salud. Con el fin de reducir el flujo de grandes cantidades de información hacia el servidor y su coste de procesamiento, cada dispositivo Android asociado reúne, procesa y analiza la información del contexto localmente.

Propone métodos y técnicas para reunir y detectar el contexto del usuario en teléfonos

inteligentes, con base en sus sensores integrados, preferencias del usuario, su ambiente y sus actividades. Ilustran su viabilidad mediante el desarrollo de una aplicación de monitorización de la salud para la plataforma Android.

Se compone de cuatro capas independientes y personalizables, un servicio en segundo plano y una interfaz para el ajuste de parámetros del servicio consciente de contexto. Los componentes para la recolección de información de sensores, el procesamiento de eventos complejos, y el razonamiento y almacenamiento de información importante en dispositivos con recursos limitados se agrupan en tres subsistemas conectados: monitorización, razonamiento y ejecución.

La primera capa de la arquitectura reúne y deserializa en *DTO* (i.e. objetos para la transferencia de datos), la información de los sensores físicos a través de un proveedor de sensores.

La segunda capa clasifica la información y almacena sus valores en objetos previamente definidos para el análisis del contexto. Cada vez que se cambia un valor, se crea un evento para notificar a la tercera capa.

La tercera capa monitoriza, casi en tiempo real, flujos continuos de información gracias a la tecnología de software *CEP* (*Complex Event Processing*) que utiliza los sensores e información contextual disponible. Asper es la tecnología empleada para detectar cambios en los sensores, generar eventos y determinar si se satisfacen las condiciones para enviar una notificación a la cuarta capa.

La última capa, nivel de supervición, genera un mensaje apropiado si se satisfacen las condiciones para una advertencia o alarma, y se propaga a través del servicio en segundo plano hacia el usuario, a un servicio web, y hacia las partes interesadas.

Existen otros marcos genéricos que permiten al desarrollador decidir qué entidades y variables contextuales integrar, como el propuesto por [Haake et al., 2010] y CAADA.

El marco propuesto por [Haake et al., 2010] se compone por cuatro capas: 1) capa de conocimiento, 2) capa de estado, 3) capa de contextualización, y 4) capa de adaptación.

La capa de conocimiento contiene todo el conocimiento relevante y factual sobre el dominio del sistema y de la aplicación. Para su representación utiliza el OWL (*Web Ontology Language*).

La capa de estado contiene información sobre la situación actual obtenida por sensores, e.g., ambiente físico, ambiente de cómputo, recursos y modelo de usuario. A partir de la información constitucional de la capa de conocimiento se define un modelo que representa el estado en curso.

Las reglas de detección definen qué información usar de las fuentes externas e internas, así como de la información derivada del sistema. Además, mediante el uso de reglas de inferencia se puede inferir conocimiento abstracto a partir de la información de un estado concreto.

El modelo del estado actual se representa en un gráfico de estados $G_S = V, E, \sigma$, en donde los individuos y las clases corresponden a los vértices $v \in V$; las propiedades, a los ejes $e \in E$; y el grado de certeza se expresa mediante la función σ. Para su representación se utiliza un diagrama RDF.

La capa de contextualización define el enfoque como un subconjunto arbitrario del estado que representa el centro de atención actual. Mediante reglas de contextualización de la forma *if - then* se determina cuál subconjunto del estado actual es relevante para un determinado enfoque. Así, puede considerarse a la contextualización como un proceso de selección. Para su representación se utiliza un diagrama RDF.

Finalmente, la capa de adaptación contiene las reglas de adaptación de la forma *if - then* y, a partir del estado contextualizado, se identifican aquellas relevantes para cierto foco de atención. Usualmente incluyen operaciones para cambiar propiedades del ambiente colaborativo.

Validan su enfoque en cuatro situaciones colaborativas típicas: 1) co-ubicación, i.e. la situación donde varia gente está reunida en una ubicación física (e.g., sala de juntas) o virtual (e.g., artefacto compartido o espacio compartido de reunión), 2) co-acceso, i.e. la situación donde varia gente accede al mismo artefacto (e.g., un diseño en dibujo o un documento), 3) co-recomendacción, i.e. la situación donde las acciones implícitas o explícitas de los colaboradores son utilizadas para sugerir recursos de información potencialmente útiles para una tarea común, y 4) co-dependencia, i.e. la situación donde múltiples tareas, objetos o usuarios son dependientes (e.g., tareas que son dependientes de tareas siendo trabajadas por otros usuarios).

Este enfoque se usa a través de la implementación de la funcionalidad central del marco en un ambiente de ejecución (adaptativo) y proporcionando otra parte para integrar las aplicaciones colaborativas.

La funcionalidad central del marco consiste en el modelado por capas del contexto que debe ser representado explícita y continuamente, así como el proceso de adaptación. El modelado consiste en tres pasos (detección, contextualización y adaptación) que son implementados en un Servidor de Adaptación utilizando tres servicios principales (*Servicio de Detección*, *Servicio de Contextualización* y *Servicio de Adaptación*). El *Modelo de Contexto* consiste en diferentes Modelos GCF (Modelo de Dominio, Reglas de Detección, Estado Actual), las reglas de contextualización, el estado contextualizado, las reglas de adaptación y el estado actualizado. Para su uso y manipulación se hace uso de otros subservicios de contexto dedicados (Servicios GCF).

Por su parte, el marco CAADA (*Context-Aware Applicacion Development Aproach*) [Jaouadi et al., 2016] es un enfoque basado en un modelo semántico para el desarrollo de aplicaciones conscientes de contexto y consiste en el marco DONCIR (*Dynamic Observation and Notification framework for Context changes In Runtime*), que monitoriza dinámicamente los cambios del contexto y los notifica al sistema en tiempo de ejecución.

CAADA propone un meta-modelo para la representación del contexto y una arquitectura

para el desarrollo de aplicaciones conscientes de contexto bajo el paradigma ECA, i.e, cuando se produce un evento, se evalúan sus condiciones asociadas y, si se satisfacen, se realiza una acción.

El meta-modelo representa los aspectos: 1) estructural, i.e., los objetos físicos o conceptuales y sus propiedades, del contexto (*Entidad del Contexto*, *Propiedad del Contexto*, y *Asociación del Contexto*); 2) dinámico, i.e., las relaciones transitorias o pasadas entre los objetos (*Asociación Dinámica*, *Restricción Temporal* y *Propiedad del Contexto Histórica*); y 3) etológico, i.e., las condiciones y procesos que deben llevarse a cabo si éstas se satisfacen (*Foco*, *Condición*, y *Regla*).

El meta-modelo utiliza archivos XML, que convierte a clases Java, para capturar y almacenar mediciones del contexto. El lenguaje MOF especifica la sintaxis y estructura de los objetos del archivo XML, mientras que XQuery es utilizado para extraer información de ellos.

El marco DONCIR se forma por tres capas: 1) capa de administración del contexto, 2) capa de administración de los cambios del contexto y 3) capa de adaptación; y cinco componentes principales: 1) *Generador de Contexto*, 2) *Administrador de Contexto*, 3) *Controles*, 4) *Ejecutor de Contexto*, y 5) *Productor de Adaptaciones*.

El *Generador de Contexto* construye las instancias pertinentes del contexto con base en una plantilla alojada en un archivo XML y las asocia con su respectiva instancia de *Context Provider*, un módulo de software que reúne información de fuentes heterogéneas. El *Administrador de Contexto* identifica las relaciones entre las entidades comunes de las instancias del contexto, y después se asignan *controles* a fuentes específicas para monitorizar los cambios en cada entidad presente. Cuando se detecta un cambio, el *ejecutor de contexto* evalúa una serie de condiciones para obtener y enviar reglas al *Productor de Adaptaciones*. Finalmente, el *Productor de Adaptaciones* ejecuta las reglas recibidas y funciones genéricas creadas por el desarrollador.

Los sistemas previos han sido comparados de acuerdo a las siguientes dimensiones: 1) elementos objetivos del contexto de uso, 2) sistemas de una única entidad vs. multi-entidad, 3) variables contextuales, 4) tipo de adaptación, y 5) si están construidos a partir de un marco de desarrollo o desde cero.

2.4.3. Elementos del contexto de uso

En el dominio de los sistemas interactivos, el "contexto de uso" involucra tres elementos: a) el *usuario* que alude al ser humano que usará el sistema interactivo; la *plataforma* que se refiere al hardware y software disponibles de las computadoras del usuario que soportan la interacción usuario-sistema; y c) el *ambiente* que involucra las condiciones físicas y sociales en las cuales la interacción toma lugar [Vanderdonckt and González Calleros, 2008].

En particular, Call Forwarding, UbiCicero, Conference Assistant, UbiDraw y Nokia Situations se adaptan al elemento *usuario*. Dependiendo de la ubicación del usuario, Call Forwarding determina la extensión más cercana a la cual una llamada telefónica puede ser redirigida, mientras UbiCicero ofrece a sus usuarios una lista de las obras de arte expuestas en el cuarto en donde se encuentran así como detalles de las obras de arte más cercanas a ellos. Además de la ubicación del usuario, Conference Assistant emplea las preferencias del usuario para proporcionarle información relevante sobre la conferencia y sobre sus colegas cercanos. UbiDraw remodela su interfaz de usuario de acuerdo a la tarea actual del usuario, sus preferencias para una determinada tarea, su frecuencia y el tamaño de la ventana definido por el usuario. Nokia Situations adapta el comportamiento del dispositivo móvil de un usuario de acuerdo a sus preferencias.

Además, el sistema de control de temperatura, ConnecTables, y Roomware se adaptan al elemento *plataforma*. El sistema de control de temperatura es capaz de ejecutarse tanto como una aplicación Web como una aplicación de escritorio y permite al usuario consultar la temperatura de algunos cuartos desde dispositivos heterogéneos (i.e., PC, PDA, teléfono

móvil y reloj inteligente). ConnecTables permite a dos usuarios acoplar y desacoplar dos tabletas para crear un espacio compartido y otro privado, respectivamente. Roomware se puede ejecutar en tres dispositivos diferentes, pero sólo muestra espacios de trabajo privados en dispositivos personales (e.g., CommChair) mientras que los espacios de trabajo compartidos son mostrados en los dispositivos personales y públicos (e.g., DynaWall e InteracTable).

El sitio Web de Sedan-Bouillon Web y el marco propuesto por [Haake et al., 2010] se adaptan a los elementos *plataforma* y *usuario*. Al primero puede accederse desde una PC o una PDA, pero es capaz de detectar cuando el usuario está usando simultáneamente ambos dispositivos para acceder a la página Web correspondiente. En esta situación, el sistema le ofrece al usuario la posibilidad de redistribuir algunos widgets de la interfaz de usuario entre estos dispositivos (e.g., el título y el contenido de la página Web en la PC, y la barra de navegación en el PDA) con el fin de facilitar la interacción humano-computadora (e.g., el PDA puede actuar como un control remoto). El marco propuesto por [Haake et al., 2010] propone un modelo semántico del usuario y de las aplicaciones colaborativas.

Sólo CoCaMAAL, su extensión BDCaM, y EgoSENSE, se adaptan a los elementos *usuario* y *ambiente*. Los tres obtienen la información proveniente de los biosensores de cada paciente y de los dispositivos emebebidos en un ambiente de asistencia a la vida (AAL); CoCaMAAL y BDCaM la procesan mediante el cómputo en la nube y EgoSENSE en dispositivos Android.

Finalmente, CAADA, al ser genérico, es flexible con el desarrollador, pues le permite determinar a cuáles elementos adaptarse.

2.4.4. Sistemas de mono vs. multi-entidad

Los sistemas de una única entidad sólo consideran el contexto de una sola entidad, e.g., un usuario o un dispositivo, mientras que los sistemas multi-entidad consideran simultáneamente el contexto de múltiples entidades, e.g., el personal de una organización. Así, Call

Forwarding, Conference Assistant y UbiCicero pertenecen a la categoría de multi-entidad porque consideran las ubicaciones de múltiples usuarios y los estados de otras entidades, e.g., las actividades disponibles en una conferencia o las obras de arte visitadas en un museo. Debido a su naturaleza colaborativa, ConnecTables y Roomware son sistemas multi-entidad porque toman en cuenta la presencia de múltiples colaboradores y los estados de sus respectivos dispositivos (e.g., si ellos están acoplados/desacoplados o si son públicos/personales). Incluso aunque Sedan-Bouillon sólo soporta un usuario, este sitio Web corresponde a la categoría multi-entidad porque su ejecución puede ser compartida entre dos dispositivos cercanos cuando sea requerido. El marco propuesto por [Haake et al., 2010] también pertenece a la categoría de multi-entidad porque está diseñado para dar soporte a sistemas colaborativos y considera el contexto de múltiples usuarios, aplicaciones y recursos de información. De forma similiar, CAADA, ofrece soporte a múltiples entidades determinadas por el desarrollador. También, CoCaMAAL, BDCaM y EgoSENSE son sistemas multi-entidad porque consideran los signos vitales de múltiples pacientes, sus biosensores y dispositivos embebidos asociados.

Por el contrario, UbiDraw es un sistema de una única entidad, dado que sólo emplea el estado del usuario expresado por su tarea actual y sus preferencias, la frecuencia de uso de las herramientas y el tamaño de la ventana. El sistema de control de la temperatura también es un sistema de una única entidad porque sólo considera el estado de la plataforma, en particular el tamaño de la pantalla. Finalmente, aunque Nokia Situations supervise múltiples variables contextuales, también es un sistema de una única entidad debido a que todas esas variables conciernen a un solo teléfono móvil.

2.4.5. Variables contextuales

La mayoría de los sistemas utilizan variables contextuales físicas para su adaptación, e.g., ubicación, tamaño de la pantalla, proximidad, tipo de dispositivo, tiempo y día. Como la mayoría de los sistemas colaborativos, ConnecTables y Roomware soportan las variables lógicas típicas, tales como la presencia de los colaboradores en el espacio de trabajo compartido. Algunos

otros sistemas, como Conference Assistant, UbiDraw, CoCaMAAL, BDCaM y EgoSENSE toman en cuenta variables lógicas más sofisticadas, e.g., las preferencias de los usuarios, la frecuencia de uso de las herramientas, y los signos vitales de los pacientes, respectivamente.

Por su parte, los marcos propuestos por [Haake et al., 2010] y CAADA, sólo utilizan algunas variables contextuales físicas y lógicas para ejemplificar su uso en las cuatro situaciones colaborativas típicas descritas y en un ambiente médico, respectivamente. El primero considera, entre otras, la ubicación de los usuarios, las preferencias de los usuarios y los dispositivos en uso; el segundo toma en cuenta el nombre, la categoría y la experiencia de los doctores, así como el nombre, el género y la fecga de nacimiento de los pacientes.

2.4.6. Tipos de adaptación

De acuerdo a [Abowd et al., 1999], la adaptabilidad de los sistemas conscientes de contexto se hace aparente en las siguientes formas:

- **Presentación de información y servicios al usuario**: se refiere a una técnica de interacción que muestra una lista de objetos (e.g., impresoras) o lugares (e.g., oficinas) cuyos elementos más importantes son enfatizados o más fáciles de seleccionar, de acuerdo al contexto de uso.
- **Ejecución automática de un servicio**: en este caso, no sólo se presenta información relevante, también se ejecuta automáticamente un servicio cuando se presenta la combinación correcta de condiciones contextuales.
- **Adición de información**: la información contextual puede ayudar al usuario a mejorar su entendimiento del ambiente colaborativo en el cual está trabajando, e.g., identificar a las personas más activas en ciertas situaciones.

Call Forwarding sólo añade información sobre la ubicación del personal y su extensión más

cercana, pero no resalta ningún elemento específico ni ejecuta ningún servicio automáticamente.

El sistema de control de la temperatura, UbiDraw y el sitio Web de Sedan-Bouillon se enfocan en la presentación de información y de servicios. El sistema de control de temperatura, dependiendo del área disponible de la pantalla, puede mostrar simultáneamente todos los cuartos o resaltar el preferido por el usuario. UbiDraw reacomoda sus herramientas y resalta aquellas usadas con frecuencia mayor. Y el sitio Web de Sedan-Bouillon, al detectar dos dispositivos cercanos que acceden a él, le proporciona al usuario una meta-interfaz que le permite seleccionar los widgets de la interfaz de usuario que se mostrarán en cada dispositivo.

Roomware se centra en la ejecución automática de servicios, ya que muestra los espacios de trabajo compartidos, tanto en los dispositivos públicos como en los dispositivos personales, y los espacios de trabajo privados sólo en los personales. ConnecTables también se encuentra en esta categoría porque transforma dos espacios de trabajo privados en uno compartido cuando se acoplan dos tabletas y viceversa. En el primer caso, el espacio de trabajo compartido permite a los colaboradores copiar y mover objetos desde una tableta a la otra.

UbiCicero añade a la información, disponible para el usuario, las ubicaciones de otros usuarios y las obras de artes que han visitado. Además, al detectar que un usuario entra a un cuarto o está cerca de una obra de arte, el sistema ejecuta un servicio automáticamente que le proporciona mapas e información relevante. Por lo cual, UbiCicero muestra dos tipos de adaptación: la ejecución automática de servicios y la adición de información.

Conference Assistant no sólo le proporciona al usuario información adicional de las actividades en curso de sus colegas cercanos, también pone énfasis en aquellas actividades que pueden ser de interés para el usuario. De esta forma, este sistema pertenece a las categorías de adición de información y de presentación de información y servicios.

El tipo de adaptación en Nokia Situations pertenece a la ejecución automática de servicios porque realiza el cambio correspondiente en la configuración del teléfono, cuando se

detecta una situación definida por el usuario. No obstante, este sistema no resalta elementos ni añade información.

Los marcos propuestos por [Haake et al., 2010] y CAADA no proporcionan ningún tipo de adaptación. El desarrollador decide e implementa qué acciones de adaptación llevar a cabo para su sistema.

Finalmente, la información obtenida por CoCaMAAL y BDCaM influye en la toma de decisiones por parte del paciente en procesos conscientes de contexto y detecta tempranamente anomalías en su estado. EgoSENSE, filtra la información contextual para determinar cuándo se ha producido una advertencia o alarma y la notifica tanto al usuario, a un servidor web y demás entidades interesadas.

2.4.7. Desarrollo basado en marco vs. desarrollo desde cero

La mayoría de los sistemas estudiados fueron desarrollados desde cero porque se basan en arquitecturas específicas. Específicamente, el marco propuesto por [Haake et al., 2010] es sólo conceptual. El sistema de control de temperatura y Roomware se construyeron utilizando los marcos ARTStudio [Calvary et al., 2001] y Roomware [Prante et al., 2004], respectivamente. EgoSENSE, utiliza el marco Funf, los servicios de Google y la tecnología Asper para su implementación. No obstante, los creadores de Nokia Situations no proporcionan información sobre la infraestructura de desarrollo que emplearon.

2.4.8. Análisis de las entidades y sus variables contextuales de las aplicaciones y los marcos mono y multi-usuario

Con el propósito de extender el modelo semántico del espacio colaborativo global, se identificaron las entidades y sus respectivas variables contextuales de las aplicaciones y de los marcos expuestos en el estado del arte. La Tablas 2.1 y 2.2 presentan las entidades y sus

respectivas variables contextuales para los aplicaciones y los marcos mono y multi-usuario, respectivamente.

Cuadro 2.1: Entidades y sus variables contextuales de las aplicaciones y los marcos mono-usuario

Sistema	**Tipo**	**SC**	**Entidades**	**Variables contextuales**
Call Forwarding	Aplicación	No	Miembros del personal	Ubicación
Conference Assistant	Aplicación	Sí	Participantes	Ubicación Preferencias Proximidad Actividades actuales de los participantes cercanos
			Conferencias	Detalles
Heating Control	Aplicación	No	Dispositivo	Tamaño de la pantalla
ConnecTables	Aplicación	Sí	Vistas	Compartición Sincronización Perspectiva Modos de acoplamiento y desacoplamiento

Cuadro 2.2: Entidades y sus variables contextuales de las aplicaciones y los marcos multi-usuario

Sistema	**Tipo**	**SC**	**Entidades**	**Variables contextuales**
Roomware	Aplicación	Sí	Colaboradores	Ubicación
			Dispositivos	Ubicación
Sedan-Bouillon	Aplicación	No	Usuario	Preferencias en la distribución de los *widgets*
			Dispositivos	Proximidad
UbiDraw	Aplicación	No	Usuario	Preferencias
UbiCicero	Aplicación	Sí	Jugadores	Ubicación
			Equipos	Obras de arte visitadas por un equipo
Nokia Situations	Aplicación	Sí	Teléfono inteligente	Tiempo Día Ubicación GPS Disponibilidad de redes Wi-Fi
Marco propuesto por [Haake et al., 2010]	Marco	Sí	Definidas por el desarrollador	Definidas por el desarrollador
CoCaMAAL	Marco	Sí	Pacientes	Ubicación Signos vitales
			Sensores	Signo vital asociado Variable física del ambiente
			Cuartos	Temperatura Luz Humedad
BDCaM	Marco	Sí	Pacientes	Signos vitales
CAADA	Marco	Sí	Definidas por el desarrollador	Definidas por el desarrollador
EgoSENSE	Marco	Sí	Usuario	Ubicación Velocidad de los movimientos del usuario Temperatura Pulso Presión

Capítulo 3

ANTECEDENTES

Este capítulo describe los trabajos previos sobre los cuales se sustenta esta tesis, específicamente la Sección 3.1 describe la arquitectura para el soporte de contexto de uso en sistemas colaborativos; la Sección 3.2, la arquitectura RAMS así como sus correspondientes ontologías; y la Sección 3.3 describe el marco de desarrollo XARE para sistemas conscientes de contexto junto a los escenarios y las aplicaciones estudiadas para su creación.

3.1. Arquitectura para el soporte de contexto de uso en sistemas colaborativos

[Olivares Toledo, 2011] se centra en estudiar el contexto de uso en ambientes de trabajo colaborativo, enfatizando la importancia de la adaptabilidad de los sistemas colaborativos con base en diversas variables típicas de los grupos de trabajo, tales como el estado de los proyectos, las políticas organizacionales, la ubicación física de los colaboradores y los recursos disponibles. Para soportar e integrar el contexto de uso en sistemas colaborativos, propone una arquitectura contextual basada en escenarios y situaciones reales que sirven como medio para validar su funcionalidad.

3.2. Arquitectura RAMS

[García García, 2013] desarrolló un soporte computacional para la compartición de recursos físicos (e.g., salas de juntas, salones de clases y auditorios), virtuales (e.g., scanners, plotters y dispositivos móviles) y humanos ubicados en una organización, siendo que éstos pueden estar distribuidos en diferentes oficinas, edificios o dispositivos de almacenamiento, dependiendo de su naturaleza.

La arquitectura RAMS es capaz de explotar las tecnologías actuales, con el fin de proporcionar a las personas la experiencia de un ambiente ubicuo en el cual se comparten recursos. Utiliza un enfoque semántico para encontrar los mejores recursos disponibles para una petición, considerando la interacción entre las personas involucradas y el contexto tanto del colaborador proveedor como del solicitante. Dicha interacción incluye la evaluación de derechos de acceso, restricciones de uso, relaciones de posesión y la disponibilidad tanto del dueño como de los recursos. El enfoque semántico de la arquitectura RAMS proporciona diversas ventajas al realizar el descubrimiento de recursos, ya que pueden ser calculadas relaciones no evidentes entre las entidades participantes, lo cual ayuda a proporcionar respuestas más precisas a las peticiones de usuarios.

Dicho enfoque semántico se logra al modelar las entidades participantes en un conjunto de ontologías, las cuales han sido diseñadas genéricamente para ser implementadas en cualquier tipo de organización, sin importar su tamaño ni su tipo. Este conjunto de ontologías almacenan información estática sobre dichas entidades, la cual corresponde a sus características técnicas o capacidades y algunos elementos de información dinámica proveniente del medio ambiente. Está información dinámica es obtenida por un mecanismo de reconocimiento humano que determina la presencia y ubicación de las personas, por medio de la identificación de su voz y cara, así como de aplicaciones que permiten a los usuarios cambiar el estado actual de sus recursos y de ellos mismos. La experiencia ubicua, que la arquitectura RAMS ofrece, es proporcionada por el mecanismo de reconocimiento humano y un soporte para localizar el recurso físico disponible más cercano al solicitante

[García García, 2013][García García et al., 2016].

3.2.1. Modelo semántico de la arquitectura RAMS

Consta de seis ontologías, tres de las cuales representan los recursos administrados por la arquitectura RAMS, i.e., los recursos humanos, físicos y virtuales. La cuarta ontología es la representación semántica del modelo de contexto de [Kirsch-Pinheiro et al., 2004]. La quinta almacena información de la institución en la cual la arquitectura RAMS se despliega. Finalmente, la sexta ontología "Metaontología" enlaza las ontologías anteriores mediante propiedades de objetos que relacionan individuos de clases que pertenecen a la misma o a diferentes ontologías.

Todas las clases en las ontologías de RAMS son disjuntas porque cada una describe entidades con características únicas. Tales clases pueden ser parte de la misma jerarquía, pero cada uno es esencialmente diferente. Así, ninguno de los individuos de la arquitectura RAMS puede pertenecer a más de una clase [García García et al., 2013].

3.2.2. Ontología *HumanResource*

Se compone por una clase principal homónima que representa la gente involucrada en el proceso de descubrimiento de recursos. Esta gente comparte alguna de su información personal, ya sea proporcionándola explícitamente o aceptando ser observada por herramientas que obtienen tal información de una forma semi- o automáticamente [García García, 2013].

Se compone por las siguientes subclases:

- ***Visitor***: representa a la gente que no pertenece a la organización pero que permaneceré por algunos meses (representada en la subclase *LongTermVisitor*) o sólo unos días (representada en la subclase *ShortTermVisitor*), y

- ***SupportStaff***: representa a la gente que trabaja en alguno de sus departamentos. Se divide en las siguientes subclases: *Administrator*, *SecurityPersonnel* y *CleaningPersonnel*.

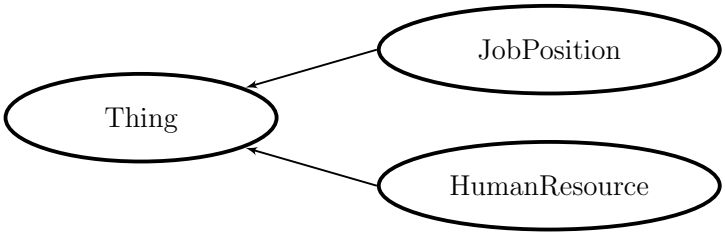


Figura 3.1: Diagrama ontológico *HumanResource*

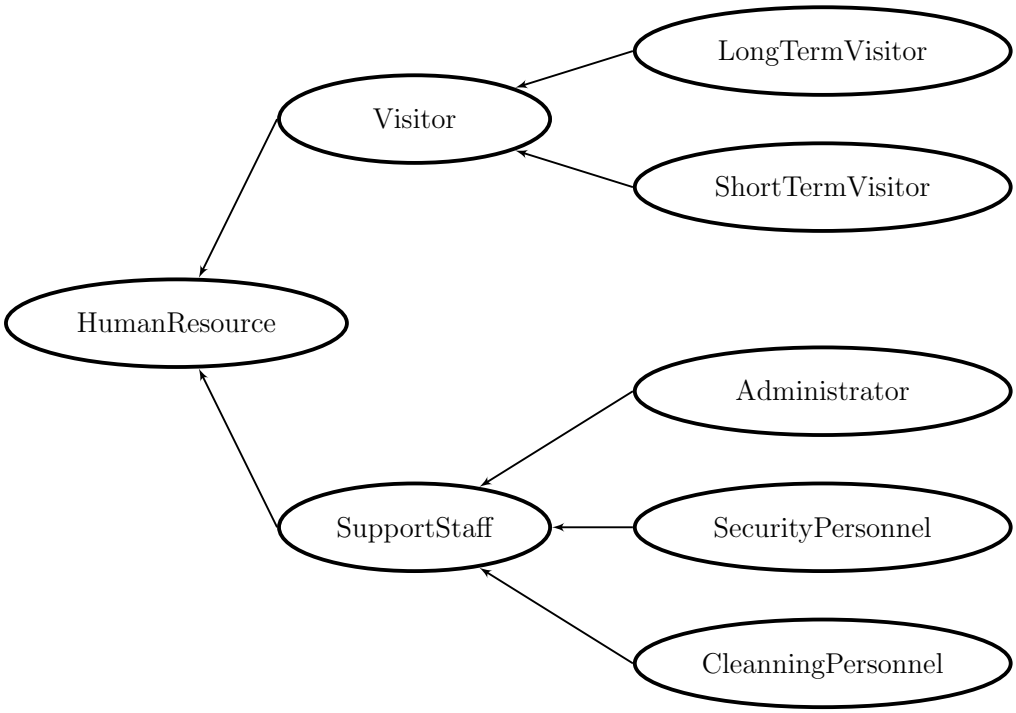


Figura 3.2: Diagrama de clases de la ontología *HumanResource*

3.2.3. Ontología *PhysicalResource*

Se compone por una clase principal homónima que agrupa todos los tipos de recursos físicos administrados por la arquitectura RAMS. De ésta, se extienden dos subclases *Building*

y *Hardware*. La primera agrupa los departamentos en la organización que están asociados a un edificio determinado, e.g., salones de clases y salas de reuniones [García García, 2013].

La clase *Hardware* contiene clases que clasifican los individuos que representan dispositivos reales compartidos entre colegas (e.g., computadoras y proyectores) y está compuesta por trece subclases: *Plotter*, *OpticalDrive*, *ElectronicalSignaturePad*, *MultimediaHardware*, *Handheld*, *Scanner*, *Mutlifunctional*, *ExternalStorage*Projector, *Camera*, *Computer Display* y *Printer*.

La clase *Handheld* representa a los dispositivos que pueden ser sujetados en una mano y está compuesta por las subclases *Tablet*, *Smartphone*, y *PDA*. En tanto, la clase *Computer* representa diferentes tipos de computadora y de ella derivan las subclases *Laptop*, *Desktop* y *Server*.

Figura 3.3: Diagrama ontológico *PhysicalResource*

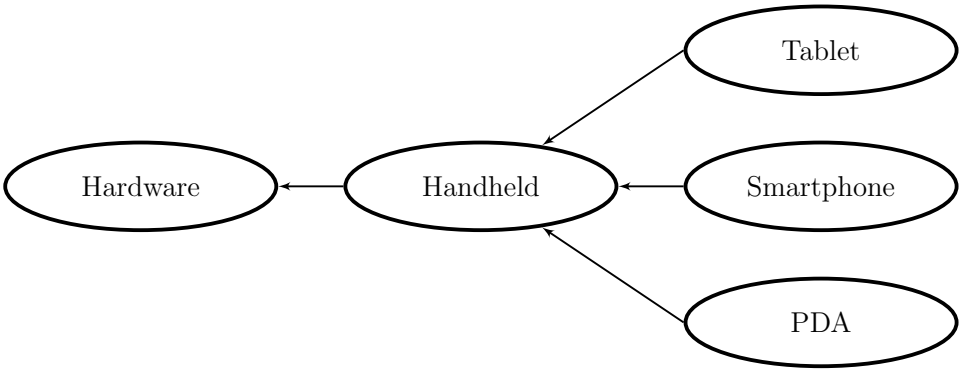


Figura 3.4: Diagrama de clases de la ontología *Handheld*

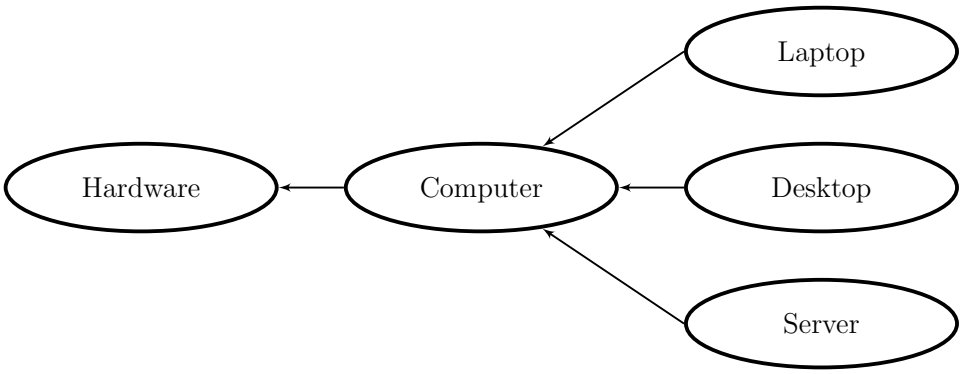


Figura 3.5: Diagrama de clases de la ontología *Computer*

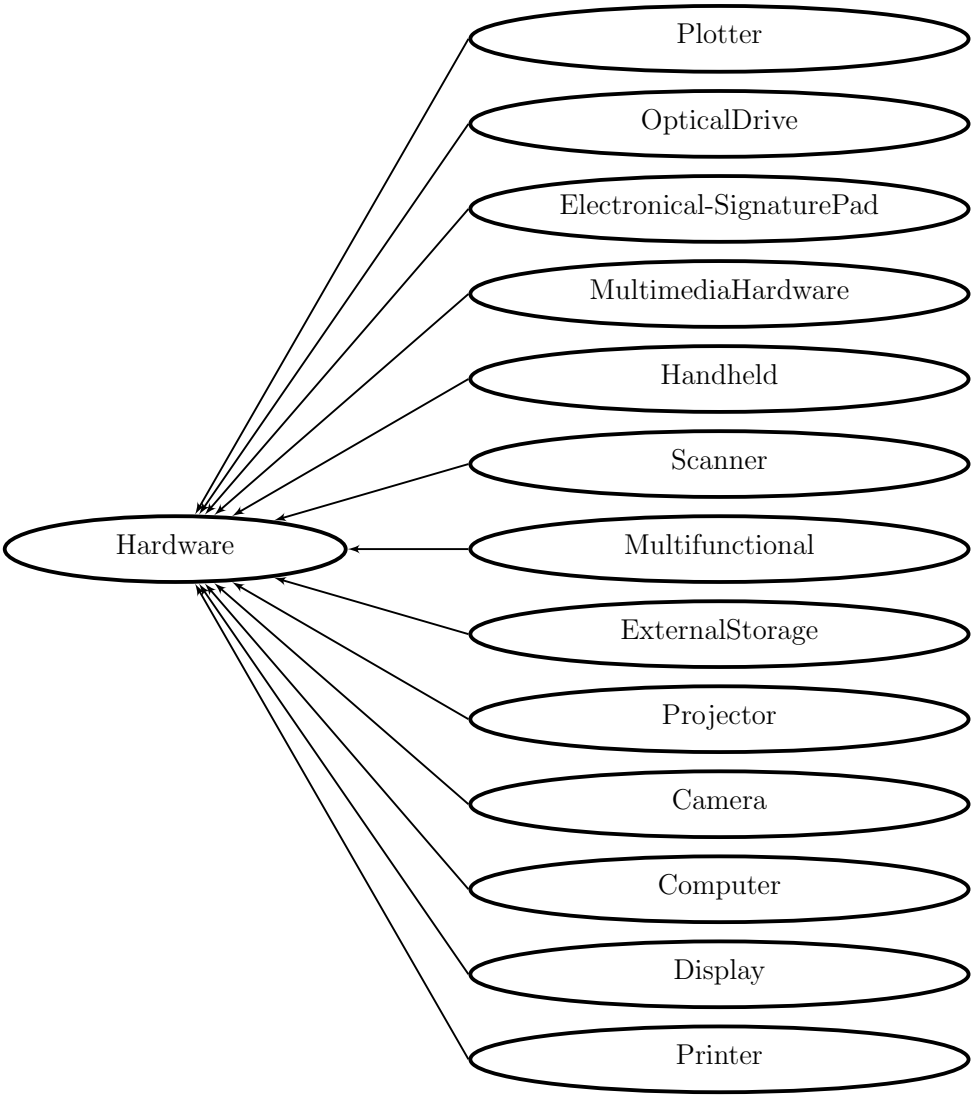


Figura 3.6: Diagrama de clases de la ontología *Hardware*

3.2.4. Ontología *VirtualResource*

Se compone por una clase principal homónima que agrupa todos los tipos de recursos virtuales compartidos entre colegas y está compuesta por cuatro subclases: *Files*, *DriverPlugins*, *Databases* y *Software* [García García, 2013].

La clase *Files* contiene las subclases *Documents*, *Forms*, *Manuals* y *Multimedia*.

La clase *Software* agrupa las clases que representan los tipos de software que pueden ser compartidos en la organización y son: *Tools*, *OperatingSystems*, *ProgrammingTools*, *OfficeApplications*. *Antivirus* y *MediaPlayer*. A su vez, la subclase *Tools* agrupa el software que ayuda a los usuarios a realizar tareas específicas en la siguientes subclases: *Burners*, *ArchiveCompressor Browsers* y *Media*. Finalmente, la subclase *OfficeApplication* contiene las subclases *WordProcessor*, *Presentation* y *SpreadSheet*.

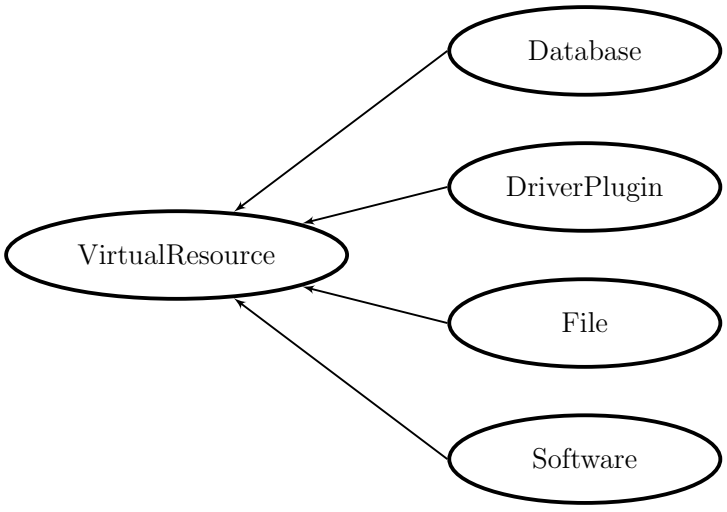


Figura 3.7: Diagrama ontológico *VirtualResource*

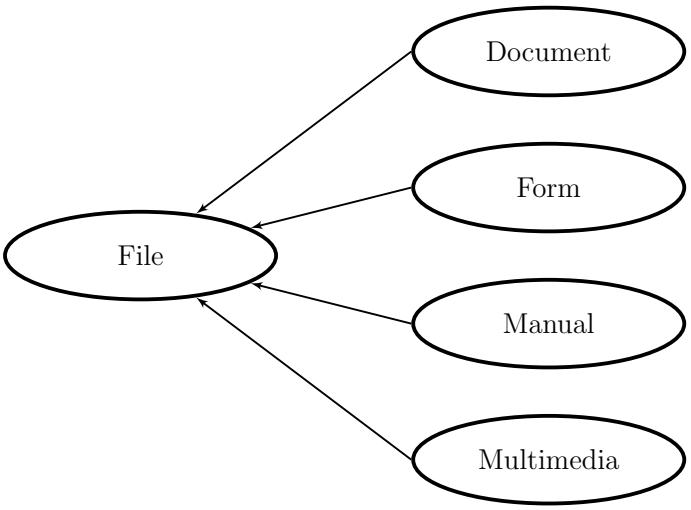


Figura 3.8: Diagrama de clases de la ontología *File*

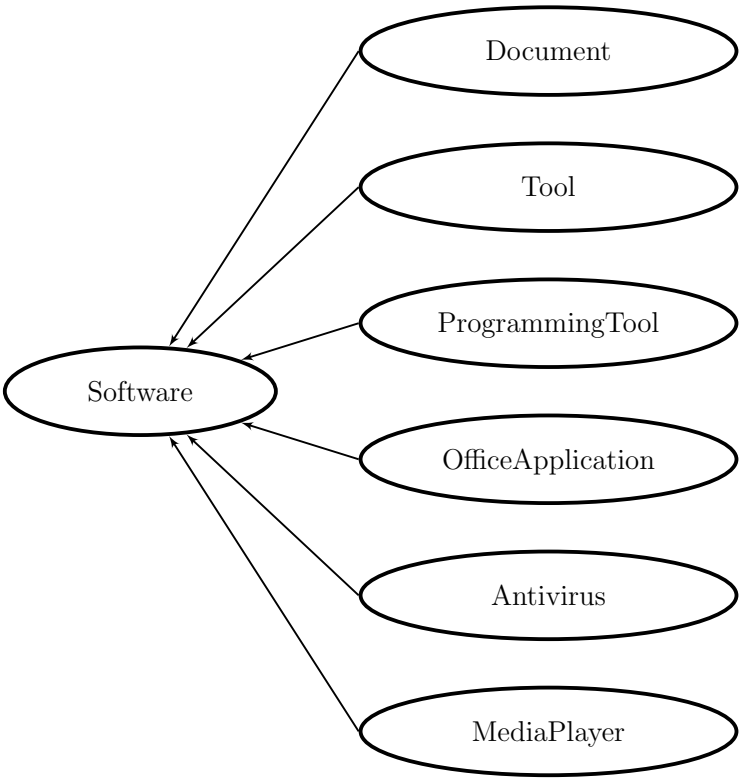


Figura 3.9: Diagrama de clases de la ontología *Software*

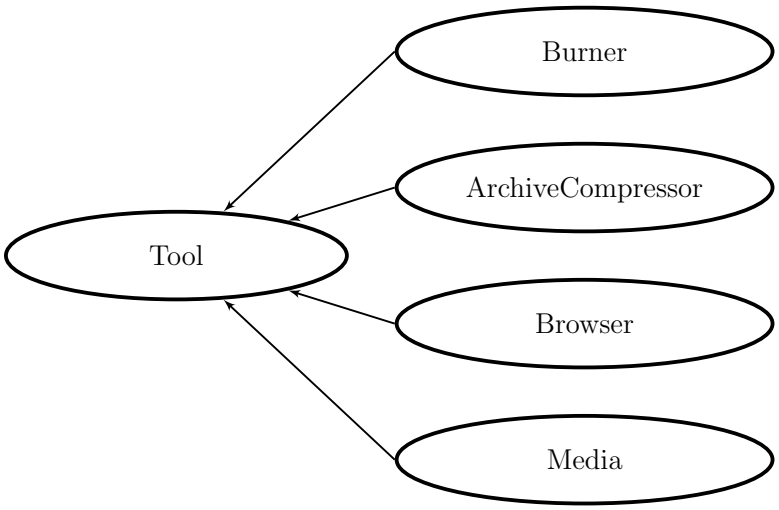


Figura 3.10: Diagrama de clases de la ontología *Tools*

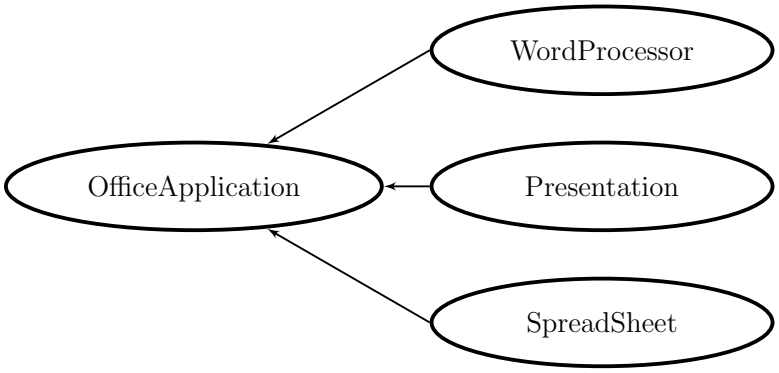


Figura 3.11: Diagrama de clases de la ontología *OfficeApplication*

3.2.5. Ontología *Context*

Contiene las variables que describen el ambiente y se constituye por dos clases principales: *OrganizationalContext* y *PhysicalContext* (véanse las Figuras 3.12 y 3.13). Ese modelo considera cinco características a fin de representar el contexto: espacio, herramienta, comunidad, tiempo y proceso. El objetivo de este modelo es representar la información de consciencia para un conjunto de actividades (procesos) ejecutados asíncronamente (tiempo) por un grupo de colaboradores (comunidad) utilizando una herramienta (dispositivo y aplicación) en una ubicación física (espacio) [García García, 2013].

La clase *OrganizationalContext* se compone por las siguientes subclases:

- ***Process***: representa una meta final (e.g., una reunión) que puede ser alcanzada por un colaborador o un grupo,
- ***Activity***: representa una actividad que puede ser realizada en un proceso,
- ***Group***: representa un conjunto de colaboradores,
- ***Role***: representa la parte que el colaborador tiene que jugar cuando realiza una acción, y
- ***Calendar***: representa el horario de un proceso.

La clase *PhysicalContext* se compone por las siguientes subclases:

- ***Restriction***: representa el valor actual de las restricciones de uso definidas por un productor para especificar una restricción de uso,
- ***Task***: representa el tipo de acciones que un individuo puede realizar sobre un recurso virtual o físico,
- ***AvailabilityMode***: esta clase enumerada se compone por los individuos *okToDisturb* y *doNotDisturb* que especifican si un individuo de la clase *HumanResource* acepta ser interrumpido o no,
- ***Company***: esta clase enumerada se compone por los individuos *accompanied* y *alone* que especifican si un individuo de la clase *HumanResource* está actualmente solo o no, y
- ***UsageStatus***: esta clase enumerada se compone por los individuos *free* y *inUse* que especifican si un individuo de la clase *PhysicalResource* o de la clase *VirtualResource* está actualmente en uso o no.

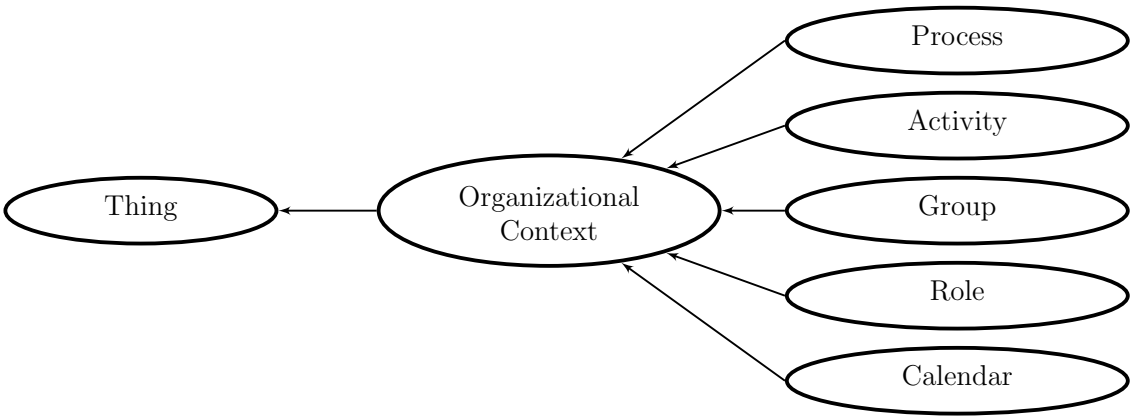


Figura 3.12: Diagrama de clases de la ontología *OrganizationalContext*

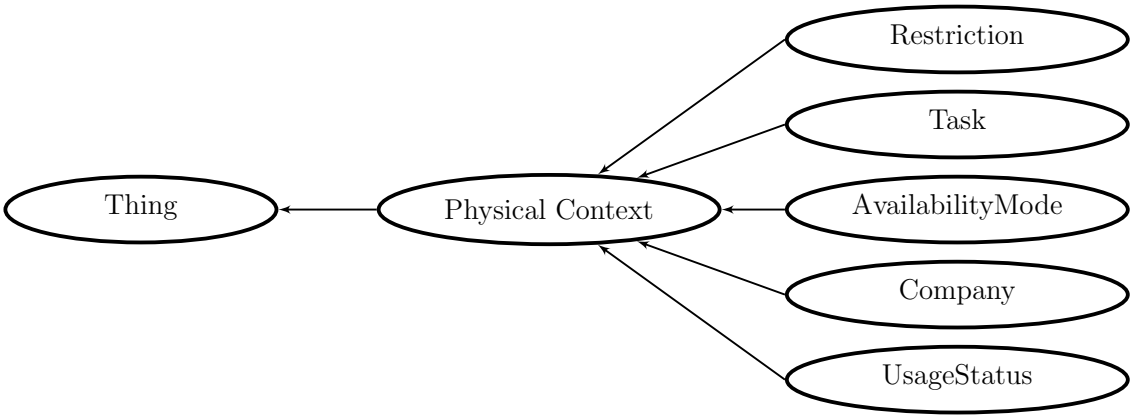


Figura 3.13: Diagrama de clases de la ontología *Context*

3.2.6. Ontología *InstitutionInformation*

En su clase principal *Department* representa los departamentos existentes en la organización en la cual la arquitectura RAMS está implementada [García García, 2013].

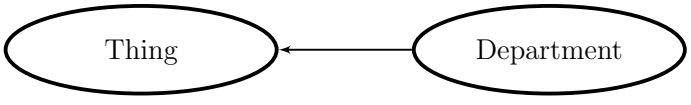


Figura 3.14: Diagrama ontológico *InstitutionInformation*

3.2.7. Metaontología

Relaciona las cinco ontologías anteriores mediante propiedades de objetos.

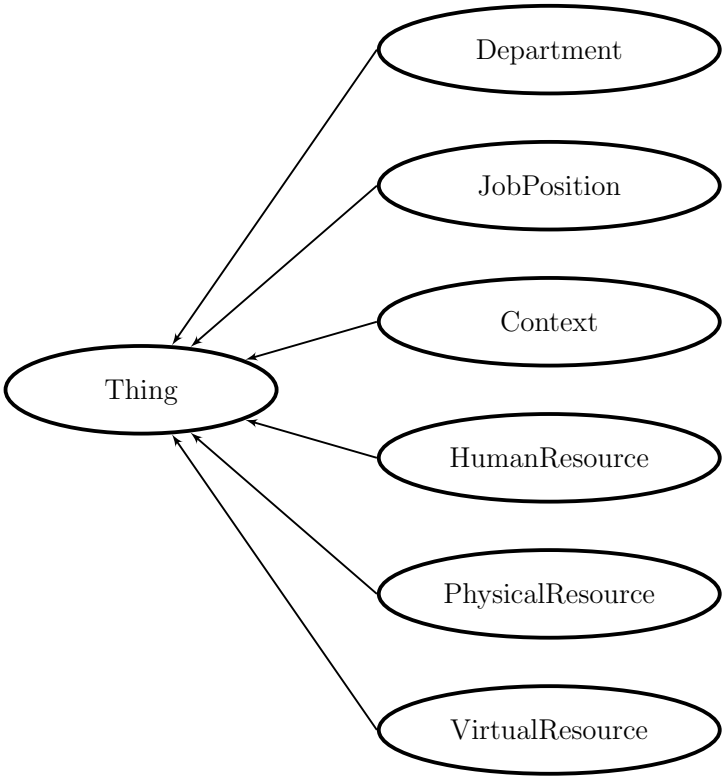


Figura 3.15: Diagrama ontológico *MetaOntology*

3.3. Marco de desarrollo XARE

Actualmente, el creciente surgimiento y adquisición de dispositivos de cómputo, junto al progreso de las redes de comunicación, permiten al usuario actual desenvolverse en un ambiente variable y que utiliza dispositivos de cómputos diversos, con el objetivo de satisfacer sus necesidades. Ante tal variedad de dispositivos, tanto fijos como móviles, las aplicaciones requieren ser capaces de adaptarse a los cambios contextuales.

La basta investigación de la adaptación contextual en sistemas mono-usuario ha generado la definición de siete dimensiones entre las cuales destaca la adaptación al con-

texto de uso, pues está en función de al menos dos de los siguientes tres componentes [Sánchez Morales, 2013]:

- **Grupo de colaboradores**: corresponde a los arquetipos humanos que utilizarán el sistema colaborativo y que están involucrados en un proyecto común,
- **Conjunto de plataformas**: compuesto por todos los dispositivos de hardware y software que alojan y permiten un mejor funcionamiento del sistema colaborativo, y
- **Ambiente común**: se refiere a las condiciones físicas y sociales que son externas al grupo de trabajo, pero que tienen una influencia en los colaboradores y/o en el sistema colaborativo. También, incluye un ambiente virtual compuesto por un escritorio, espacios de trabajo públicos y privados, y objetos compartidos.

Con base en lo anterior, [Sánchez Morales, 2013] diseñó XARE (conteXt-Aware groupwaRE), un marco de desarrollo para sistemas colaborativos conscientes de contexto que considera cinco de las siete dimensiones previamente mencionadas. Éste se especializa en la plasticidad de de las Interfaces de Usuario (IU). Además, este marco está acompañado por un conjunto de clases y un mecanismo que facilitan su implementación.

La técnica empleada para la creación del marco de desarrollo XARE es la de *bottom-up*. Así, se partió de un conjunto de siete escenarios en los cuales se identificaron los requerimientos de algunas aplicaciones colaborativas adaptables al contexto. A cada escenario le corresponde al menos una aplicación colaborativa, como se muestra en la Tabla 3.1.

Cuadro 3.1: Relación entre escenarios y aplicaciones

Escenario	Aplicaciones colaborativas
1. Mejoramiento del trabajo colaborativo	Escritorio enriquecido
2. Remodelación de la interfaz de usuario	Herramienta de votación Editor de mapas mentales Editor de dibujos Herramienta de medios de contacto y disponibilidad Escritorio enriquecido
3. Redistribución de una aplicación colaborativa	Barra de colaboradores Herramienta de votación Editor de mapas mentales Editor de dibujos
4. Mejoramiento de la conciencia grupal	Herramienta de contacto y disponibilidad
5. Manipulación de la información	Editor de textos
6. Adaptación de la información	Administrador de contenidos vía *NFC*
7. Enriquecimiento con notificaciones multimodales	*(Ninguna)*

3.3.1. Aplicaciones

Para facilitar la comprensión de los escenarios, primero se describirán brevemente las aplicaciones identificadas:

1. **Barra de colaboradores**: muestra la conciencia del grupo a través de la foto y el nombre de los colaboradores pertenecientes a un equipo de trabajo. De acuerdo a su ubicación física los organiza como: *presentes*, *virtualmente presentes* y *ausentes* [Romero Gama et al., 2013].

2. **Herramienta de votación**: permite a un grupo de colaboradores realizar y tomar parte en una votación electrónica, de forma anónima o conocida [Romero Gama et al., 2013]. Permite tanto la interacción colocalizada como la distribuida, muestra las funcionali-

dades relacionadas al rol de cada colaborador y utiliza la *barra de colaboradores* para proporcionar conciencia de grupo.

3. **Editor de mapas mentales**: permite a un grupo de colaboradores crear diagramas para organizar ideas [Romero Gama et al., 2013]. Permite tanto la interacción colocalizada como la distribuida, muestra las funcionalidades relacionadas al rol de cada colaborador y utiliza la *barra de colaboradores* para proporcionar conciencia de grupo.

4. **Editor de dibujos**: permite a un grupo de colaboradores crear imágenes vectoriales soportando las transiciones del trabajo individual al colaborativo y viceversa. De esta manera, es posible que los colaboradores pasen de su respectivo espacio privado a un espacio común, en el cual pueden interactuar a través de un dispositivo con pantalla grande (e.g., un pizarrón interactivo o un arreglo de *tablets*). La aplicación, por su parte, adapta sus funcionalidades para dar soporte a la interacción de múltiples usuarios, en función de sus roles y jerarquías.

5. **Herramienta de medios de contacto y disponibilidad**: muestra la disponibilidad de un colaborador junto a sus respectivos medios de contacto accesibles. Para ello, considera parámetros tanto del colaborador a contactar y de quien desea contactarlo [Decouchant et al., 2013b].

6. **Escritorio enriquecido**: permite a un grupo de colaboradores relacionar objetos manipulados en distintas aplicaciones a través de referencias deícticas. Éstas, preservan siempre el contexto de dichos objetos [Decouchant et al., 2013b].

7. **Administrador de contenidos vía *NFC***: mediante tarjetas *NFC* (*Near Field Communication*) proporciona información relevante de acuerdo al tiempo y lugar especificados a un grupo de colaboradores que participen en una reunión previamente planificada.

8. **Editor de textos**: previene la filtración de información mediante el ocultamiento de información restringida ante personas no autorizadas, ubicadas cerca del área de su despliegue.

3.3.2. Escenarios

En esta sección se describen los escenarios identificados en el marco de desarrollo XARE:

1. **Mejoramiento del trabajo colectivo**: enriquece la colaboración entre los colaboradores al siempre mantener el contexto de los objetos compartidos en el espacio de trabajo y, al mismo tiempo, encontrar relaciones entre éstos con base en el uso de los objetos. Tales relaciones permiten formular sugerencias pro-activas (e.g., presentar a los miembros del equipo aplicaciones adecuadas para manejar un objeto dado, acorde a su rol, o enlazar diversas conversaciones textuales que estén en curso y que hagan referencia a tal objeto). Este escenario se asocia con la herramienta *escritorio enriquecido*.

2. **Remodelación de componente de la Interfaz de Usuario**: modifica una interfaz de usuario a través del ocultamiento, sustitución o adición de sus componentes. Esta modificación puede referenciar a otro dispositivo de hardware que provea de la misma funcionalidad del componente modificado. Este escenario realiza la adaptación de los iconos (e.g., en la *herramienta de votación*, en el *editor de mapas mentales*, en el *editor de dibujos* y en la *herramienta de medios de contactos y disponibilidad*) y las aplicaciones (e.g., *escritorio enriquecido*) de acuerdo al rol de los colaboradores.

3. **Redistribución de una aplicación colaborativa en función de la configuración del grupo**: adapta los componentes afectados por el trabajo colocalizado y redistribuido, entre ellos: el espacio público y compartido, la conciencia de grupo y el empleo de dispositivos que dispongan de pantallas grandes y una buena resolución (e.g., un pizarrón interactivo). Este escenario se asocia con las herramientas: la *barra de colaboradores*, la *herramienta de votación*, y la *herramienta de dibujo*.

4. **Mejoramiento de la conciencia grupal**: modifica los componentes que se relacionan con la disponibilidad de los colaboradores y sus respectivos medios de contacto, para ello considera varios parámetros de los colaboradores. Este escenario se asocia con la herramienta *medios de contacto y disponibilidad*.

5. **Manipulación de la información**: adapta la información en función del dispositivo en el cual se despliegue (e.g., un pizarrón interactivo o una *tablet*), cuando detecta una persona ajena al equipo de trabajo. Al estar cerca del área de despliegue una persona no autorizada, la información restringida se cambia por otra de menor relevancia y, en este escenario, se notifica al resto de los colaboradores de la presencia ajena para que eviten enviar mensajes u otro tipo de información importante en dicho momento. Este escenario se asocia con la herramienta *editor de textos*.

6. **Adaptación de la información**: proporciona información que puede ser de utilidad para los colaboradores, tomando en cuenta el lugar, la fecha y el rol de los colaboradores. Este escenario se asocia con la herramienta *administrador de contenidos vía NFC*.

7. **Enriquecimiento con notificaciones grupales**: provee de una variedad de posibilidades para notificar a los miembros del equipo de trabajo de un evento específico, con base en la actividad actual de cada uno de ellos, junto a la importancia de tal evento. Hasta el momento, este escenario no tiene asociada una aplicación en específico.

Capítulo 4

MODELADO SEMÁNTICO DEL ESPACIO COLABORATIVO GLOBAL

Ese capítulo define brevemente el contexto en ambientes colaborativos en la Sección 4.1. A continuación describe cómo llevar a cabo a composición de situaciones 4.2. Finalmente, la Sección 4.3 muestra el modelado semántico del espacio colaborativo global, i.e., sus clases y sus propiedades de objeto, cuya usabilidad se valida a través de la representación de los elementos de los escenarios descritos en el marco de desarrollo XARE para sistemas conscientes de contexto (véase Sección 3.3).

4.1. Contexto en ambientes colaborativos

Un sistema mono-usuario se forma por el usuario, la aplicación y demás entidades relevantes entre el usuario y la aplicación (e.g., dispositivos, proyectos, tareas, y lugares). Así, todas las situaciones sólo consideran el contexto de un sólo usuario y las entidades con las cuales éste interactúa. A diferencia, en un ambiente colaborativo participan varios colaboradores junto a

las entidades con que interactúan y, dado que cada colaborador tiene diferentes características (e.g., rol, tareas asignadas, equipos a los que pertenece) cada entidad puede ser afectada por diferentes situaciones.

Así, el *contexto en ambientes colaborativos* es la composición del contexto individual de cada una de las entidades que forman parte del ambiente colaborativo.

4.2. Composición de situaciones

La composición de situaciones consiste en la activación de metasituaciones, es decir, situaciones cuyas condiciones incluyan, además, la activación y/o desactivación de otras situaciones permanentes o la activación de una situación transitoria. Tales metasituaciones aplicarán operaciones del álgebra de conjuntos para resolver los casos en que una nueva situación entre en conflicto con las existentes.

La operación más simple, es la **unión** de la situaciones, por ejemplo supóngase un equipo de desarrollo web trabaja de forma no co-localizada a través de un ambiente de desarrollo remoto que permite que cada integrante tenga acceso a sus recursos (e.g., hojas de estilo, imágenes, y scripts) y al de cada uno de los miembros restantes. Cuando un integrante nuevo ingresa a la sesión de trabajo, es decir se activa la situación permanente *El usuario i ha iniciado sesión* y el conjunto de recursos mostrados es la unión del conjunto previo de recursos mostrados con los recursos del nuevo integrante.

Otra operación es la **intersección** de situaciones, en un editor de textos colaborativo varios usuarios inician sesión, entonces el editor muestra una lista de todos los documentos cuyos autores sean los usuarios en línea. Cuando un nuevo usuario inicia sesión, se activa la situación permanente *El usuario i ha iniciado sesión* y la composición de situaciones corresponde a la intersección de los documentos cuyos autores sean los usuarios en línea y el nuevo usuario.

La **diferencia** de situaciones puede utilizarse en una aplicación colaborativa para realizar

votaciones, que muestre una lista de los colaboradores que faltan de emitir su voto. Cuando un colaborador emite su voto, se realiza la diferencia del conjunto de los usuarios que están registrados en una determinada situación con el conjunto de los usuarios que han emitido su voto.

Una operación más, el **complemento** de situaciones puede ser aplicado en un aplicación colaborativa que muestre una lista de las tareas de un proyecto y coloque al inicio aquellas tareas incompletas. El cambio de estado de una situación permanente del tipo *The task i is finished* implicaría realizar obtener el complemento de las tareas completas en el conjunto de las situaciones que denoten todas las tareas.

4.3. Modelo semántico del espacio colaborativo global

Un espacio de colaboración global consiste en todos los usuarios y sus respectivos espacios de trabajo, las aplicaciones, los servicios y los recursos necesarios para realizar un proyecto entre los usuarios involucrados [Haake et al., 2010].

El espacio global de colaboración está compuesto por el espacio de trabajo de cada uno de los usuarios, compuesto por su propia configuración y su conjunto de aplicaciones. A su vez, cada aplicación tiene un conjunto asociado de acciones soportadas y servicios; las acciones describen la interacción entre los usuarios y las aplicaciones, y los servicios, entre la aplicación y los recursos, tanto para manipular información o realizar operaciones de coordinación. Un recurso es utiliza colaborativamente tan pronto sea accedido por más de un usuario.

No obstante, cada usuario tiene asignado al menos un rol, que representa las acciones que éste puede ejecutar en una aplicación, pues es modelado explícitamente para ésta y es asignado por el espacio de trabajo.

El dominio de la aplicación así como diferentes aspectos del contexto (e.g., los ambientes físico y computacional, los recursos y el modelo de usuario) pueden representarse como clases de individuos; y su interacción, mediante relaciones.

Las construcciones del *Ontology Web Language* (OWL) son adecuadas para su representación porque:

1. Trabajan con conceptos en lugar de palabras clave, que simplifican la unificación de las distintas representaciones de un mismo concepto.

2. El modelo creado en una ontología puede modificarse sin alterar las instancias que ya se tienen.

3. El objetivo de las ontologías es generar conocimiento. En los sistemas basados en modelos se obtiene información directamente de las relaciones evidentes entre los conceptos, y se genera conocimiento al inferir aquellas que no son evidentes.

4. Se integrará y extenderá las ontologías de la arquitectura RAMS, que describe el dominio de la compartición de recursos, y **sólo** los roles de los trabajadores de una institución.

En las aplicaciones colaborativas que implementan el paradigma Modelo-Vista-Controlador (MVC) cada vez que los usuarios interactúan con la aplicación mediante las acciones permitidas por sus roles, éstas son recibidas por el controlador correspondiente. Entonces, a través de los servicios el controlador modifica el recurso; y el modelo, los componentes de la vista [Haake et al., 2010].

Las clases sugeridas por [Haake et al., 2010] para modelar sermánticamente el espacio colaborativo se han adaptado a fin de integrarlas con las clases de la arquitectura RAMS y se muestran en la Figura 4.1.

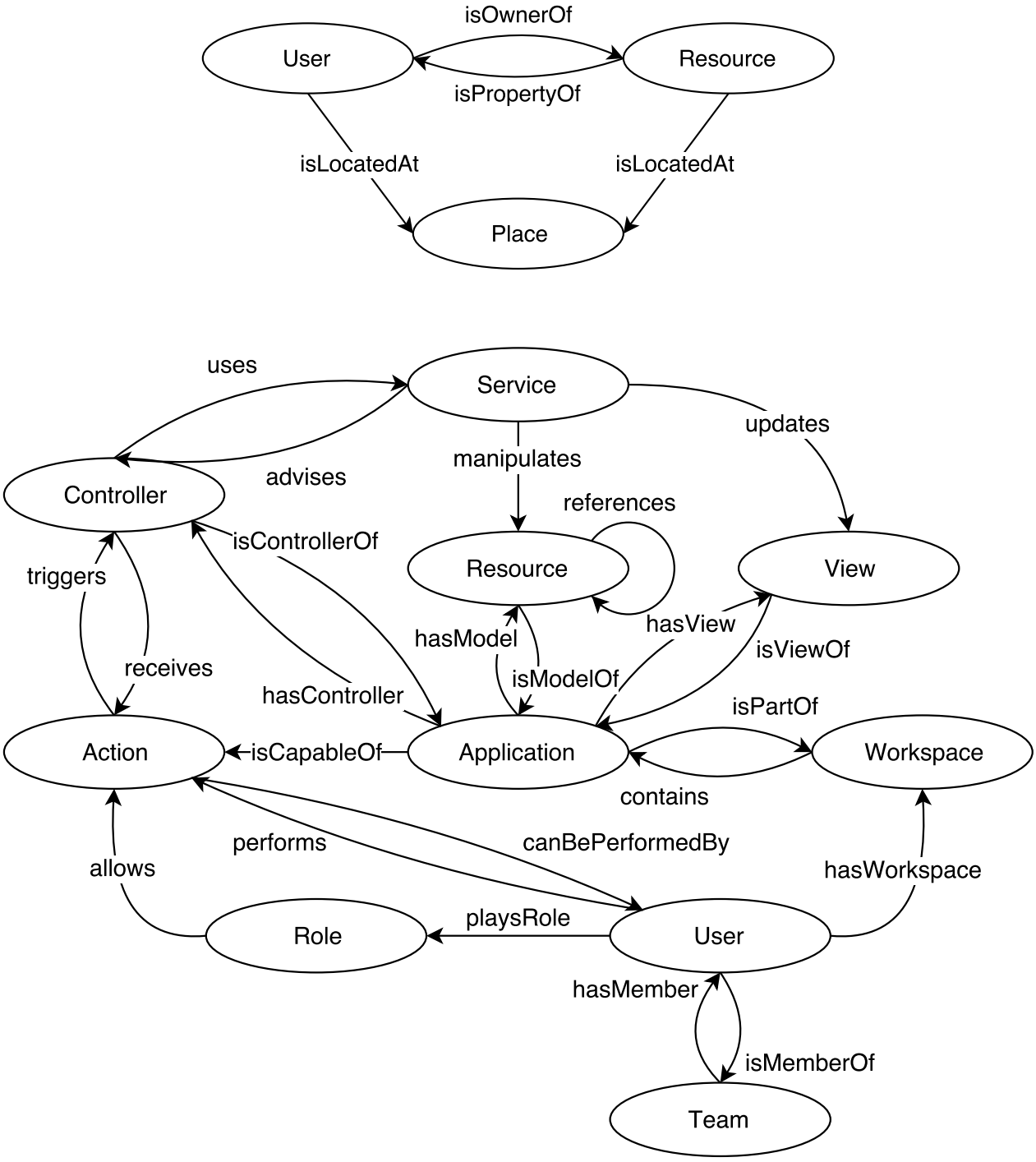


Figura 4.1: Modelo semántico para un sistema colaborativo

Para un mejor entendimiento de la Figura 4.1 se describen cada una de las clases representadas y sus propiedades de objeto en las secciones 4.3.1 y 4.3.2, respectivamente.

4.3.1. Clases

- ***User***: corresponde al prototipo humano que utilizará el sistema colaborativo.
- ***Team***: corresponde a los prototipos humanos que utilizarán el sistema colaborativo y están involucrados en un proyecto común, i.e. instancias de la clase `User`.
- ***Workspace***: es el espacio de trabajo de un determinado usuario, define un conjunto de aplicaciones disponibles, i.e. instancias de la clase `Application`, y sus respectivos roles, i.e. instancias de la clase `Role`.
- ***Application***: es un programa implementado bajo el paradigma Modelo - Vista - Controlador (MVC), por consiguiente la clase `Resource` representa el modelo; la clase `Controller`, el controlador; y la clase `View`, la vista.
- ***View***: corresponde a la vista de una aplicación, i.e. una instancia de la clase `Application`.
- ***Controller***, corresponde al controlador de una aplicación, i.e. una instancia de la clase `Application`, y manipula el modelo, i.e. instancias de la clase `Resource`, a través de servicios.
- ***Action***: representa una interacción entre una aplicación, i.e. una instancia de la clase `Application`, y un usuario, i.e. instancias de la clase *User*.
- ***Service***: representa una operación realizada por una aplicación, i.e. una instancia de la clase `Application`, sobre un recurso, i.e. un instancia de la clase `Resource`.
- ***Role***: define un conjunto de las posibles acciones, i.e. instancias de la clase `Action`, que un actor puede ejecutar en una aplicación específica, i.e. en una instancia de la clase `Application`.
- ***Resource***: representa un recurso de información (e.g., una base de datos, ontología, documento, imagen).

4.3.2. Propiedades de objeto

Relacionadas con el usuario

- ***playsRole***: relaciona un individuo de la clase `User` con al menos un individuo de la clase `Role`, a fin de especificar el o los roles que un determinado usuario tiene.
- ***hasWorkspace***: relaciona un individuo de la clase `User` con un individuo de la clase `Workspace`, puesto que cada uno de los usuarios cuenta con su propio espacio de trabajo.
- ***isMemberOf***: relaciona un individuo de la clase `User` con un individuo de la clase `Team`, puesto que un usuario pertenece a un equipo de trabajo.
- ***performs***: relaciona un individuo de la clase `User` con los individuos de la clase `Action`, para especificar las acciones que un determinado usuario es capaz de realizar en la aplicación, i.e., una instancia de la clase `Application`.

Relacionadas con la aplicación

- ***isPartOf***: relaciona un individuo de la clase `Application` con un individuo de la clase `UserWorkspace`, puesto que una aplicación pertenece a uno o más espacios de trabajo.
- ***hasModel***: relaciona un individuo de la clase `Application`
- ***hasController***: relaciona un individuo de la clase `Application` con un individuo de la clase *Controller*, puesto que el segundo es el controlador de una determinada aplicación desarrollada bajo el paradigma MVC.
- ***hasView***: relaciona un individuo de la clase `Application` con un individuo de la clase *Controller*, puesto que el segundo es la vista de una determinada aplicación desarrollada bajo el paradigma MVC.

Relacionadas con el modelo de la aplicación

- ***isCapableOf***: relaciona un individuo de la clase `Functionality` con un individuo de la clase `Action`, a fin de especificar las acciones que una aplicación soporta.
- ***triggers***: relaciona un individuo de la clase `Action` con un individuo de la clase `Controller`, puesto que la ejecución de una acción por parte del usuario es notificada a el controlador.

Relacionadas con la vista de la aplicación

- ***usesView***: relaciona un individuo de la clase `View` con un individuo de la clase `Service`, para actualizar la interfaz de usuario de una determinada aplicación.

Relacionadas con el controlador de la aplicación

- ***receives***: relaciona un individuo de la clase `Controller` con un individuo de la clase `Action`, puesto que la ejecución de una acción por parte del usuario es notificada a el controlador.
- ***usesService***: relaciona un individuo de la clase `Controller` con un individuo de la clase `Service`, puesto que el controlador manipula un resurso de información mediante un servicio.
- ***accesses***: relaciona un individuo de la clase `Service` con un individuo de la clase `Resource`, puesto que los servicios son los encargados de operar un recurso.
- ***notifies***: relaciona un individuo de la clase `Service` con un individuo de la clase `View`, para actualizar la inferfaz de usuario de una determinada aplicación.
- ***uses***: relaciona un individuo de la clase `Resource` con un individuo de la clase `Service`, a fin de representar las operaciones de un recurso por parte de un servicio.

- ***references***: relaciona dos individuos de la clase `Resource`, puesto que un recurso de información puede contener a otro(s), o bien, ser parte de otro.

4.4. Validación del modelo semántico

Sánchez identificó siete escenarios de trabajo colaborativo en los cuales la adaptación al contexto mejora significativamente la labor de los colaboradores. Con el propósito de extender el modelo semántico del espacio colaborativo global, se identificaron las entidades y sus respectivas variables contextuales de las aplicaciones y marcos expuestos en el estado del arte, con el objetivo. La Tabla 2.1 presenta las entidades y sus respectivas variables contextuales para las aplicaciones y los marcos mono-usuario.

Para validar el modelo semántico, a partir de éste se modelaron las aplicaciones propuestas en los siete escenarios identificados en el marco XARE. En las siguientes secciones se presentan las clases pertinentes.

Las descripciones de los escenarios fueron extraídas de [Sánchez Morales, 2013].

Dado que la *herramienta de administración de contenidos vía NFC* es el prototipo que valida esta tesis éste se explicará en el Capítulo 7.

4.4.1. Herramienta 1: Herramienta de votación

Descripción del escenario

> Supongamos que Coello, el jefe del Departamento de Computación de un universidad, ha organizado una reunión en la sala de juntas que cuenta con un pizarrón interactivo. El objetivo de la reunión es elegir al nuevo coordinador académico. Al llegar a la sala de juntas, cada participante es autenticado en el sistema ya sea: 1) de manera proactiva a través de *NFC* (*Near Field Communication*) o 2) por medio de una ventana que le solicita su nombre de usuario y contraseña. Al inicio

de la reunión, sólo algunos de los participantes están físicamente presentes. La información de consciencia sobre la presencia de los miembros no sólo puede ser inferida de manera natural, sino que también es mostrada en el pizarrón interactivo por medio de una barra que presenta a los miembros ausentes y virtualmente presentes.

Coello inicia la *herramienta de votación*, la cual le asigna el rol de *proponente*. Por medio del pizarrón interactivo, Coello puede plantar a todos los profesores presentes la pregunta principal por la cual se organizó la reunión, las opciones de respuesta y las reglas de la sesión de votación.

Coello escribe la pregunta en el pizarrón interactivo y establece que la votación puede ser anónima, i.e. los votantes pueden elegir si hacen público o no su nombre de usuario al votar. Coello además determina que la votación será válida siempre y cuando haya votado al menos el 90 % del total de los profesores del departamento. Una vez que Coello ha confirmado la información sobre la votación en el pizarrón interactivo, cada dispositivo de los miembros física y virtualmente presentes recibe una notificación de votación. Cuando un miembro selecciona dicha notificación, la *herramienta de votación* se inicia automáticamente en su dispositivo móvil y le asigna el rol de *votante*.

La aplicación da un tiempo de espera para que los profesores analicen las opciones disponibles y emitan su voto, siendo que la aplicación proporciona una opción de abstención y un botón para emitir un voto anónimo. Al finalizar el tiempo establecido, la aplicación detecta que el porcentaje mínimo para validar la votación no se ha alcanzado. Por lo tanto, la aplicación muestra en el pizarrón interactivo un mensaje de aviso que proporciona las siguientes opciones: 1) validar la votación con el porcentaje actual, 2) posponer la votación (con el fin de alcanzar el porcentaje requerido en otro momento) y 3) cancelar la votación. Coello selecciona la primera opción. Como resultado, la aplicación muestra los resultados en

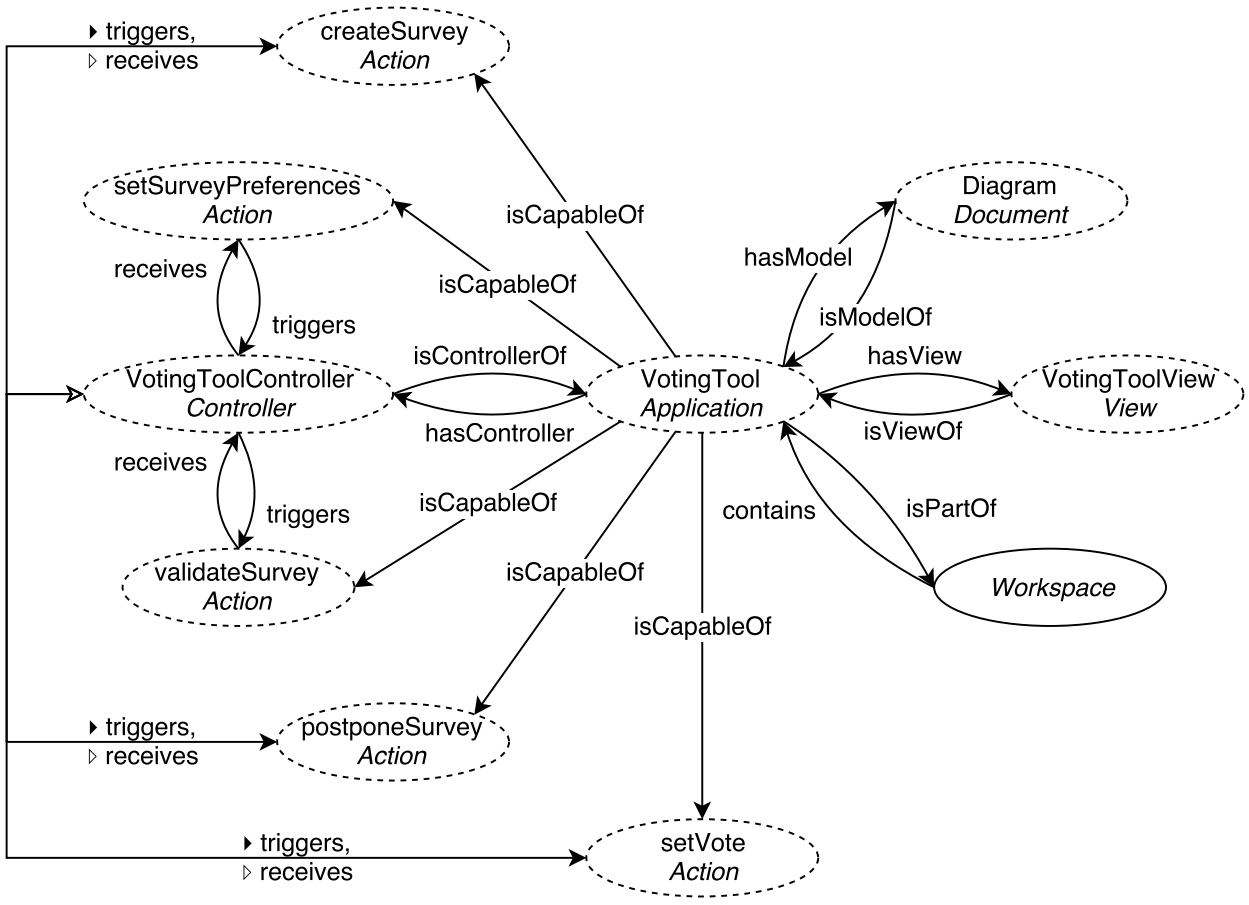
el pizarrón interactivo y los envía a los dispositivos móviles de los virtualmente presentes.

La Tabla 4.1 muestra las entidades y sus respectivas variables contextuales de la *herramienta de votación*, que está asociada a los escenarios *remodelación de la interfaz de usuario* y *redistribución de una aplicación colaborativa*.

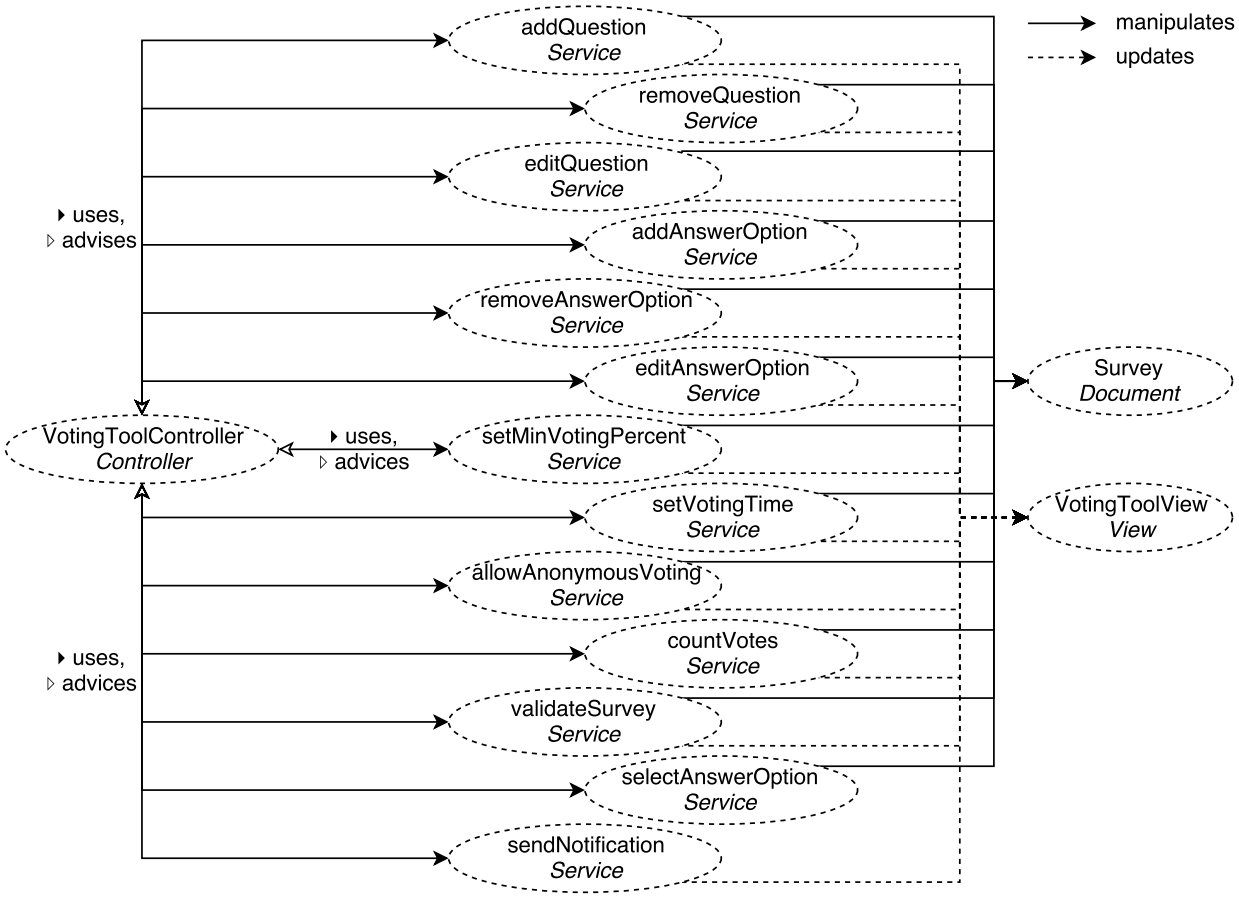
Cuadro 4.1: Entidades y sus variables contextuales de la *herramienta de votación*

Entidad	**Variables contextuales**
Equipo	Configuración (co-localizado, distribuido e híbrida)
Dispositivo	Tamaño de la pantalla
Usuario	Dispositivo en uso, rol (*proponente* y *votante*), ubicación física (a través de la herramienta *barra de colaboradores*)
Votación	Porcentaje mínimo de votación, tiempo de espera

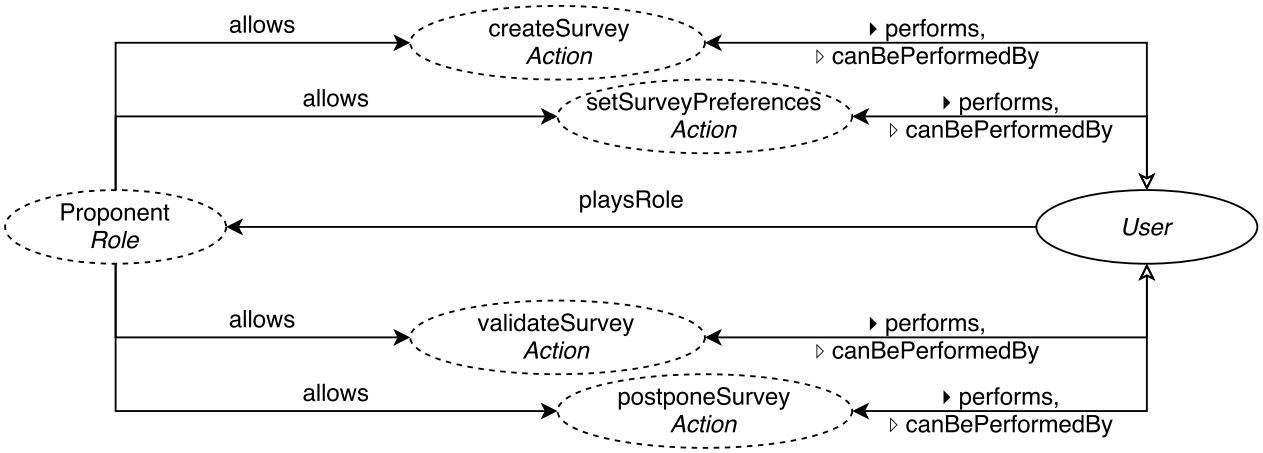
Las figuras siguientes muestran las clases, sus instancias y sus respectivas propiedades de objeto que modelan el escenario previamente expuesto.



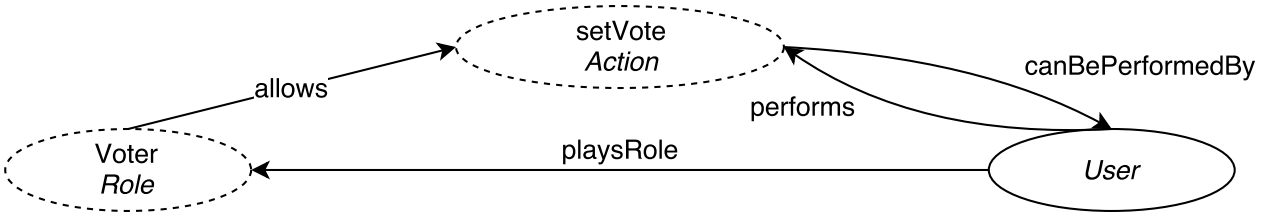
(a) Propiedades de la aplicación `VotingTool`



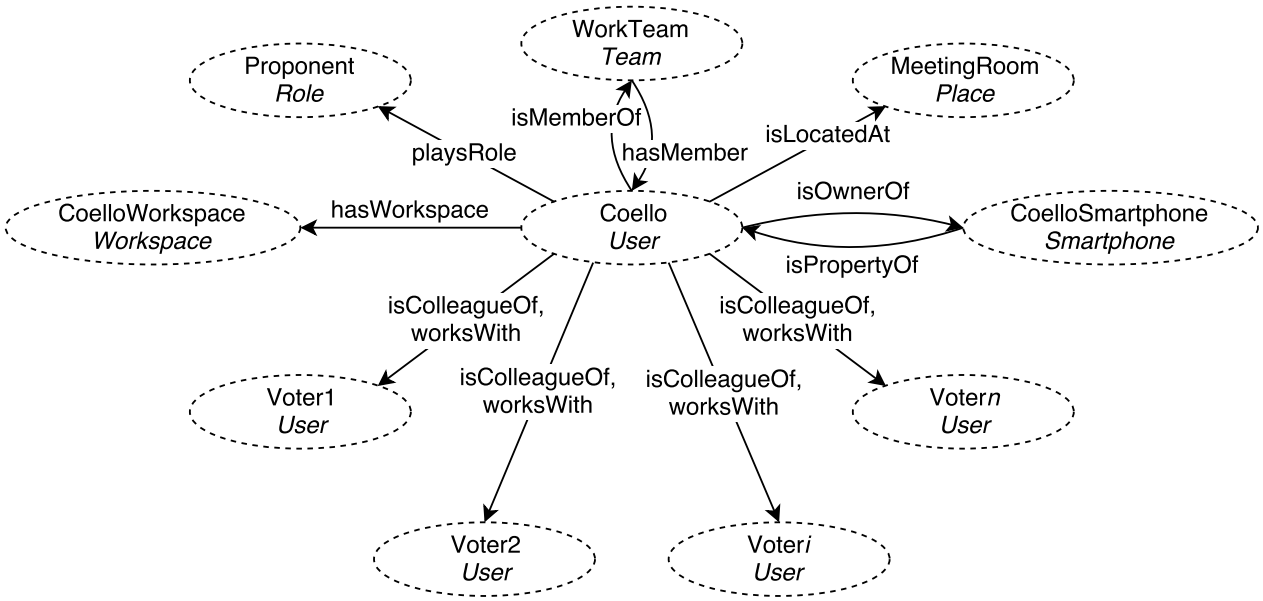
(b) Servicios de la aplicación `VotingTool`



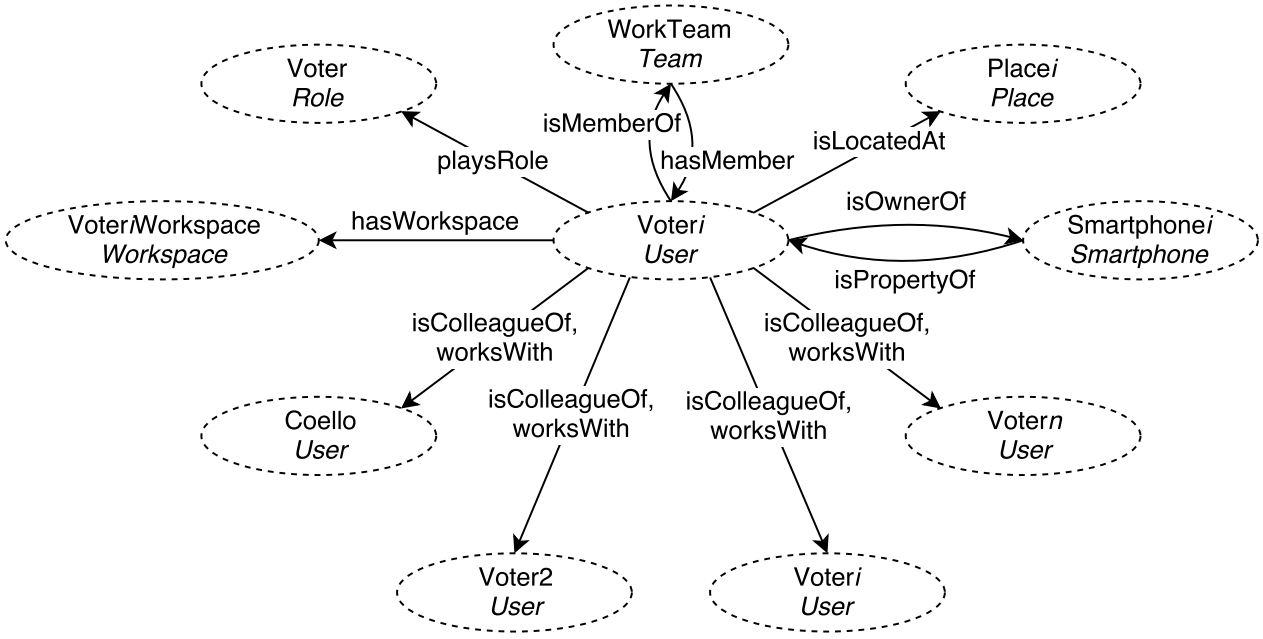
(c) Propiedades del rol `Proponent`



(d) Propiedes del rol `Voter`



(e) Propiedades del usuario `Coello`



(f) Propiedades del usuario $\texttt{Voter}_1$

Figura 4.2: Modelado semántico de la aplicación `VotingTool`

4.4.2. Herramienta 2: Editor de mapas mentales

Descripción del escenario

Un grupo de colaboradores compuesto por Coello, Mejia, Fraga, Chakraborty y Morales, necesita realizar un mapa mental sobre software libre. Dichos colaboradores han planeado formalmente una cita que ha sido registrada en la agenda de la sala de reuniones.

Coello y Fraga asisten a la reunión llevando consigo cada uno un teléfono inteligente, mientras que Morales acude con su *tablet PC*. Cuando Coello, Fraga y Morales entran a la sala de reuniones, son autenticados automáticamente por medio de una etiqueta *NFC* que se encuentra en la entrada de la sala. Mientras tanto, Mejia y Chakraborty entran al sistema manualmente introduciendo su nombre de usuario y contraseña mediante su *tablet PC*, ya que ellos han decidido interactura con sus colegas de manera remota.

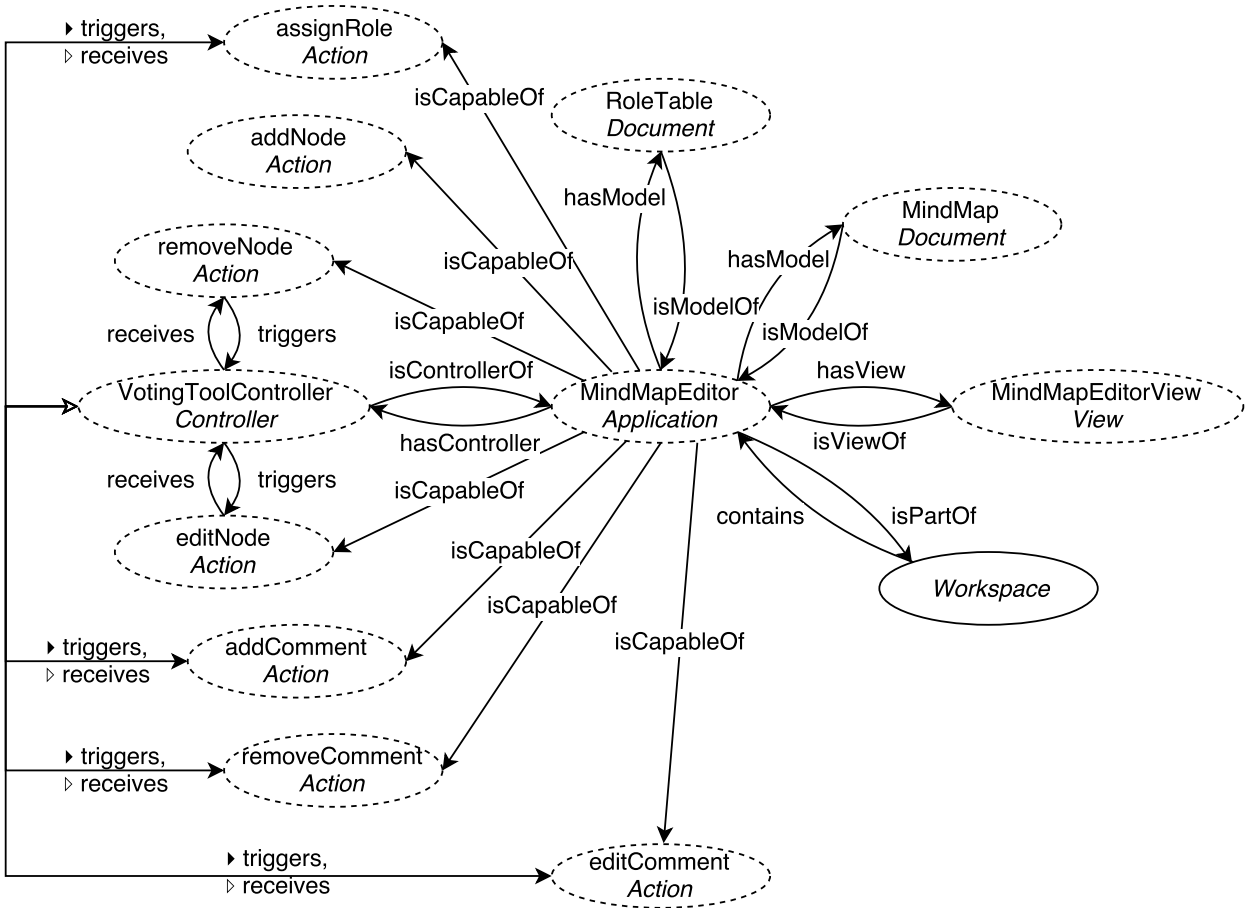
Cuando Coello se aproxima al pizarrón interactivo de la sala de juntas y toca la etiqueta *NFC* asociada a dicho pizarrón, se muestra la pantalla principal que contiene las aplicaciones disponibles y un mensaje que indica que su sesión ha iniciado. Además, se visualiza una barra de colaboradores donde se muestran los colaboradores virtualmente presentes, en este caso, el nombre y la foto de Mejia y Chakraborty. Desde la pantalla principal, Coello ejecuta el editor colaborativo de mapas mentales, el cual le asigna el rol de *administrador* que le permitirá seleccionar a los coautores y asignarles sus respectivos roles. Coello decide dar el rol de *revisor* a Mejia y Chakraborty, en tanto el rol de *autor* es asignado a Coello, Fraga y Morales. En particular, el rol de *autor* permite la adición, o eliminación y modificación de los elementos del mapa mental, mientras que el rol de *revisor* permite la adicción, modificación y eliminación de los comentarios asociados a dichos elementos. Una vez que Coello concluyó la asignación de roles

a los coautores, el editor de mapas mentales se ejecuta automáticamente en los dispositivos móviles de sus colegas [Sánchez Morales, 2013].

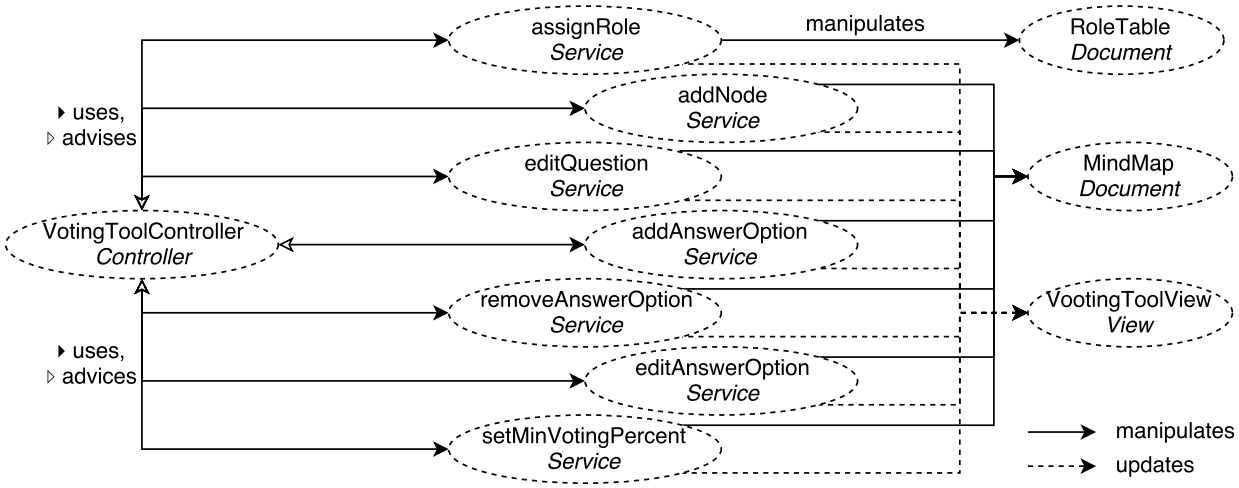
La Tabla 4.2 muestra las entidades y sus respectivas variables contextuales de la herramienta *editor de mapas mentales*, que está asociada a los escenarios *remodelación de la interfaz de usuario* y *redistribución de una aplicación colaborativa*.

Cuadro 4.2: Entidades y sus variables contextuales de la herramienta *editor de mapas mentales*

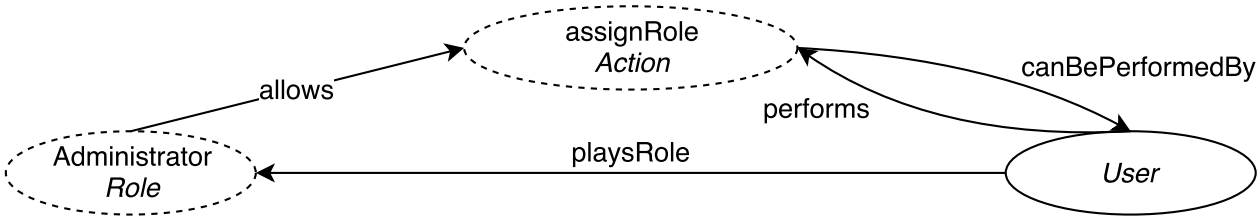
Entidad	**Variables contextuales**
Equipo	Configuración (co-localizado, distribuido e híbrida)
Dispositivo	Tamaño de la pantalla (*pantalla pequeña*, cuando la pantalla del dispositivo tiene un tamaño menor a siete pulgadas y *pantalla grande*, en cualquier otro caso), Espacio libre en disco duro
Mapa mental	Dimensiones
Usuario	Dispositivo en uso, rol (administrador, autor y revisor), ubicación física (a través de la herramienta *barra de colaboradores*)



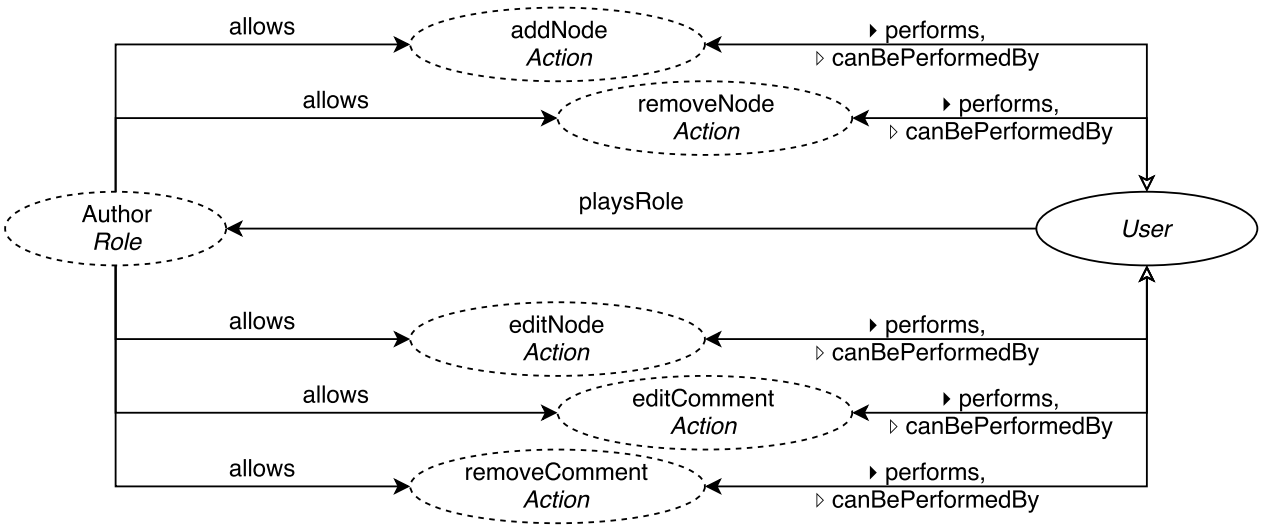
(a) Propiedades de la aplicación `MindMapEditor`



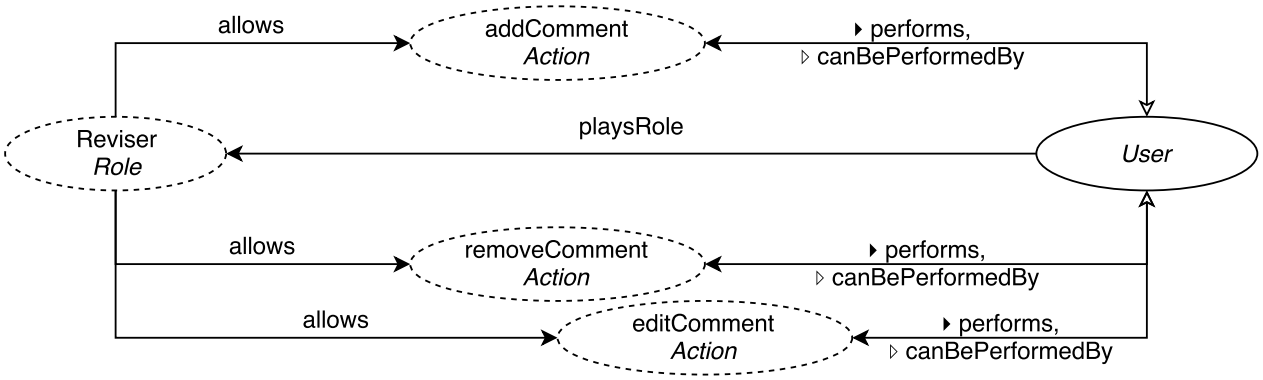
(b) Servicios de la aplicación `MindMapEditor`



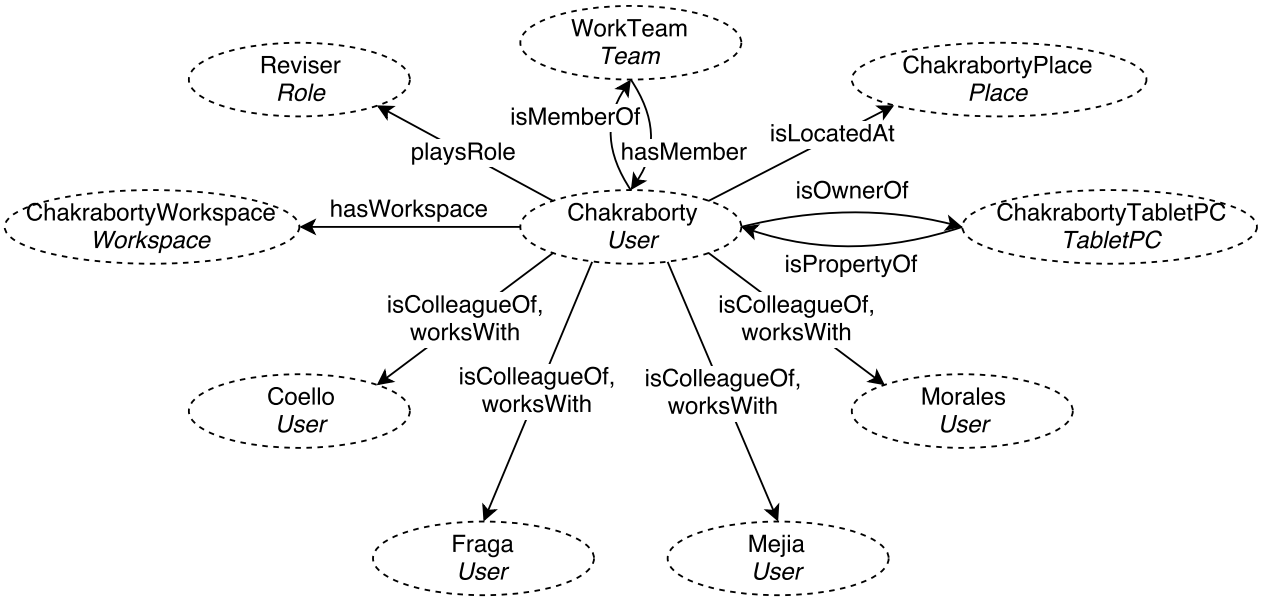
(c) Propiedades del rol `Administrator`



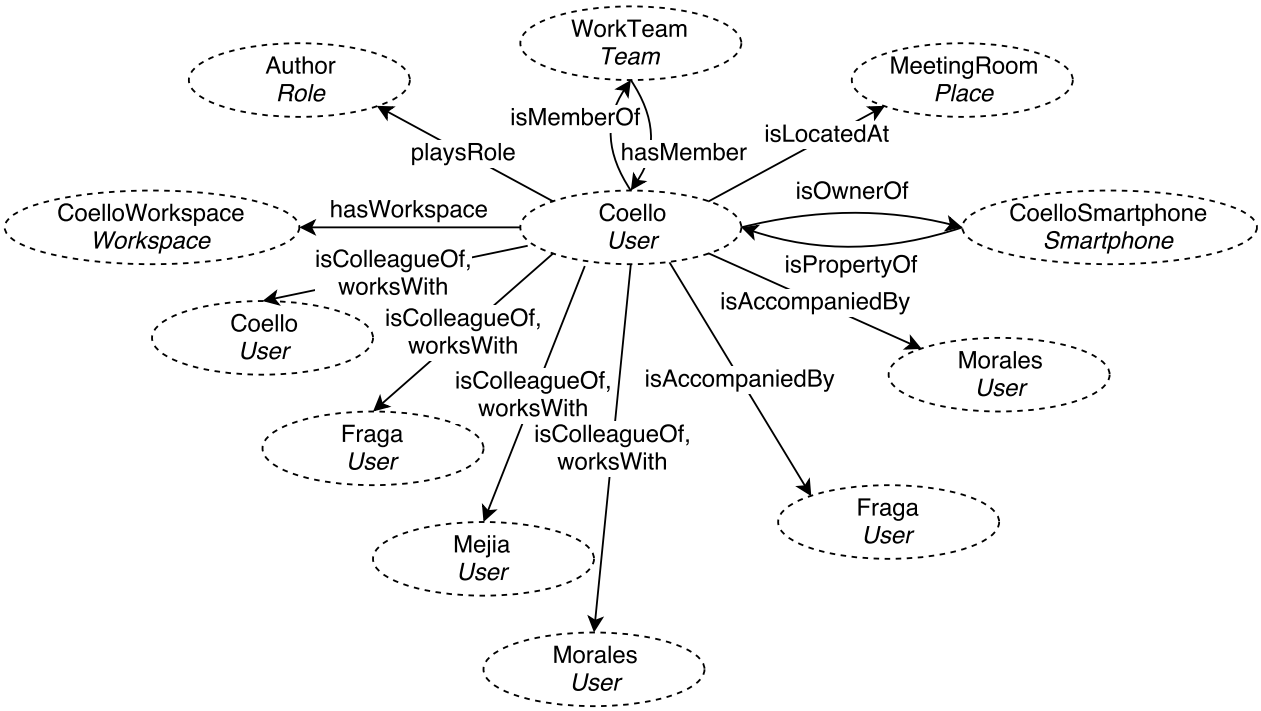
(d) Propiedades del rol `Author`



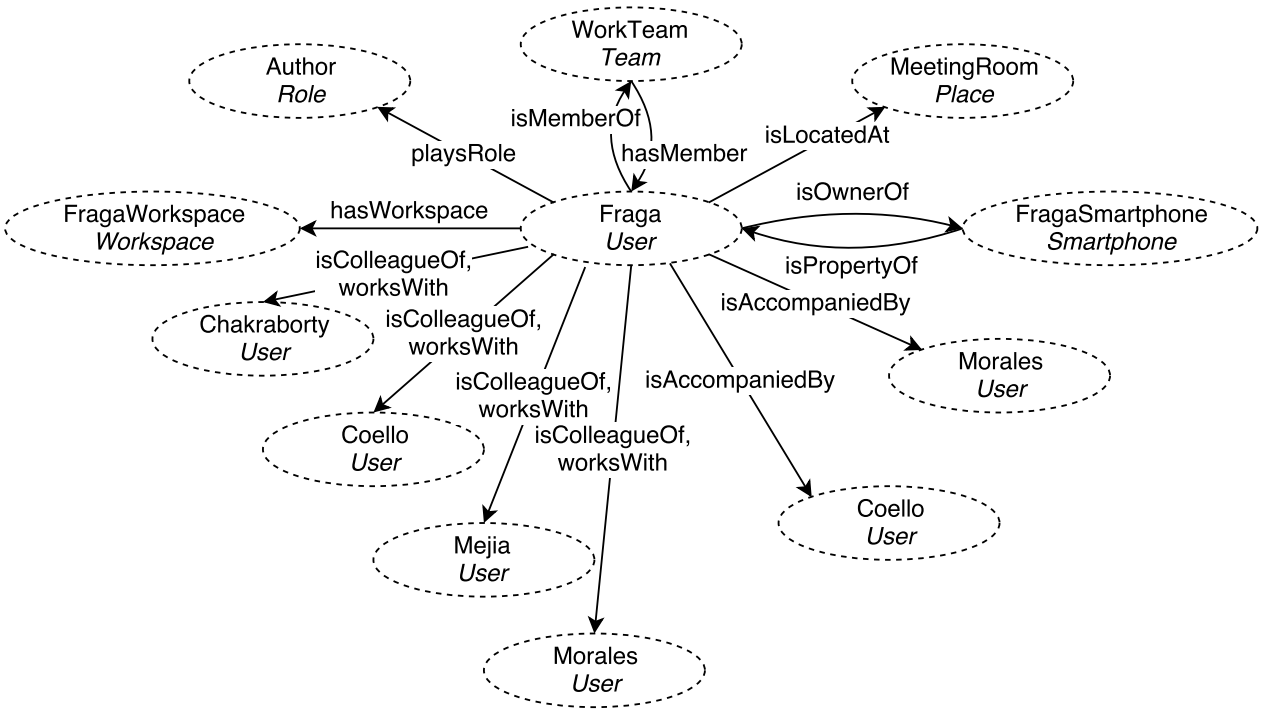
(e) Propiedes del rol `Reviser`



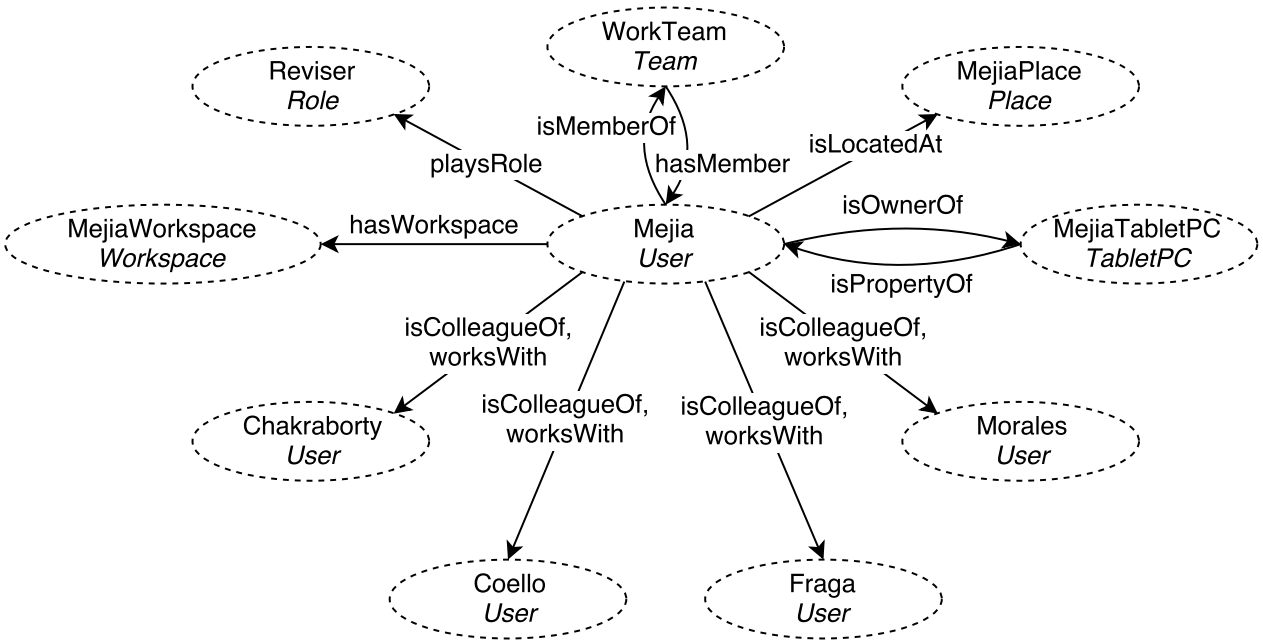
(f) Propiedades del usuario `Chakraborty`



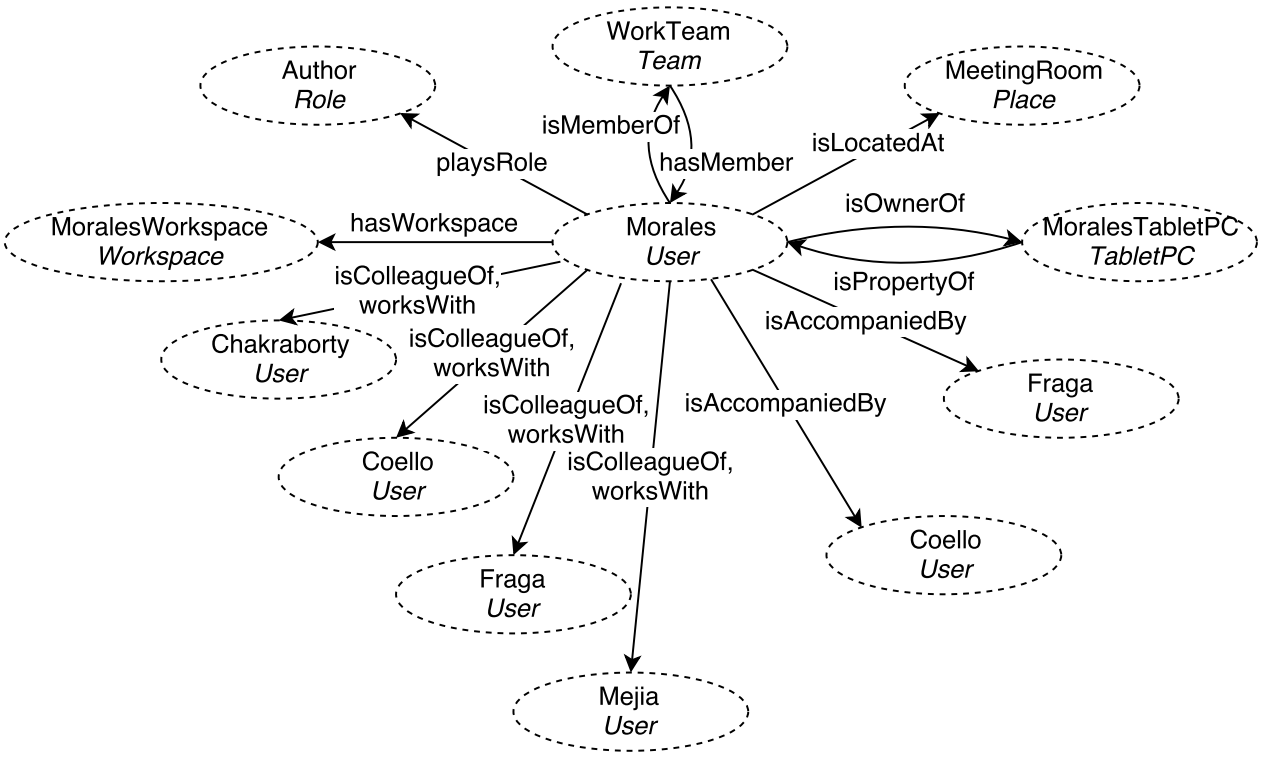
(g) Propiedades del usuario `Coello`



(h) Propiedades del usuario `Fraga`



(i) Propiedades del usuario `Mejia`



(j) Propiedades del usuario `Morales`

Figura 4.3: Modelado semántico de la herramienta *Editor de mapas mentales*

4.4.3. Herramienta 3: Herramienta de medios de contacto y disponibilidad

Descripción del escenario

Un grupo de colaboradores, cuyos nombres corresponden a Isaac, Alice y Jenny, se encuentra trabajando a distancia en la redacción de un reporte técnico para un nuevo producto de software. En particular, Isaac es responsable de escribir la sección de desarrollo y, Alice y Sandy están respectivamente encargadas de escribir y revisar la sección de conclusiones.

Isaac ingresa al espacio de trabajo, para terminar algunas subsecciones de la sección de desarrollo, ya que la fecha límite de esta actividad está relativamente cercana. Así, basado en las preferencias de Isaac, el espacio de trabajo determina su disponibilidad como sigue:

1. **Ocupado** para los colegas: a) cuya actividad actual es *muy poco* o *no similar* a la suya, aún si una de sus potenciales actividades es *más o menos* o *poco similar* a su actividad actual, y b) cuya actividad actual o potencial son *muy poco* o *no similares* a su actual actividad,

2. **Alcanzable si es posible** para los colegas: a) cuya actividad actual es *más o menos* o *poco similar* a la suya, y b) cuya actividad actual es *muy poco* o *no es similar* a la suya, pero una de sus actividades potenciales es *similar* o *muy similar* a su actividad actual, y

3. **Accesible** para los colegas, cuya actividad actual es *similar* o *muy similar* a la suya.

Entonces, Isaac define sus propios medios de contacto basados en sus sub-estados de disponibilidad (el primer parámetro de entrada):

1. Cuando él está **ocupado**, él puede ser contactado por *correo electrónico*, así que temas no relacionados con su actividad actual serán tratados cuando él pueda/quiera,

2. En el caso de estar **accesible si es posible**, él podrá ser contactado sólo por *mensajería instantánea*, para ser informado inmediatamente de algún tema y retrasar su respuesta en caso de ser necesario, y

3. Cuando él esté **accesible**, él podrá ser contactado por *vídeo-conferencia*, *voz* o *mensajería instantánea* porque, desde su punto de vista, los medios de comunicación directa en tiempo real hacen entendibles las ideas y sugerencias relacionadas con su actividad actual.

Cuando Alice y Sandy ingresan al espacio compartido, ellas prefieren que éste interfiera su disponibilidad y sus medios de contacto. Tanto Alice como Sandy deciden trabajar en la sección de conclusiones, ellas colocan su cursor en dicha sección, para empezar a escribir y redactar respectivamente. Dado que la fecha límite de esa actividad es relativamente distante, el espacio de trabajo determina los siguientes sub-estados de su disponibilidad:

1. **Ocupadas** para colegas, cuya actividades actual y potenciales sean *poco* o *no similares* a sus actividades actuales,

2. **Alcanzables si es posible** para colegas, cuya actividad actual es *muy similar* o *no similar* a las suyas, pero una de sus potenciales actividades es *muy similar*, *similar*, *más o menos similar* o *poco similar* a su actividad actual, y

3. **Accesible** para colegas, cuya actividad actual es *muy similar*, *similar*, *más o menos similar* o *poco similar* a las suyas.

Además, el espacio de trabajo establece el medio de contacto de Sandy basado en

la similitud entre sus actividades y las actividades de sus colegas, considerando las preferencias de Sandy. De esta manera, ella puede ser contactada a través de:

1. *Anotaciones privadas* en el texto o *correo electrónico* por colegas, cuya actividad actual o potencial son *muy similares* o *similar* a su actual o potencial actividad. En este caso, Sandy ha seleccionado esos medios de contacto, ya que éstos aseguran la persistencia de datos, y

2. Sus medios de contacto preferidos, *mensajería instantánea* o *voz*, de otra manera.

El espacio de trabajo de Alice también determina sus medios de contacto basado en la proximidad lógicas entre ella y sus colegas dentro del espacio, Así, ella puede ser contactada a través de:

1. *Anotaciones privadas* en el texto por colegas que están trabajando en el mismo objeto compartido que está siendo accedido por ella.

2. *Vídeo-conferencia*, *voz* y *mensajería instantánea* de otra manera.

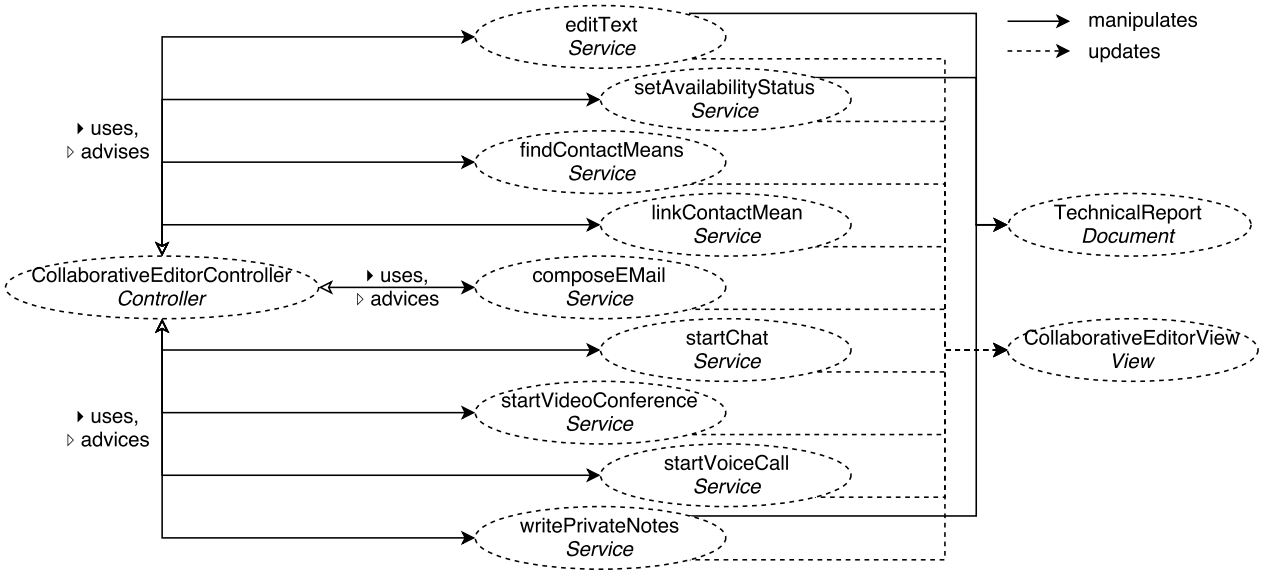
En todos los casos, la disponibilidad de los medios de contacto dependen del hardware y software de los dispositivos de Isaac, Sandy y Alice.

La Tabla 4.3 muestra las entidades y sus respectivas variables contextuales de la *herramienta de medios de contacto y disponibilidad*, que está asociada a los escenarios *remodelación de la interfaz de usuario* y *mejoramiento de la conciencia grupal*.

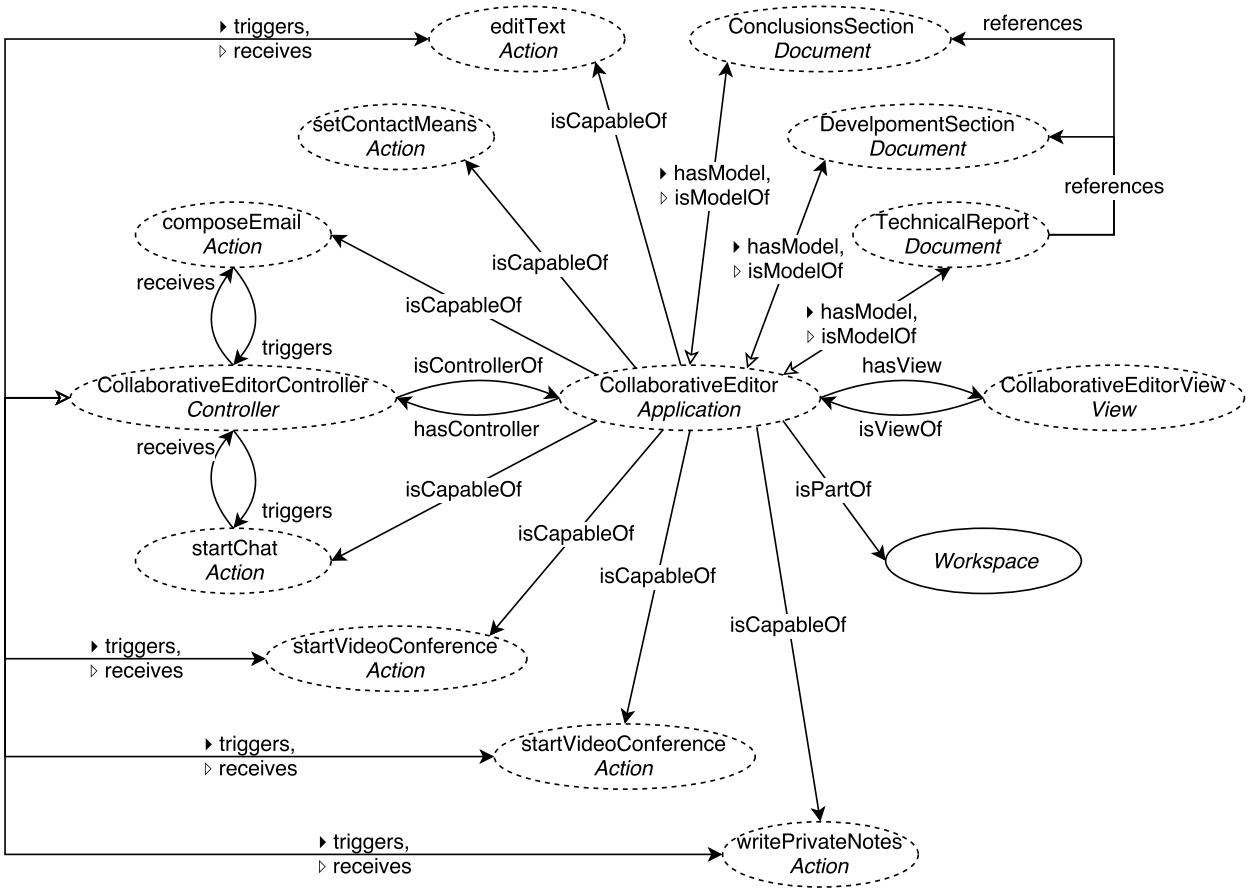
Esta herramienta comprende las dimensiones *grupo de colaboradores* y *conjunto de plataformas*.

Cuadro 4.3: Entidades y sus variables contextuales de la *herramienta de medios de contacto y disponibilidad*

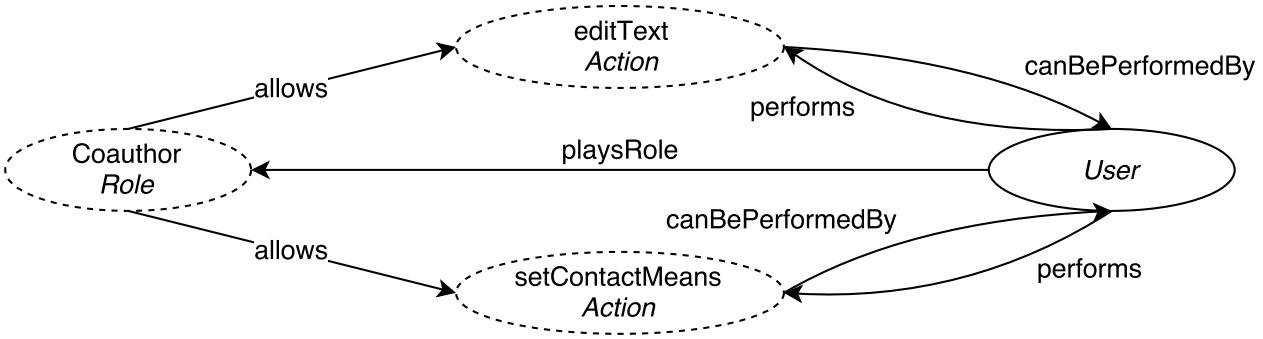
Entidad	Variables contextuales
Equipo	Cercanía entre colaboradores
Proyecto	Fecha límite, similitud entre actividades (*muy similar*, *similar*, *más o menos similar* y *poco similar*)
Usuario	Actividades en curso, actividades potenciales, dispositivo en uso, estado de disponibilidad (*ocupado*, *alcanzable si es posible* y *accesible*), preferencias (medios de contacto predilectos de acuerdo a su estado de disponibilidad), ubicación física
Dispositivo	Hardware disponible, Software disponible



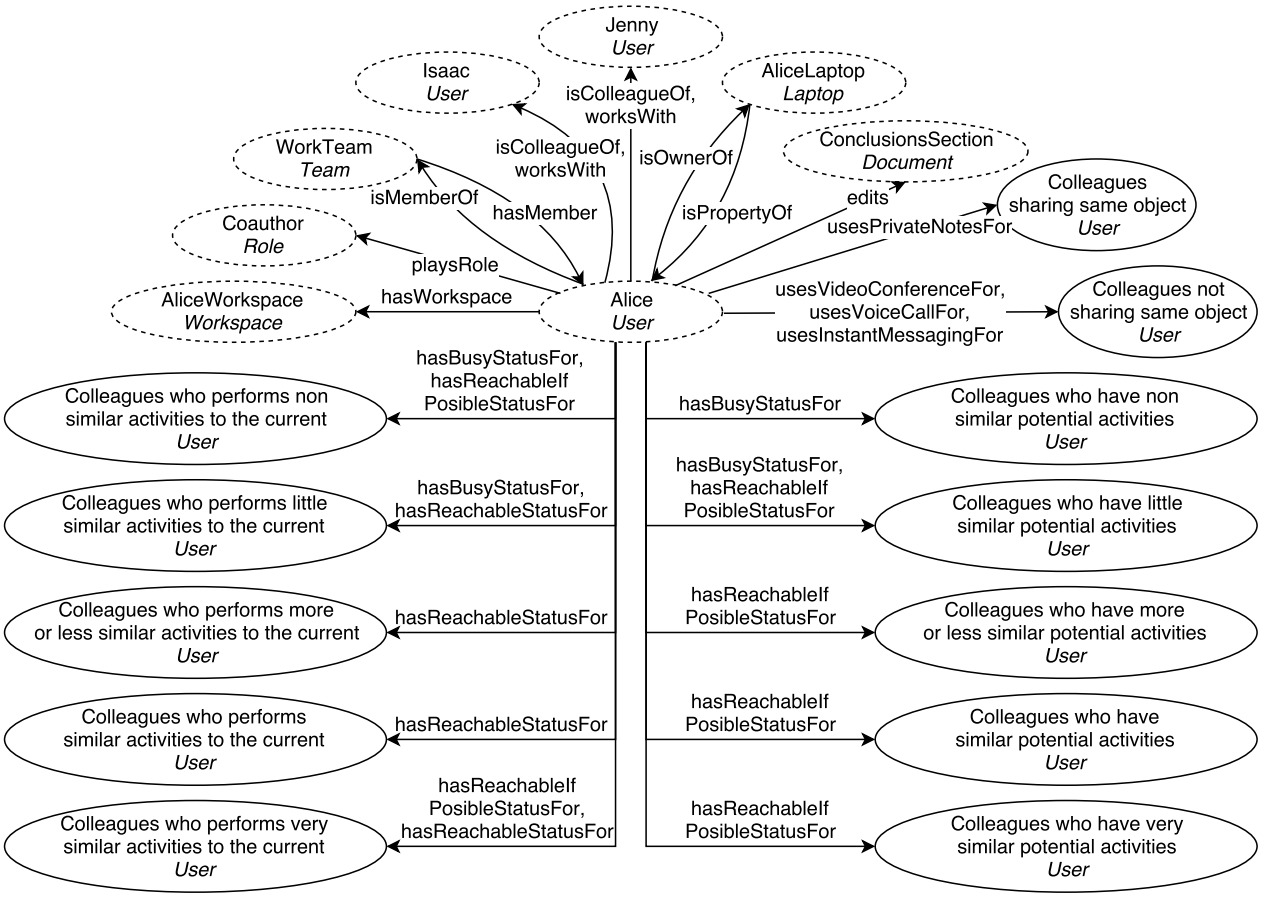
(a) Servicios de la aplicación `CollaborativeEditor`



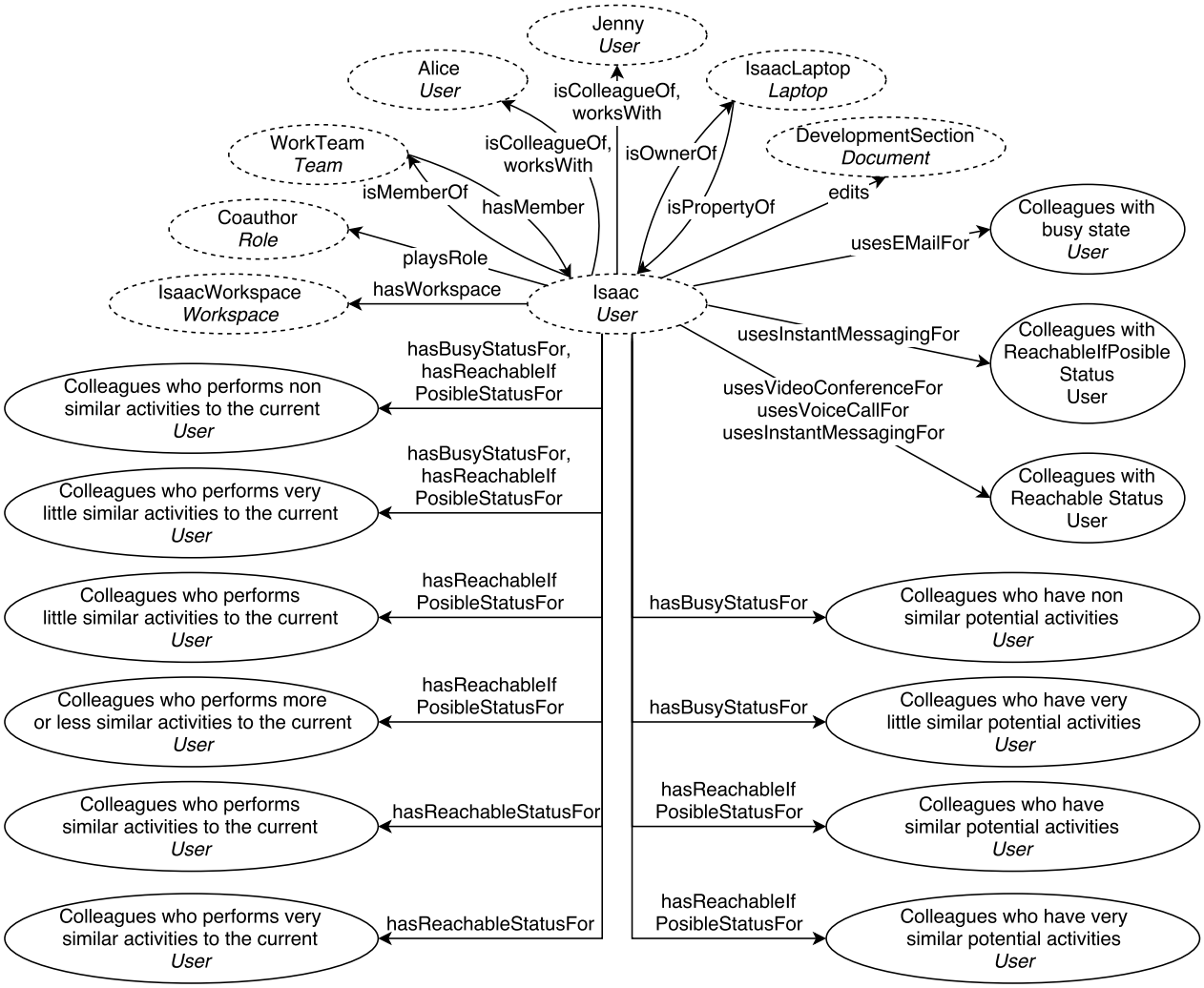
(b) Propiedades de la aplicación `CollaborativeEditor`



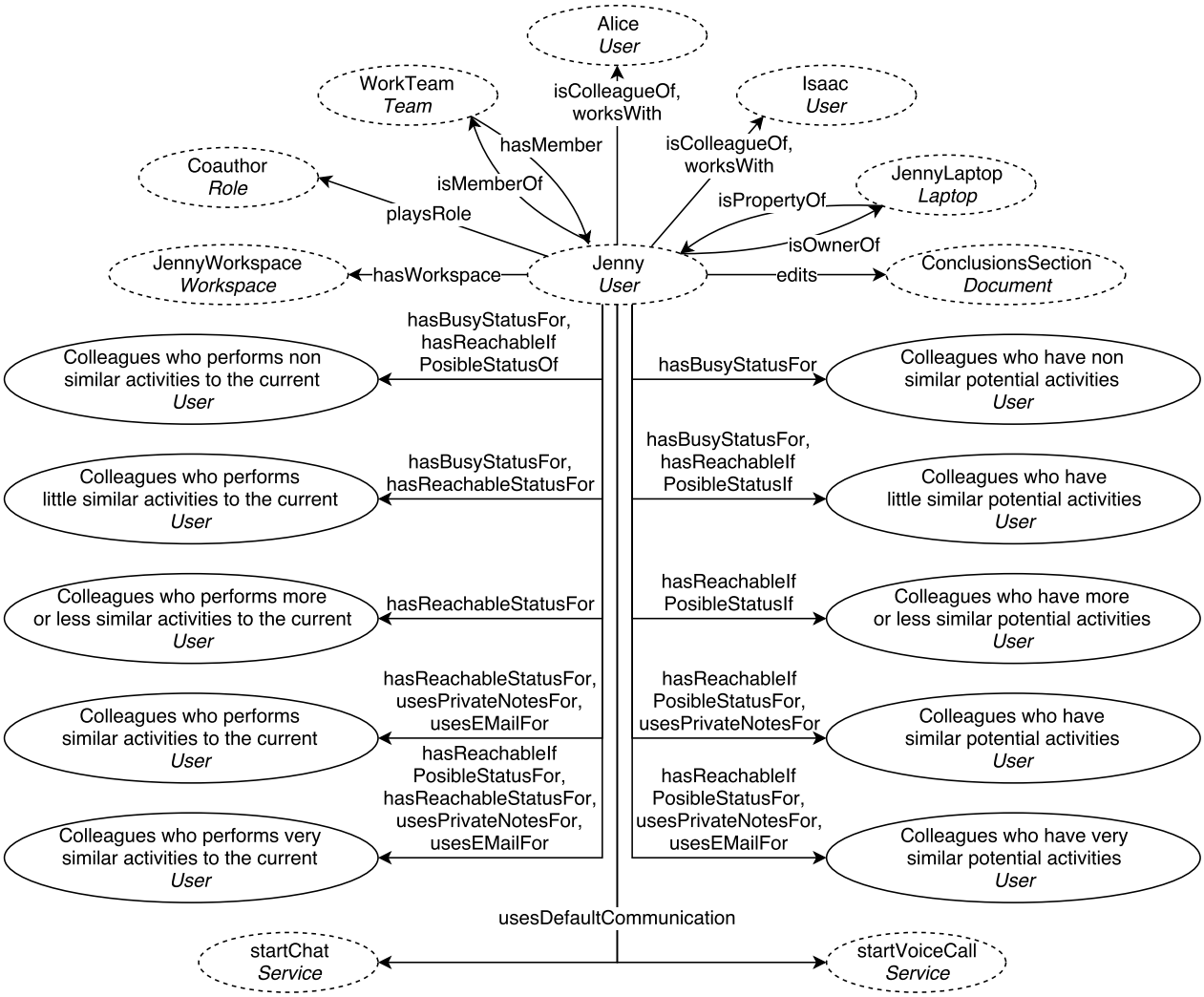
(c) Propiedades del rol `Coauthor`



(d) Propiedades del usuario `Alice`



(e) Propiedades del usuario `Isaac`



(f) Propiedades del usuario `Jenny`

Figura 4.4: Modelado semántico de la *Herramienta de medios de contacto y disponibilidad*

4.4.4. Herramienta 4: Escritorio enriquecido

Descripción del escenario

Una instancia del escritorio enriquecido ofrece tres aplicaciones, cuyos objetos pueden estar relacionados entre sí mediante referencias deícticas para enriquecer la interacción, y son:

1. **Pizarrón multi-usuario**: permite que los colaboradores realicen lluvias de ideas y dibujar diagramas con extensión JPEG.

2. **Editor colaborativo**: permite la producción concurrente de archivos HTML a través de su fragmentación. Una imagen desplegada en este editor puede ser arrastrada hacia el *pizarrón multi-usuario* para su edición, siendo que sus actualizaciones serán reflejadas automáticamente en el editor.

3. **Servicio de mensajería**: soporta la comunicación directa entre los miembros de un grupo. Además, es capaz de relacionar diferentes conversaciones que hacen referencias deícticas a un mismo objeto compartido.

El *pizarrón multi-usuario* y el *editor colaborativo* soportan tres roles:

1. **Coautor**: permite escribir texto y dibujar diagramas,

2. **Lector**: posibilita leer texto y visualizar imágenes, y

3. **Revisor**: permite hacer comentarios respecto a la forma y al contenido de documentos y diagramas.

Suponga que un grupo compuesto por cuatro miembros, Alice, Isaac, Jenny y Sandy, está preparando un reporte técnico de manera distribuida. Aunque los miembros del grupo utilizan dispositivos heterogéneos, éstos tienen características técnicas suficientes (e.g., buen tamaño y alta definición de pantalla, así como alto

poder de procesamiento y capacidades de comunicación). Dadas las características de los dispositivos, la adaptabilidad de las aplicaciones colaborativas no se expresa mediante la remodelación de su interfaz de usuario, sino por las funcionalidades que dichas aplicaciones proveen para que los miembros del grupo puedan llevar a cabo sus actividades.

El equipo de trabajo ha dividido el documento en secciones, así que cada miembro está encargado de escribir una sección, mientras sus colegas son responsables de revisarla. De esta manera, los colaboradores pueden escribir el documento de manera concurrente. En particular, Sandy e Isaac han estado trabajando juntos porque sus secciones respectivas están relacionadas. El mismo caso ocurre con Alice y Jenny, ya que ambas están encargadas de dibujar todas las figuras del reporte técnico.

Cuando Sandy termina de redactar la primera versión de su sección, le pide a Isaac que la revisen conjuntamente, ya que ella tiene algunos problemas para expresar sus ideas. En consecuencia, estos colaboradores deciden entablar una comunicación directa mediante la mensajería instantánea, mientras que el editor colaborativo está a cargo de mostrarles los párrafos sujetos a revisión.

El escritorio enriquecido provee una funcionalidad de deixis, la cual permite a los colaboradores crear referencias deícticas, en forma de ligas de hipertexto, entre objetos de la herramienta de la mensajería instantánea (e.g., palabras deícticas como *this figure*, *the next section*, *this paragraph*, *those references* o *here*) y objetos del editor colaborativo (e.g., un título, un párrafo, una frase, una palabra, una referencia bibliográfica, una figura o una sección). Gracias a ésta funcionalidad, los colaboradores no necesitan copiar los objetos referenciados desde el editor colaborativo hacia la mensajería instantánea cuando están revisando. De este modo, los objetos referenciados siempre mantienen su contexto en el editor colaborativo, i.e., párrafos anteriores y siguientes, figuras asociadas y referencias bibliográficas.

Mediante la funcionalidad *deixis*, Isaac escribe en la mensajería instantánea las palabras deícticas "...*in this paragraph*...", las cuales tienen una liga de hipertexto que apunta a un párrafo del documento abierto en el editor colaborativo. Cuando Sandy da clic en la liga creada por Isaac, el editor colaborativo muestra el párrafo correspondiente a la mitad de la vista y lo resalta con letra de color roja, en este caso.

El párrafo en cuestión mantiene su contexto, i.e., referencias bibliográficas, párrafos alrededor y figuras referenciadas.

Mientras Sandy e Isaac están produciendo una nueva versión del párrafo, Alice y Jenny también decidieron usar la mensajería instantánea para hablar acerca de los diagramas que han creado. Alice escribió las palabras deícticas "*in this diagram*". Al igual que Isaac, Alice creó una referencias deíctica a una figura del reporte técnico. De este modo, cuando Jenny da clic en estas palabras, el editor colaborativo muestra el diagrama correspondiente a la mitad de la vista resaltándolo mediante un borde color azul. Este diagrama referenciado también mantiene su contexto, i.e., sección contenedora, título de la figura, párrafos circundantes.

Como no es posible modificar este diagrama desde el editor colaborativo, el escritorio enriquecido compartido provee una funcionalidad que permite a los colaboradores hacer una réplica de tal diagrama en el *pizarrón multi-usuario*, donde éste puede ser modificado. Esta réplica mantiene una referencia contextual al diagrama mostrado en el editor colaborativo (en este caso, un identificador de objeto interno y la localización del objeto en el documento). Por lo tanto, gracias a esta referencia contextual, el escritorio enriquecido permite a los colaboradores reemplazar la versión caducada del diagrama mostrado en el editor colaborativo, por la nueva versión creada en el pizarrón multi-usuario.

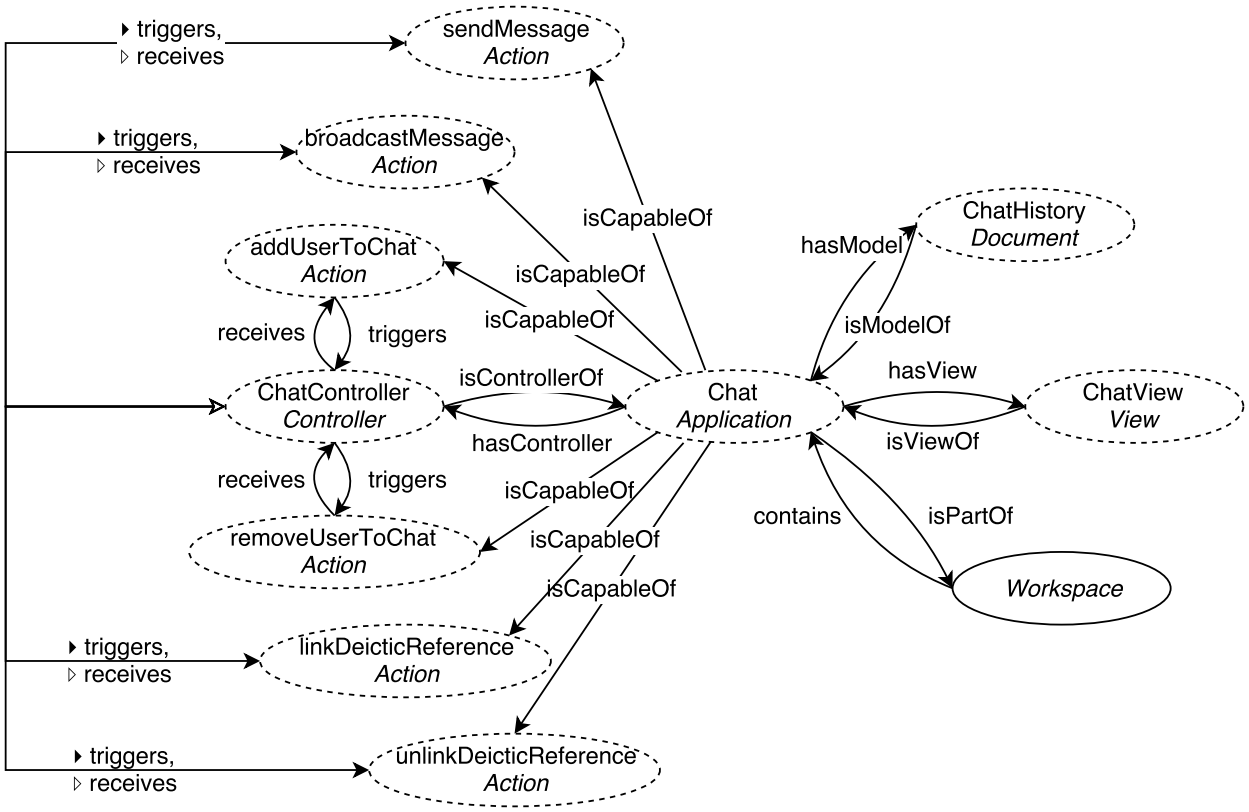
Como Alice y Jenny decidieron modificar el diagrama, una réplica es automáticamente creada en el pizarrón multi-usuario. En este momento, el escritorio enrique-

> cido deduce que Sandy e Isaac están escribiendo un párrafo que hace referencia a la figura que está siendo modificada por Alice y Jenny. En consecuencia, el escritorio enriquecido sugiere a Sandy e Isaac seguir la conversación de Alice y Jenny porque puede ser de interés, ya que ellas están modificando la figura asociada al párrafo que ellos están redactando. Esta sugerencia también es hecha a Alice y Jenny. En caso de que Sandy e Isaac estén interesados en seguir la conversación de Alicde y Jenny y viceversa, ellos pueden acceder a estas conversaciones por medio de una nueva pestaña agregada en sus respectivos mensajeros instantáneos [Sánchez Morales, 2013].

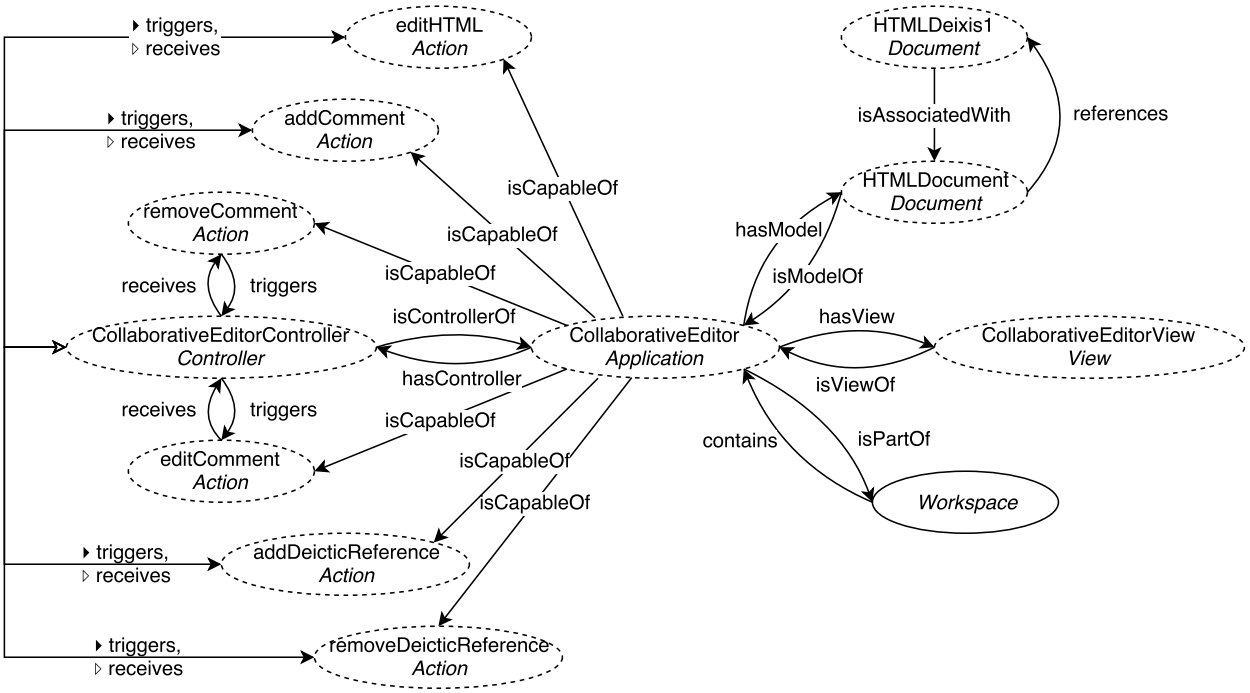
La Tabla 4.4 muestra las entidades y sus respectivas variables contextuales de la *herramienta escritorio enriquecido*, que está asociada a los escenarios *mejoramiento del trabajo colaborativo* y *remodelación de la interfaz de usuario.*

Cuadro 4.4: Entidades y sus variables contextuales de la herramienta *escritorio enriquecido*

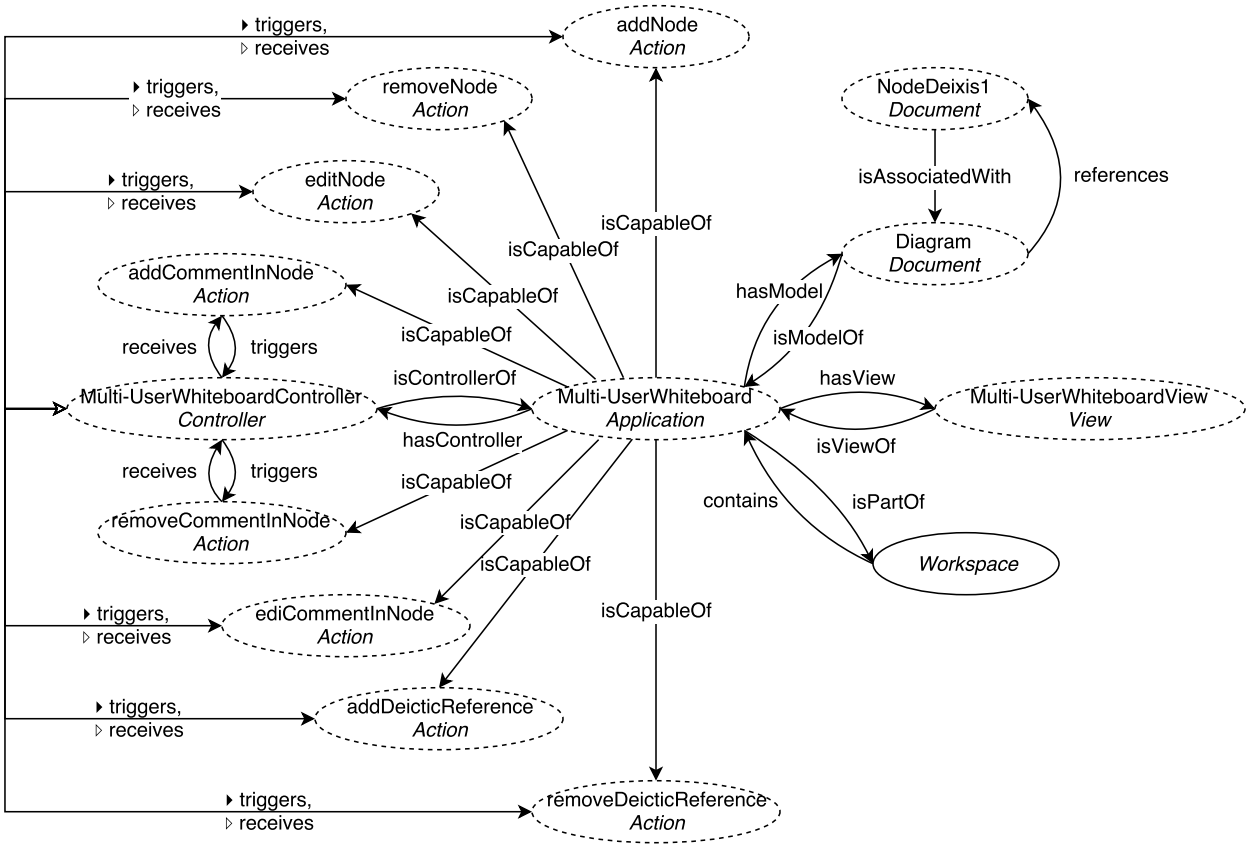
Entidad	**Variables contextuales**
Conversación	Referencia deíctica (sección, párrafo, frase, palabra, diagrama, línea, figura geométrica)
Documento	Objetos compartidos en uso
Usuario	Actividad, conversaciones actuales, rol (*coautor*, *lector* y *revisor*)



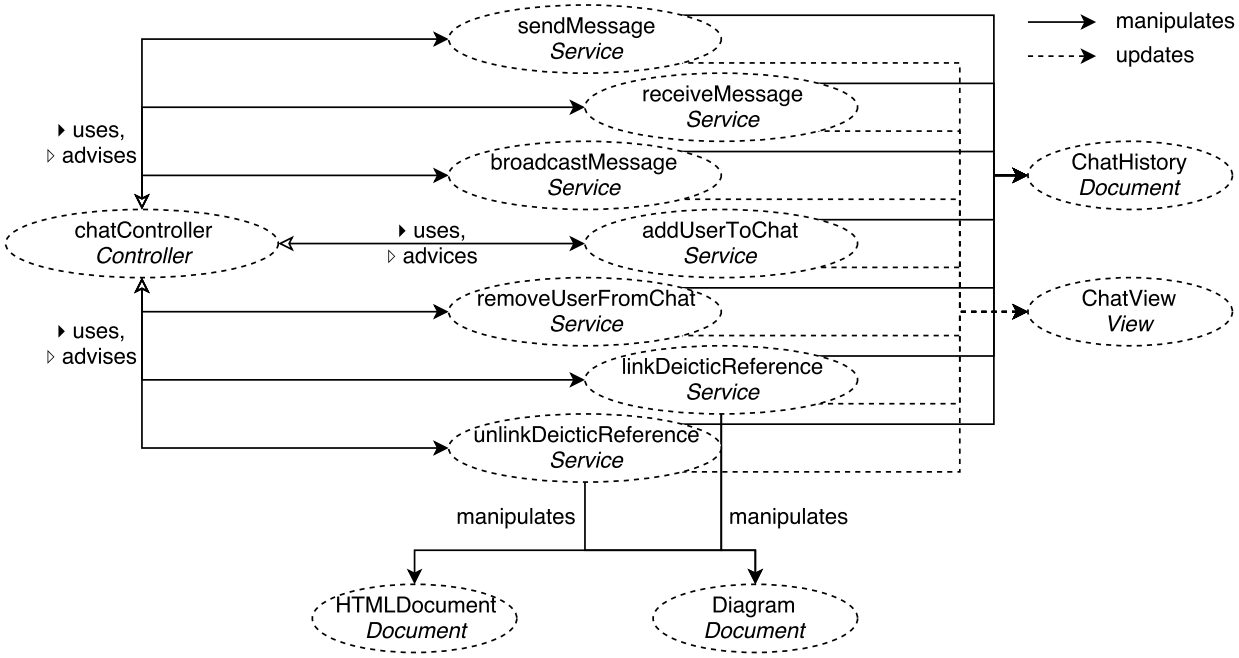
(a) Propiedades de la aplicación `Chat`



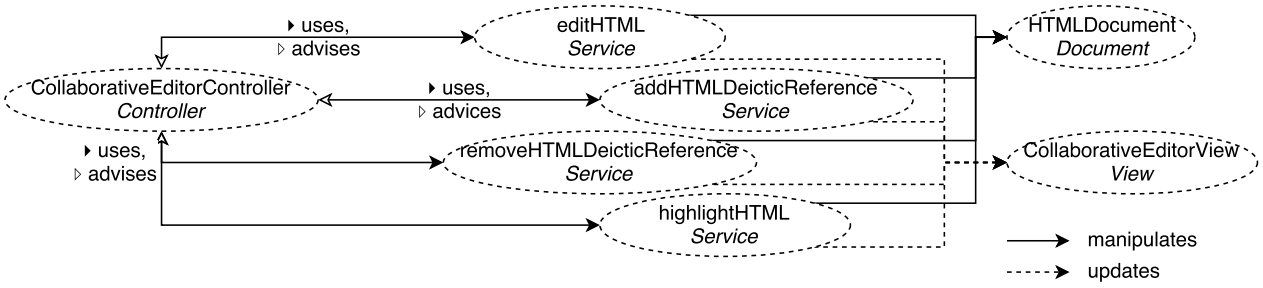
(b) Propiedades de la aplicación `Collaborative Editor`



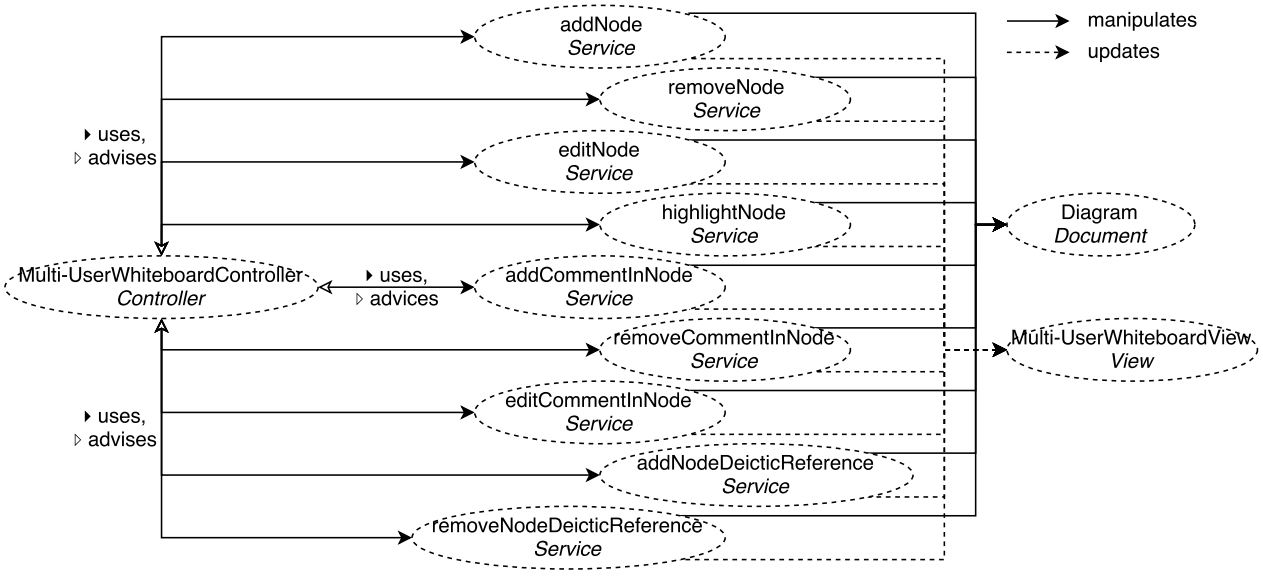
(c) Propiedades de la aplicación `Multi-UserWhiteboard`



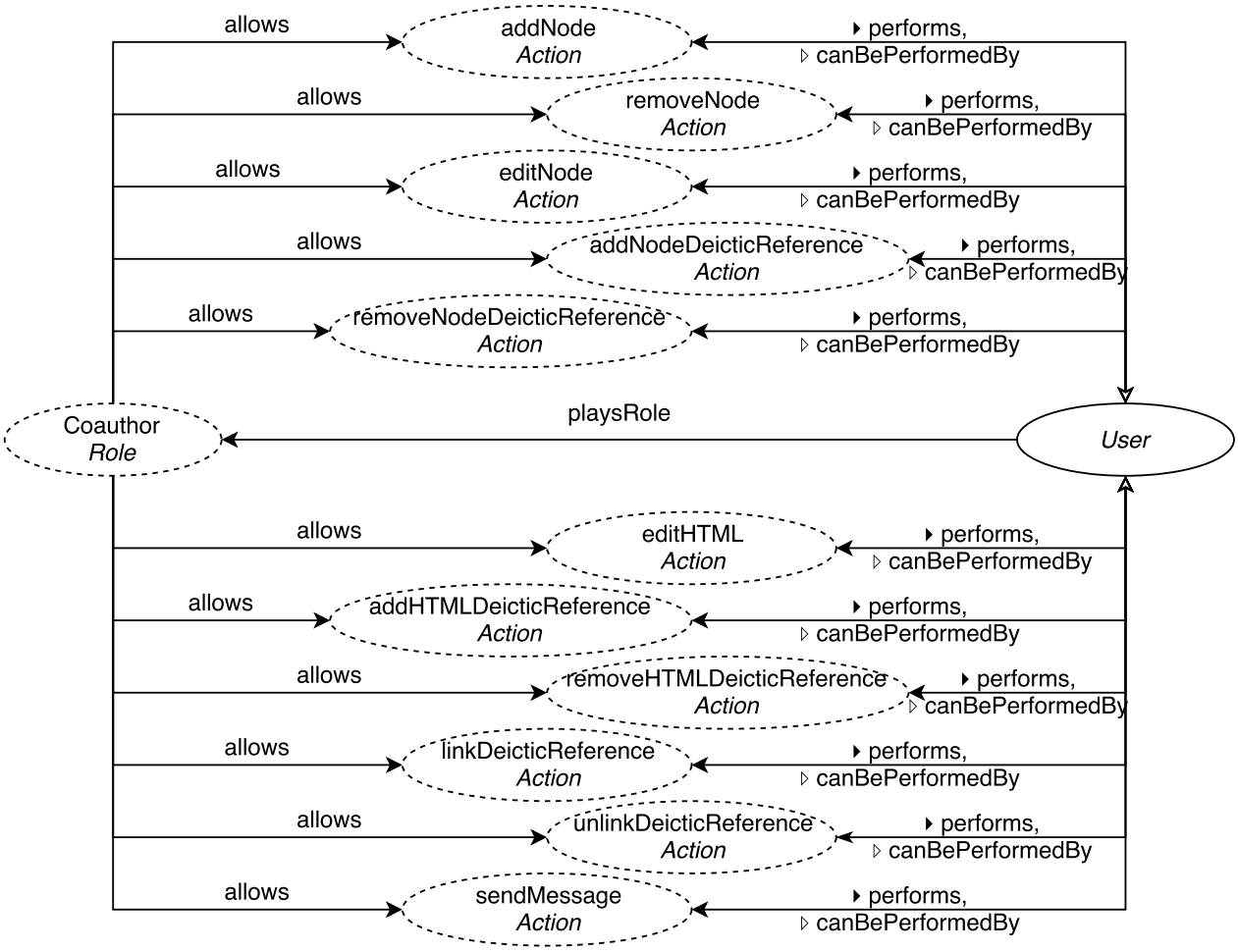
(d) Servicios de la aplicación `Chat`



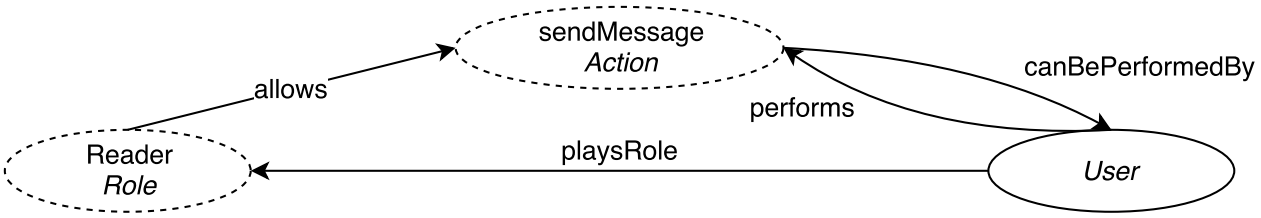
(e) Servicios de la aplicación `CollaborativeEditor`



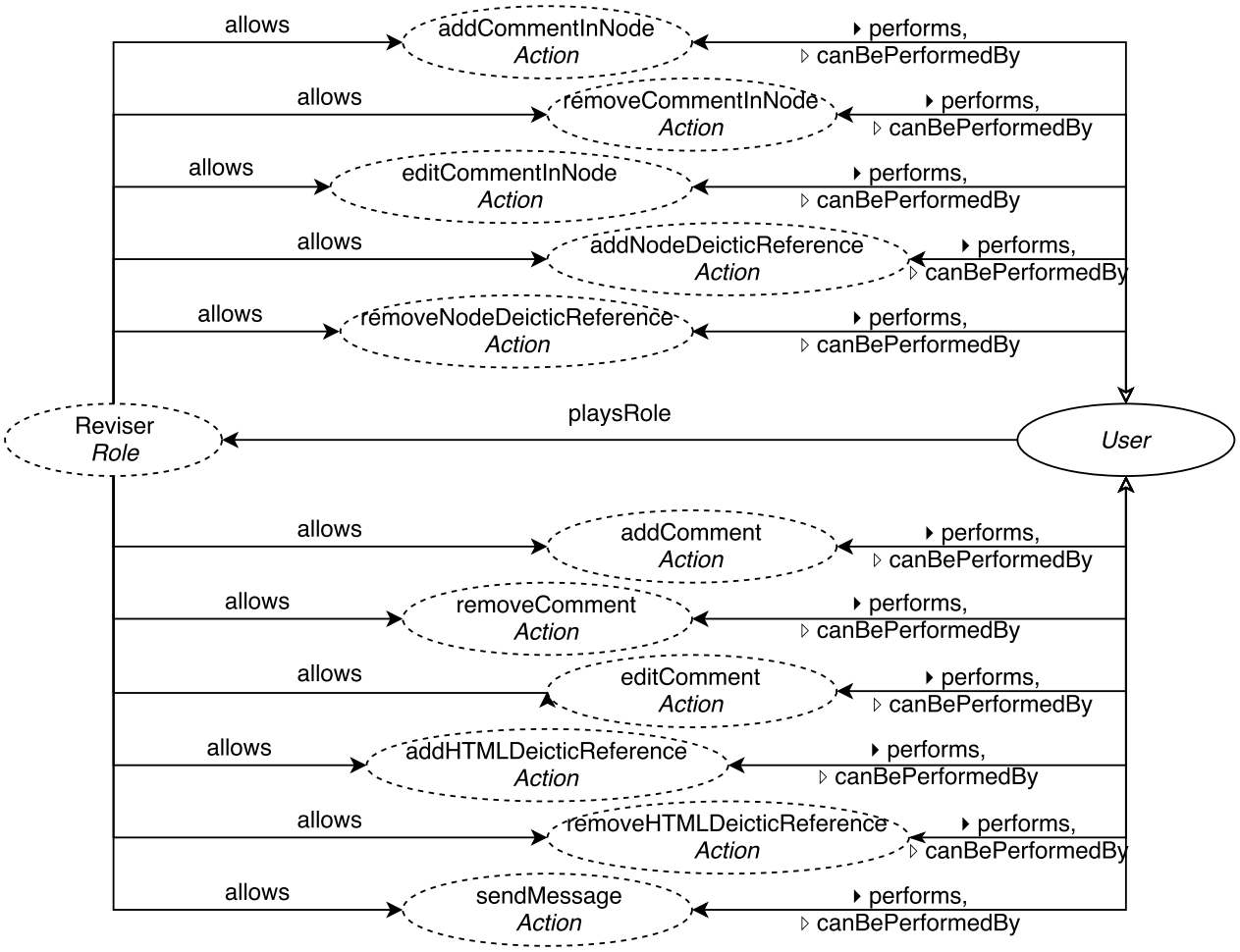
(f) Servicios de la aplicación `Multi-UserWhiteboard`



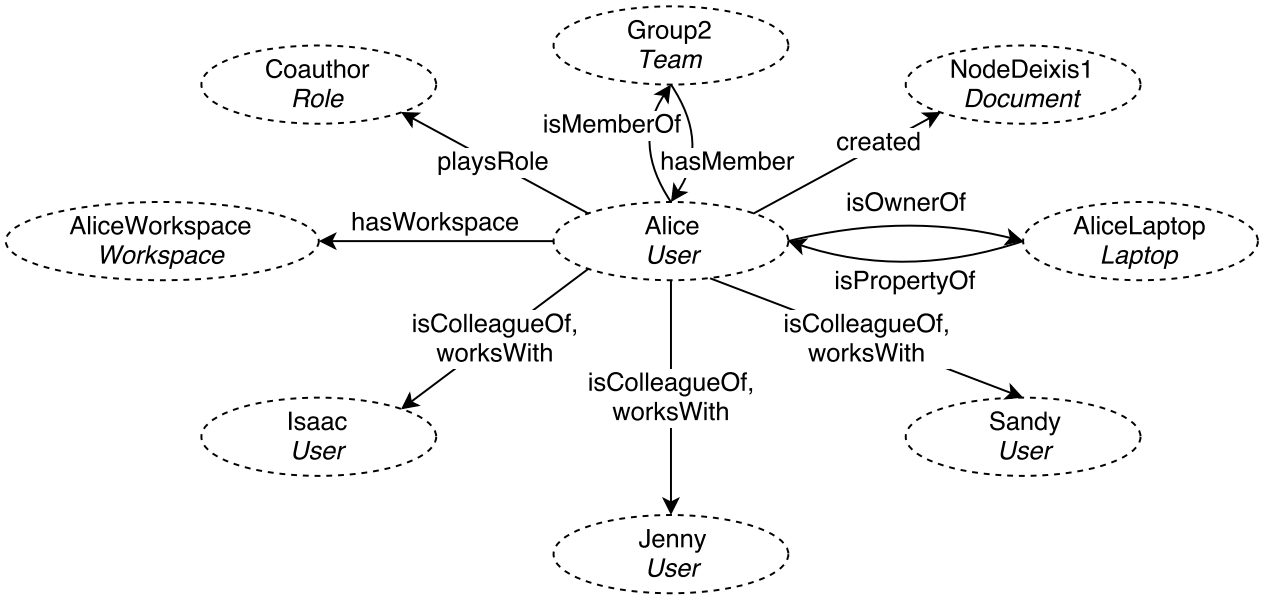
(g) Propiedades del rol `Coauthor`



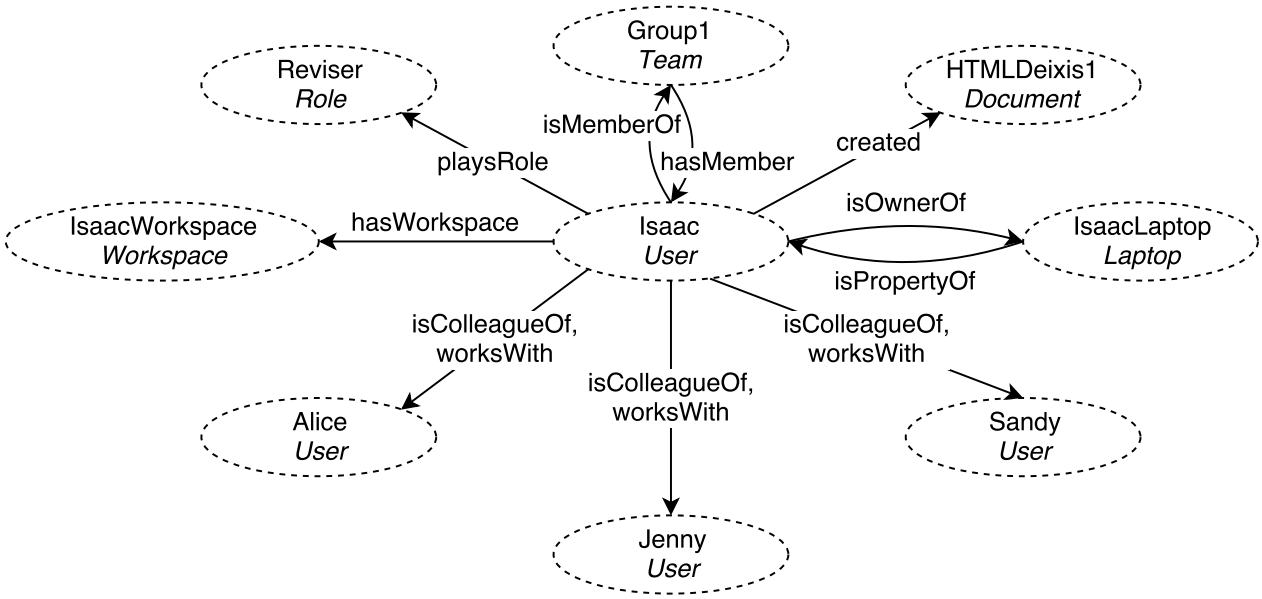
(h) Propiedades del rol `Reader`



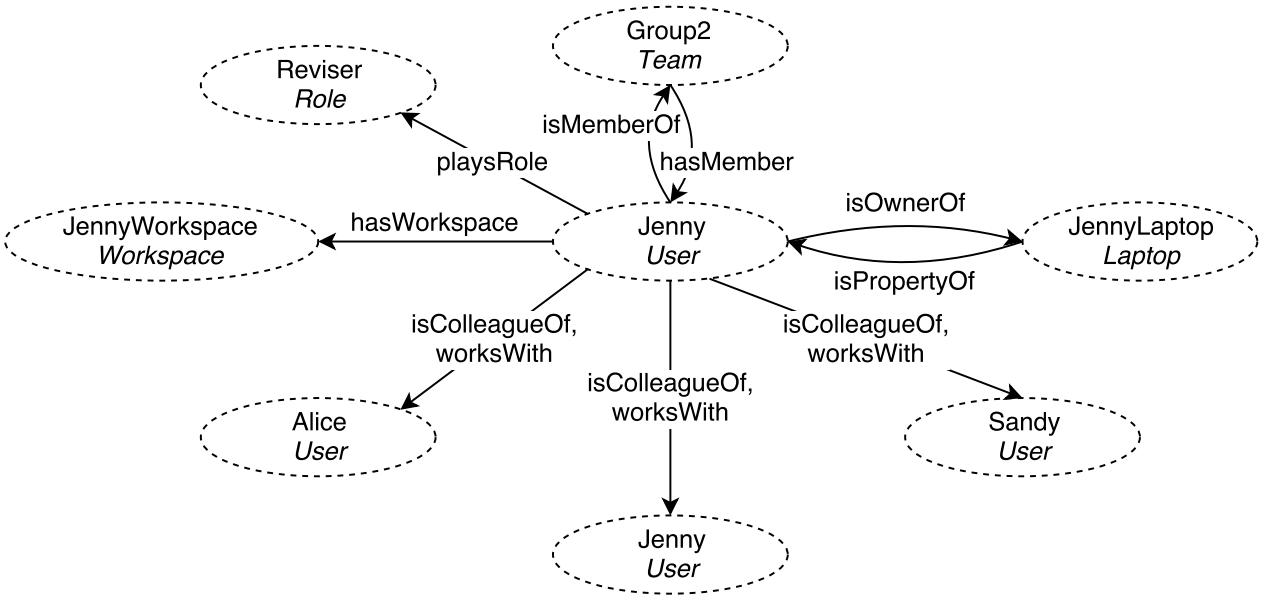
(i) Propiedades del rol `Reviser`



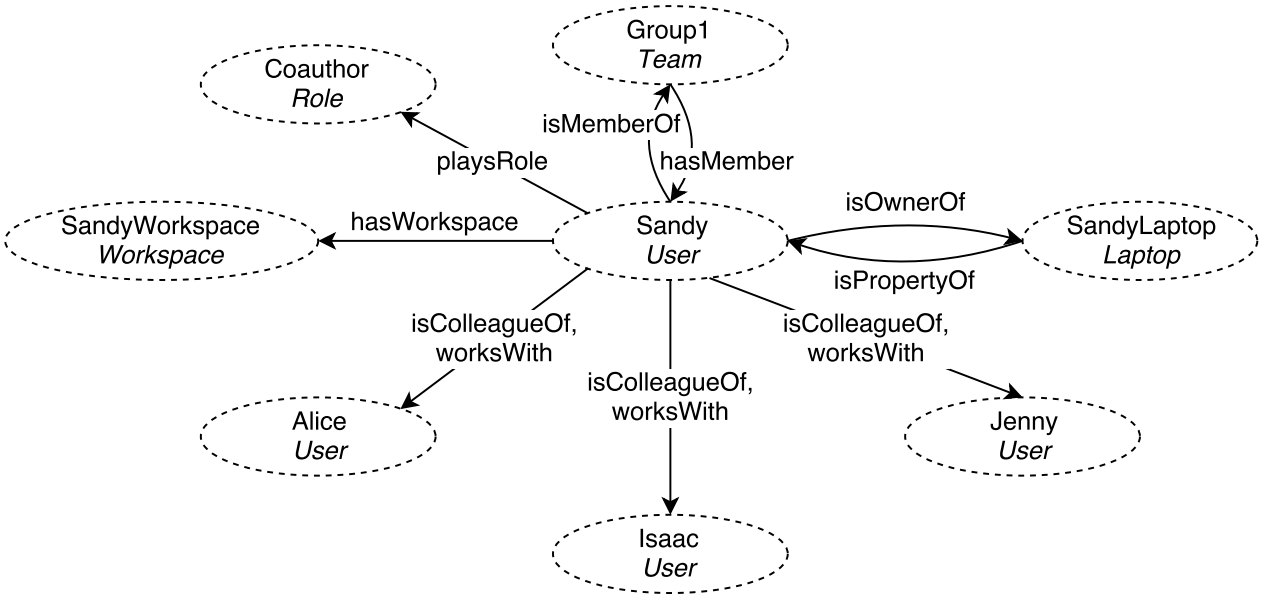
(j) Propiedades del usuario `Alice`



(k) Propiedades del usuario `Isaac`



(l) Propiedades del usario `Jenny`



(m) Propiedades del usuario `Sandy`

Figura 4.5: Modelado semántico de la herramienta *Escritorio enriquecido*

4.4.5. Herramienta 5: Editor de textos

Supongamos que una empresa de desarrollo de software recibió un reporte de un error grave en un sistema de administración que ellos desarrollaron. Dada la gravedad del problema, la empresa de desarrollo lanzó una convocatoria a todos sus desarrolladores con la finalidad de encontrar y proponer una solución a dicho error de software. Por su parte, dicha empresa premiará al grupo que resuelva el problema.

Un grupo de colaboradores, compuesto por Sandy, Alice e Isaac, encontraron el problema en el software. Así que ellos decidieron hacer un documento explicando tanto los errores encontrados así como la solución propuesta mediante diagramas de clases y diagramas de estado. Para la redacción del documento usan el editor de textos.

Después de que el grupo decide el contenido del reporte, ellos se reparte la redacción del documento de la forma siguiente: Sandy tiene asignada la tarea de escribir los errores encontrados; Isaac debe hacer y describir los diagramas de clase, y Alice tiene que hacer y redactar los diagramas de estado,

Debido a que el grupo de colaboradores pasó mucho tiempo en una oficina buscando la solución; ellos deciden trabajar de manera distribuida en la redacción del documento, Sandy se retira de la oficina, en donde estaban trabajando, para salir a comer en el comedor de la empresa. Ella decide empezar a trabajar en la redacción del documento mientras espera que le entreguen la comida. Cuando ella va a la mitad de la redacción le surgen dudas de los errores encontrados en el software, así que ella decide entablar una comunicación con el resto del grupo por medio de la mensajería instantánea, la cual forma parte del editor. Mientras el grupo de trabajo está participando en la conversación, Sandy puede ver en su pantalla tanto el documento que están generando así como la conversación del

grupo. Dentro de la conversación, Sandy hace un pregunta al resto del equipo.

Poco tiempo después de iniciada la comunicación, un programador ajeno al equipo se acerca a platicar con Sandy. Dicho programador sabe que el equipo de ella ha encontrado tanto el error como la solución al problema de software, por tal motivo él decide acercarse a la computadora portátil de ella para intentar leer tanto el reporte como su conversación con el equipo. Tan pronto el programador se acerca a la computadora, el sistema del editor detecta que una persona ajena al grupo está observando de frente la pantalla de la computadora de Sandy. Como consecuencia el sistema oculta tanto el reporte como la conversación que ella lleva a cabo.

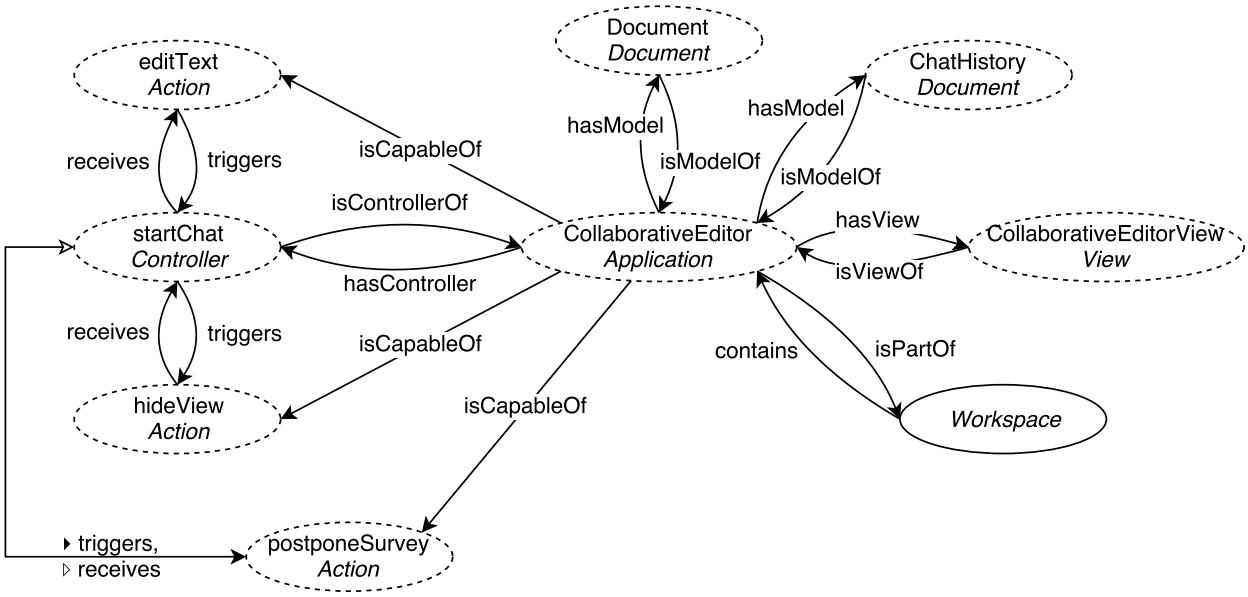
Por otro lado, Alice responde a la pregunta generada por Sandy; pero observa que Sandy no responde a la explicación dada. De repente Alice nota que la foto de Sandy cambió de color. Cuando Alice ubica su cursor del ratón sobre la foto de Sandy, el sistema muestra un mensaje indicando que una persona ajena al grupo está observando la pantalla de Sandy. Por tal motivo ella no puede observar los mensajes enviados ni las actualizaciones, así como tampoco puede desarrollar sus actividades.

La Tabla 4.5 muestra las entidades y sus respectivas variables contextuales de la *herramienta editor de textos*, que está asociada al escenario *manipulación de la información*.

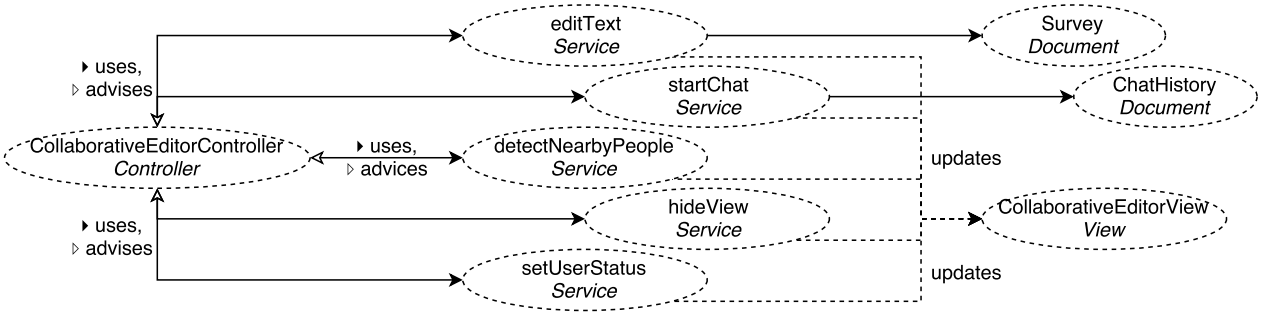
Cuadro 4.5: Entidades y sus variables contextuales de la *herramienta editor de textos*

Entidad	**Variables contextuales**
Recurso	Colaboradores autorizados a la visualización y manipulación del recurso
Usuario	Grupo de trabajo, ubicación física

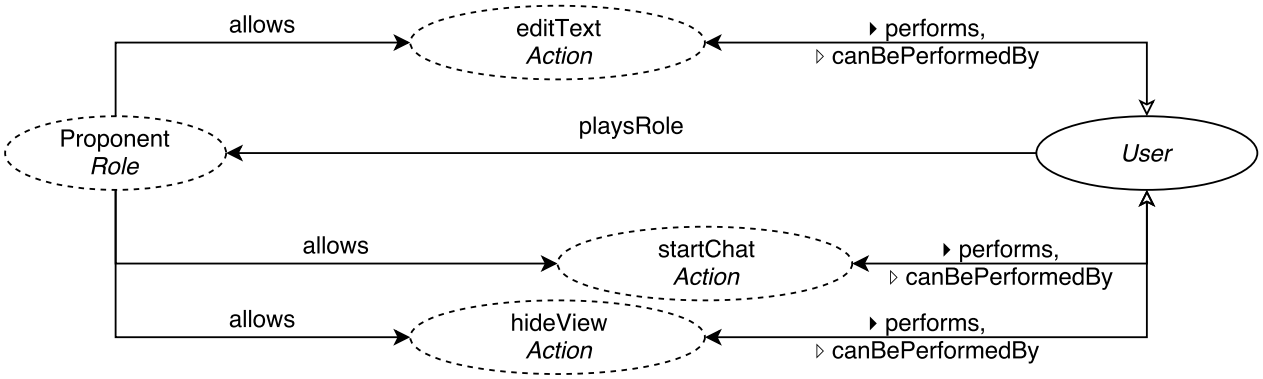
Un *recurso* es un archivo (e.g. un documento) o parte del mismo (e.g., un diagrama contenido en un documento).



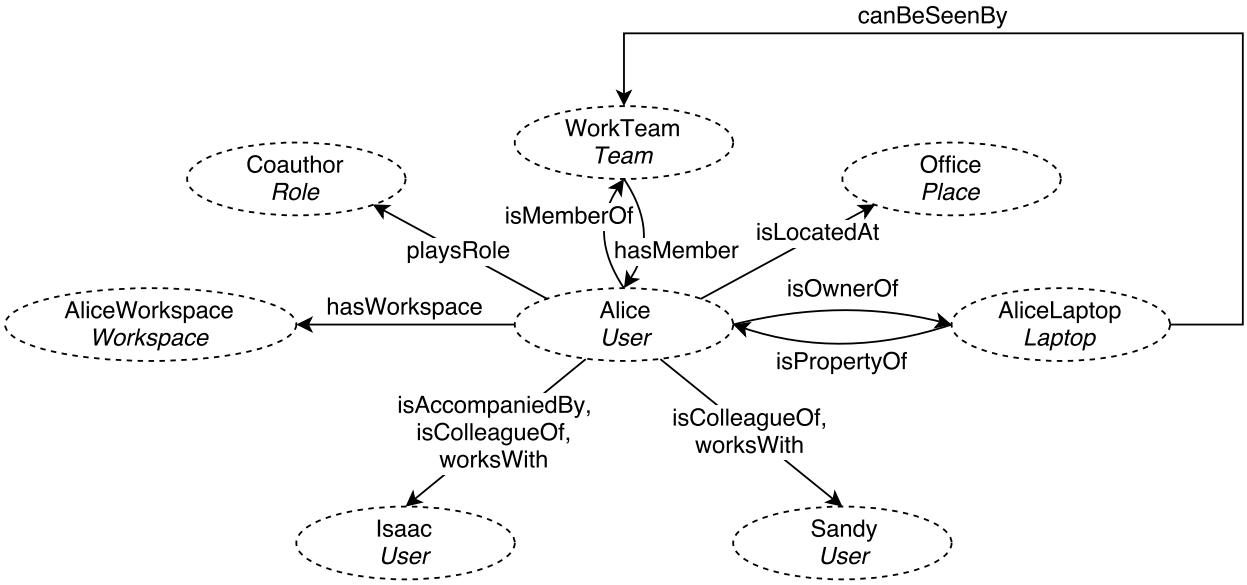
(a) Propiedades de la aplicación `CollaborativeEditor`



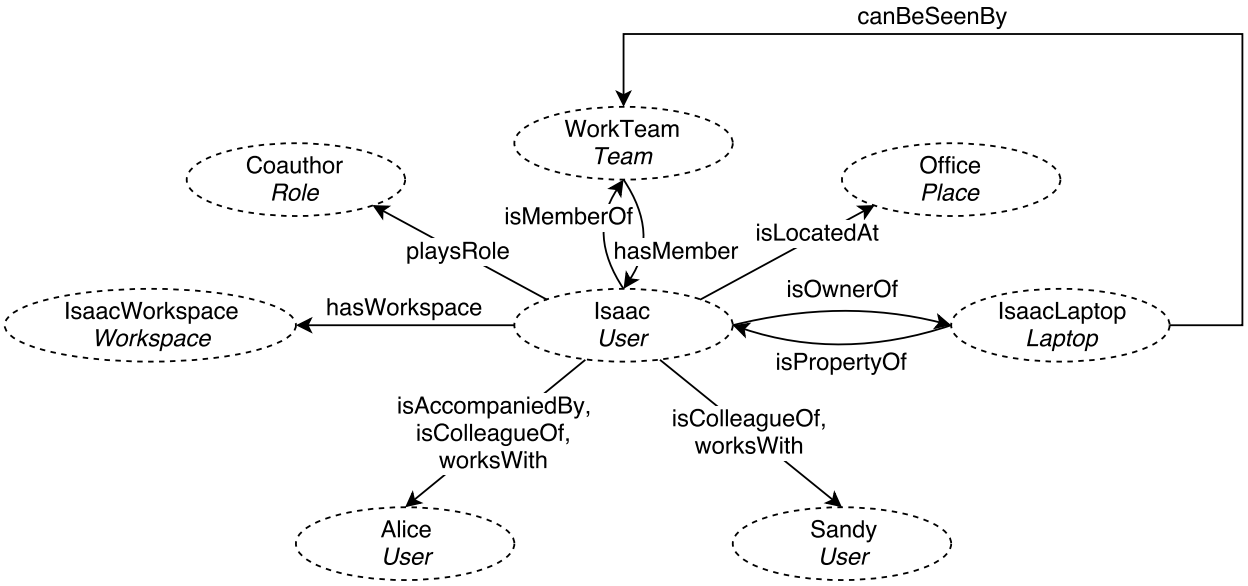
(b) Servicios de la aplicación `Collaborative Editor`



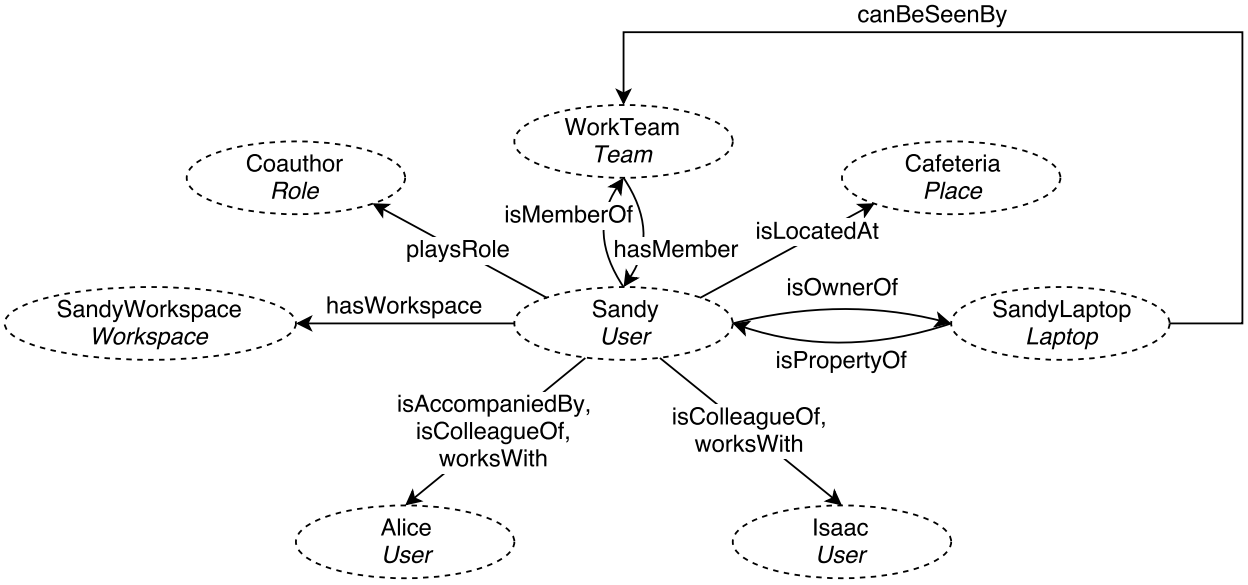
(c) Propiedades del rol `Coauthor`



(d) Propiedades del usuario `Alice`



(e) Propiedades del usuario `Isaac`



(f) Propiedades del usuario `Sandy`

Figura 4.6: Modelado semántico de la herramienta *Editor de textos*

Capítulo 5

ARQUITECTURA CONCEPTUAL

En este capítulo se detallan el diseño y la implementación para la arquitectura conceptual. Específicamente, en la Sección 5.1 se describe el ciclo de adaptación de una aplicación los cambios contextuales de una aplicación. Posteriormente, las siguientes secciones describen las etapas de la arquitectura conceptual propuesta, la Sección 5.2 explica la etapa Percepción de eventos contextuales; la Sección 5.3, la etapa Detección de la situación; la Sección 5.4, la etapa Comunicación de la situación; y la Sección 5.5, la etapa Adaptación del sistema. Finalmente, la Sección 5.6 muestra la implementación para la arquitectura conceptual.

5.1. Ciclo de adaptación

El ciclo de adaptación de una aplicación a los cambios contextuales se compone de las siguientes etapas:

1. Acción del usuario o cambio en el ambiente,
2. detección,
3. selección de las reglas de adaptación, y
4. ejecución de las reglas de adaptación.

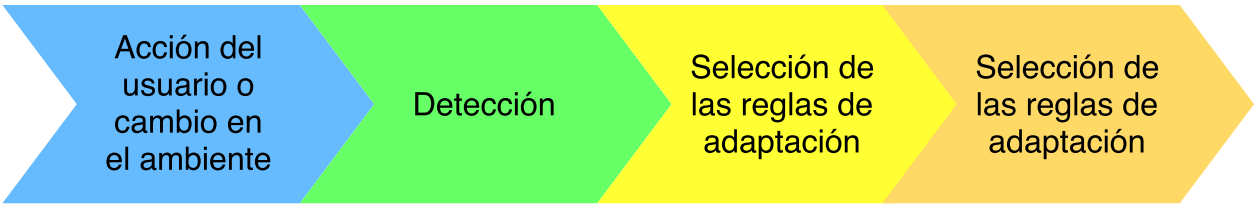


Figura 5.1: Etapas del ciclo de adaptación

En la **detección** se supervisa el comportamiento del usuario, dado que sus acciones pueden modificar el ambiente y el estado del sistema, las condiciones del ambiente y las condiciones del sistema. Para ello, se emplea el componente de software llamado *motor de detección* que mediante diversas técnicas representa la información de forma abstracta, inicializa el modelo de estado, y lo monitoriza.

En la etapa siguiente, la **selección de reglas de adaptación**, el componente de software llamado *motor de contextualización* decide qué información del modelo de estado es relevante y construye el estado contextualizado. Con base en éste, se activan algunas reglas de adaptación y se crea una secuencia de ejecución.

Finalmente, en la **ejecución de las reglas de adaptación**, el componente de software llamado *motor de adaptación* procesa la secuencia de adaptación antes creada, entonces se actualizan el ambiente colaborativo y el modelo de estado.

La arquitectura conceptual propuesta ofrece un soporte para integrar la sensibilidad al contexto en sistemas colaborativos, permitiendo su adaptación con base en los cambios contextuales en el ambiente colaborativo.

Con base en el ciclo de adaptación, y con base de la arquitectura propuesta en [Olivares Toledo, 2011], se han diseñado las siguientes etapas (véase la Figura 5.2):

1. **Percepción de eventos contextuales**. Corresponde a la etapa de *detección* del ciclo de adaptación. En esta etapa, se monitorizan las variables a las cuales está sujeto un ambiente colaborativo, cuyos cambios significativos producen *eventos contextuales*,

2. **Detección de la situación**. Corresponde a la etapa de *selección de reglas de adaptación*. Forma parte del *servicio de contexto*, que recibe los eventos contextuales y con base en ellos determina si una situación nueva ha sido activada (e.g., *Sonia* ha entrado a su oficina) o si una situación previa ha sido desactivada (e.g., *Ismael* ha abandonado el proyecto p),

3. **Comunicación de la situación**. También forma parte del *servicio de contexto*, que notifica los cambios en el estado de una situación a los nodos del sistema colaborativo interesados en recibir tales actualizaciones. Se sugiere el uso del patrón de mensajería *Publicación/Suscripción* [Eugster et al., 2003], y

4. **Adaptación del sistema**. Corresponde a la etapa de *ejecución de las reglas de adaptación*. Los nodos del sistema colaborativo que han sido notificados ejecutan una serie de instrucciones para adaptarse a las situaciones en curso.

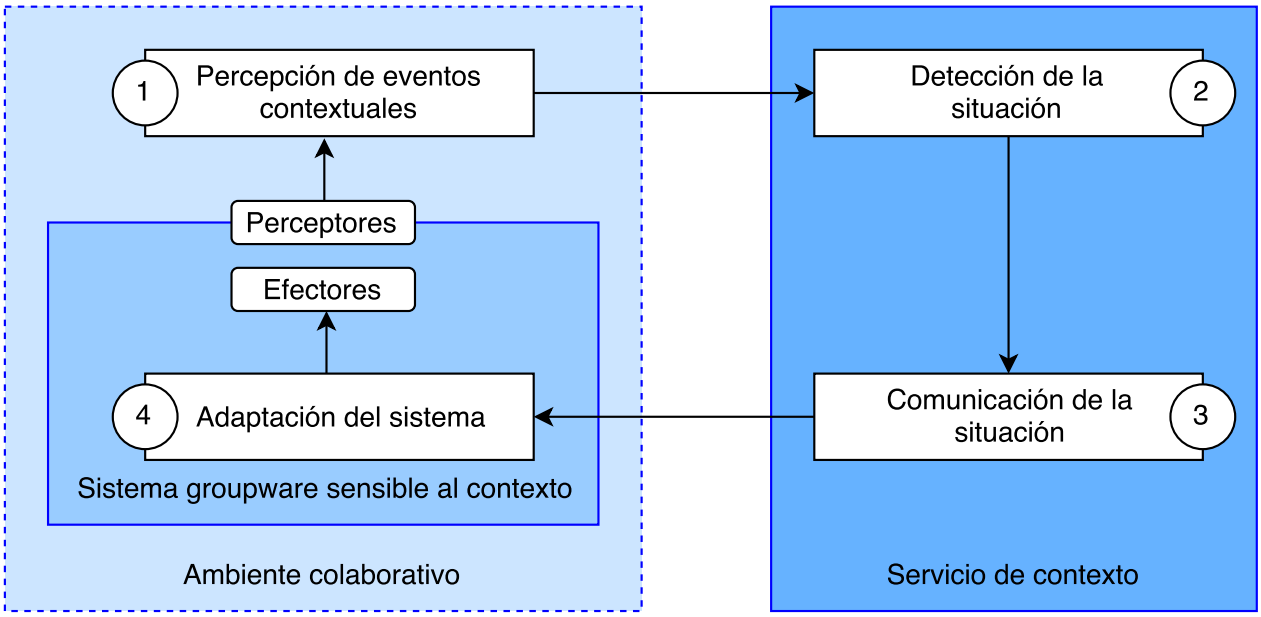


Figura 5.2: Fases de la arquitectura conceptual

5.2. Percepción de eventos contextuales

Un ambiente colaborativo está compuesto por cada una de las entidades que son relevantes entre el usuario y la aplicación, incluyéndose. Cada una de estas entidades está sujeta a diferentes variables, pero sólo se consideran aquellas cuyo cambio es relevante para llevar a cabo una adaptación, llamadas *variables contextuales*. Los cambios significativos en las variables contextuales producen *eventos contextuales* y éstos, pueden conducir a la activación o desactivación de situaciones.

Dependiendo de su naturaleza, los eventos contextuales se clasifican en *físicos* y *lógicos*.

5.2.1. Eventos contextuales físicos

Los eventos contextuales físicos reflejan cambios significativos en variables físicas (e.g., la ubicación, los colaboradores y los dispositivos detectados, y el ruido). Las fuentes de estos eventos son cambios en la naturaleza y las acciones del usuario. Con los sensores de hardware disponibles es posible medir casi cualquier información física [Baldauf et al., 2007]. En la Tabla 5.1 se presentan ejemplos de variables físicas y sus sensores asociados.

Cuadro 5.1: Ejemplos de variables físicas y sus respectivos sensores

Variable física	**Sensores de hardware**
Light	Color sensors, infrared sensors, photodiodes, and ultraviolet sensors
Audio	Microphones
Temperature	Infrared sensors, and thermometer
Touch	Contact sensors, and switches
Distance	Infrared and ultrasonic sensors
Olfactory	Electrochemical sensors

En algunos casos se requiere el uso de múltiples sensores, ya sea para que cierta información física, como la ubicación simbólica (e.g., un salón de clases, o la sala de juntas) de un colabo-

rador, sea determinada o, en otros casos, como determinar el número de personas presentes en una habitación, sea confiable.

Considérese la siguiente premisa de diseño:

> "*La gente y los dispositivos ubicados en el mismo lugar son generalmente afectados por las mismas variables físicas*".

Con base en esta premisa se propone el elemento *perceptor*. Un perceptor es un componente de software cuyas tareas principales son: 1) supervisar los sensores de hardware ubicados en una determinada ubicación, 2) determinar cuáles eventos contextuales tienen que ser activados, ayudándose del preprocesamiento, y 3) notificar al servicio de contexto de los eventos contextuales activados.

El perceptor debe ser capaz de identificar todos los sensores de hardware co-localizados en una misma ubicación (e.g., sala de juntas, oficina). Por ello se usa la *Tabla de sensores*, que asocia cada variable física supervisada con los identificadores de sus respectivos sensores de hardware.

Para ilustrar esta situación, supóngase que en una habitación se hayan los siguientes sensores: 1) termómetro, 2) sensor infrarrojo, 3) micrófono, y 4) foto-diodos. La relación entre variables físicas y sensores de hardware se muestra en la Tabla 5.2.

Cuadro 5.2: Ejemplo de la tabla de sensores de un perceptor

Variable física	**Sensores asociados**
`Audio`	`microphone`
`Light`	`infrared sensor`, `photodiode`
`Temperature`	`infrared sensor`, `thermometer`

Un *evento contextual* se produce por cambios significativos de las variables físicas. Para determinar cuándo se ha producido un cambio significativo se requiere definir rangos que simbolicen posibles estados delimitados por umbrales y, de esta forma, sólo si hay un cambio

en el estado actual de la variable física se activará el evento contextual correspondiente.

Si se considera la variable contextual `temperature` de un perceptor ubicado en una oficina, sus posibles estados y umbrales se muestran en la Tabla 5.3.

Cuadro 5.3: Estados y umbrales de la variable contextual `temperature`

Estado	**Umbral inferior**	**Umbral superior**
`Cold`	$-\infty$	$15°$
`Warm`	$16°$	$24°$
`Hot`	$25°$	∞

El conjunto de estos estados forman el vocabulario contextual. Éste permite a los desarrolladores de aplicaciones utilizar descripciones abstractas en lugar de tratar con información técnica [Baldauf et al., 2007], e.g., es más sencillo entender el término "*warm*" que el intervalo de valores expresados en grados. El vocabulario contextual puede almacenarse dentro de una tabla, con los campos *variable contextual*, *estado*, *umbral inferior*, y *umbral superior*.

La Tabla 5.4 muestra ejemplos de vocabulario contextual.

Cuadro 5.4: Ejemplos de vocabulario contextual

Variable física	**Vocabulario contextual**
`Light`	`Dark`, `normal`, `bright`
`Temperature`	`Cold`, `warm`, `hot`
`Projector status`	`On`, `off`
`Number of persons`	`Empty room`, `few people`, `saturated room`

Una vez detectado al menos un cambio de estado, el perceptor debe considerar realizar preprocesamiento, como la *fusión de sensores* y la *calidad del contexto*.

1. **Fusión de sensores**. Se da cuando se utilizan dos o más sensores, ya sean iguales o diferentes, para medir la misma variable física, y tiene como propósito aumentar la precisión de la medición [Calvary et al., 2001], e.g., para obtener la temperatura promedio en una habitación pueden colocarse varios termómetros.

2. **Calidad del contexto**. Es la medición de los defectos de la información monitorizada a causa de las fallas y el ruido en los sensores y en la red de comunicaciones [McKeever et al., 2008]. Algunos errores pueden ser corregidos; otros, sin embargo, provocarán descartar la medición del sensor.

5.2.2. Eventos contextuales lógicos

Los eventos contextuales lógicos reflejan cambios significativos en la dimensión interna de los ambientes colaborativos, e.g., el estado de los proyectos, las políticas organizacionales, actividades en progreso, y los roles de usuario. Las fuentes que activan eventos contextuales lógicos son aplicaciones de software, bases de datos, repositorios de información y conocimiento, otros sistemas, y las acciones del usuario.

En las organizaciones es común el uso de sistemas de información, que son fuentes potenciales de eventos lógicos. Sin embargo, alguna de esta información necesita ser procesada a través de otras aplicaciones, e.g., un administrador de alarmas de tiempo puede ser responsable de la activación del evento "*End of working day*" a las 18:00 horas, o es generada por estas aplicaciones, e.g., un editor de texto colaborativo puede generar el evento contextual lógico " `$COLLABORATOR` *logged in*".

La Tabla 5.5 muestra algunos sistemas de información y aplicaciones junto a ejemplos de los eventos contextuales que éstos pueden producir.

Cuadro 5.5: Ejemplos de eventos contextuales lógicos

Fuente	Eventos contextuales lógicos
Human Resource Management System	The `$COLLABORATOR` has been enrolled, The skills of `$COLLABORATOR` has been updated, The seniority of `$COLLABORATOR` has been updated, *y* The vacation of `$COLLABORATOR` have started.
Repository of organizational rules and policies	Lunch time has started/finished, `$COLLABORATOR` has been late `three` times, *y* The `$POLICY` has been included.
Project Management System	The priority of the `$PROJECT` has been updated, The delivery date of the `$PROJECT` has been reached, *y* The `$COLLABORATOR` has been asigned to an activity.

5.2.3. Funcionamiento del perceptor

El funcionamiento de un perceptor comprende las etapas siguientes: 1) inicialización, 2) detección de cambio de estado, 3) preprocesamiento, y 4) notificación al servicio de contexto. Cada etapa consiste en lo siguiente:

1. **Inicialización**. Se cargan en memoria la *tabla de sensores* y el *diccionario de vocabulario contextual*. Para cada sensor registrado, se verifica su disponibilidad y se determina su estado actual. Además, se inicializa el modelo de estado.

2. **Detección de cambio de estado**. Cada cierto intervalo de tiempo, se compara la medición actual del sensor con los límites de los umbrales para la variable física que vigilan. Cuando el estado de al menos un sensor cambia, se procede a la etapa de *preprocesamiento*.

3. **Preprocesamiento**. Si hay más de un sensor para la variable contextual que cambió de estado, la información de todos los sensores que vigilan tal variable se procesa (*fusión de sensores* y *calidad de contexto*) para determinar si se ha producido un evento contextual.

4. **Notificación al servicio de contexto**. Si el evento contextual es legítimo, se envía una notificación al servicio de contexto, ésta debe incluir el *nombre* y los *parámetros asociados*.

En la etapa de **preprocesamiento** se propone el algoritmo mostrado en la Tabla 5.6 para realizar la *fusión de sensores*, y la *calidad del contexto*:

Cuadro 5.6: Algoritmo para el preprocesamiento de eventos contextuales

Requiere: el conjunto de variables contextuales V cuyo estado cambió

```
1:  /* Let V the set of contextual variables whose stated changed. */
2:  for every contextual variable v in V:
3:     /* Let v one variable whose state changed to the new state t and S the set of
       sensors that monitor v */
4:     if |S| = 1 do
5:        Notify the produced contextual event (contextual variable, current state and
          perceptor id) to the context service.
6:     else
7:        counter = 0
8:        for each sensor s in S
9:           if s has state t
10:             counter++
11:       if counter > ⌊|S|/2⌋
12:          Notify the produced contextual event (contextual variable, current state
             and perceptor id) to the context service
```

5.3. Detección de la situación

Los eventos contextuales pueden cambiar el estado de las situaciones, i.e., de *activa* a *inactiva* y viceversa.

Una **situación** es una configuración del modelo de estado propicia para realizar una adaptación y cuenta con las siguientes propiedades:

1. **Nombre**: es un identificador descriptivo y único que distingue la situación de las otras.
2. **Tipo**: indica si la situación es *transitoria* o *permanente*.
3. **Entidad relacionada**: asocia a la entidad afectada por el estado de la situación, e.g., un usuario, un proyecto, o un lugar.
4. **Escuchas**: es una lista de las entidades que están interesadas en el cambio de estado de la situación (*activa* o *inactiva*).
5. **Eventos contextuales asociados**: es una lista de los eventos contextuales que pueden cambiar el estado de la situación (*activa* o *inactiva*).
6. **Condiciones**: es una lista de las restricciones que tienen que ser satisfechas para activar o desactivar la situación.

Con base en su continuidad, las situaciones se clasifican en *situaciones transitorias* y *situaciones permanentes*.

5.3.1. Situación transitoria

Una situación es **transitoria** si sólo es importante cuando sus escuchas están presentes.

El algoritmo mostrado en la Tabla 5.8 muestra el proceso general para determinar si una situación transitoria está presente.

Para ilustrar, considérese la situación transitoria "*Change of location*", la cual puede activarse por los eventos contextuales "*A collaborator enters a location*" o "*A collaborator leaves a location*". Esta situación es explotada por mecanismos de presencia de grupo para informar de la ubicación de cada usuario de forma continua. Por ello, esta situación sólo es importante cuando está presente para reflejar un cambio en las situaciones mostradas en la G.U.I.

Cuadro 5.8: Algoritmo para la evaluación de la presencia de una situación transitoria

Requiere: evento contextual activado e

```
1:  /* Let be s a transient situation associated with e .*/
2:  if no conditions then
3:      A ← true /* the situation s is present from the contextual event e */
4:  else
5:      A ← result of evaluating the conditions of the situation s
6:  if A is true then
7:      notify s to its listeners
```

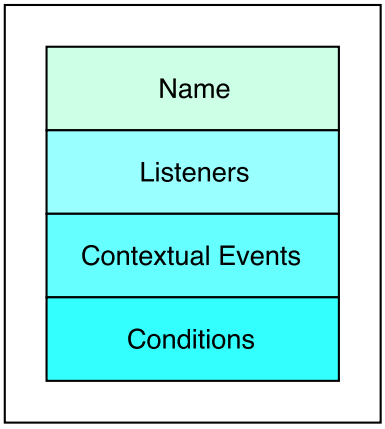


Figura 5.3: Propiedades de una situación transitoria

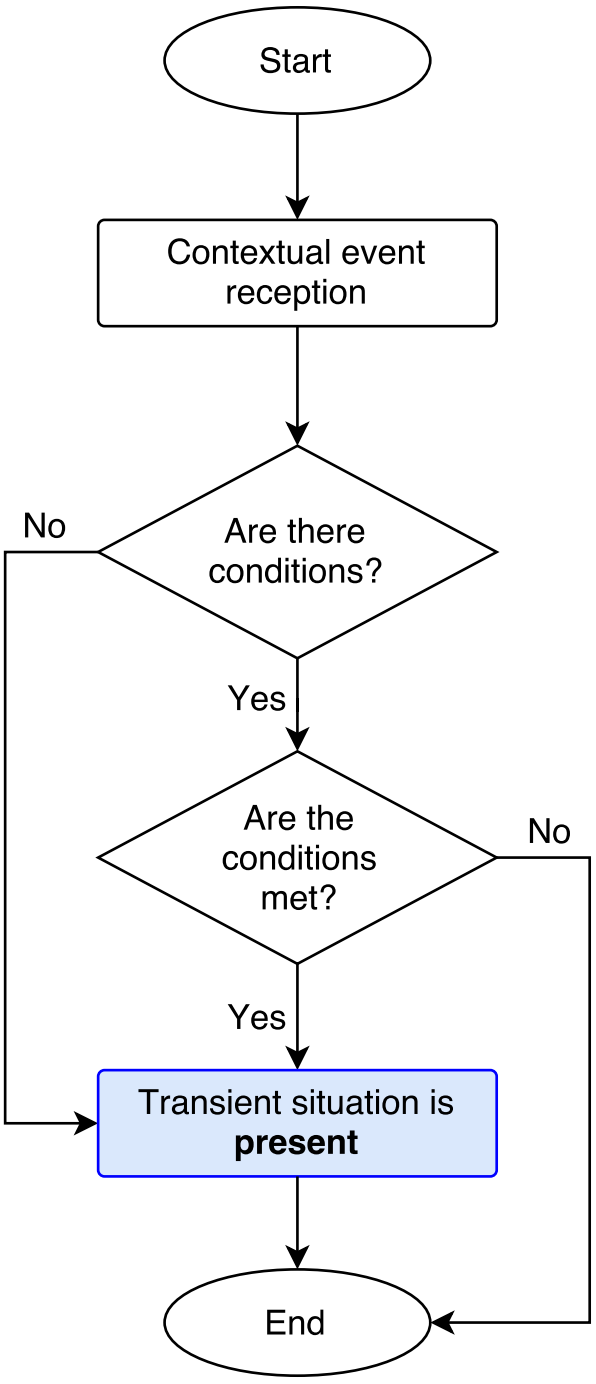


Figura 5.4: Estados de una situación transitoria

5.3.2. Situación permanente

Una situación es **permanente** si siempre está presente y su estado, *activa* o *inactiva*, dependerá de los eventos activados y las condiciones satisfechas.

El algoritmo mostrado en la Tabla 5.9 muestra el proceso general para determinar el estado de una situación permanente.

Cuadro 5.9: Algoritmo para la determinación del estado de una situación permanente

Requiere: evento contextual activado e

1: /* Let be s a permanent situation associated with e such that $s \in Ss$ and $T = active, inactive$ be the set of states for the permanent situations and f be a partial function from S to T. */
2: $A \leftarrow$ result of evaluating the conditions of the situation s
3: A is $true$ and $f(s)$ is $inactive$ **then**
4: $f(s) \leftarrow active$ /* the situation s is activated*/
5: notify s and its state $f(s)$ to its listeners
6: **else**
7: **if** A is $false$ and $f(s)$ is $active$ **then**
8: $f(s) \leftarrow inactive$ /* the situation s is desactivated */
9: notify s and its state $f(s)$ to its listeners

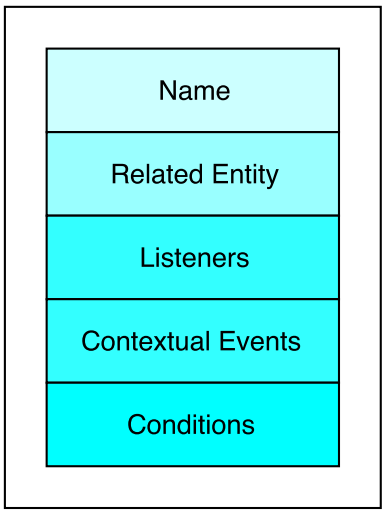


Figura 5.5: Propiedades de una situación permanente

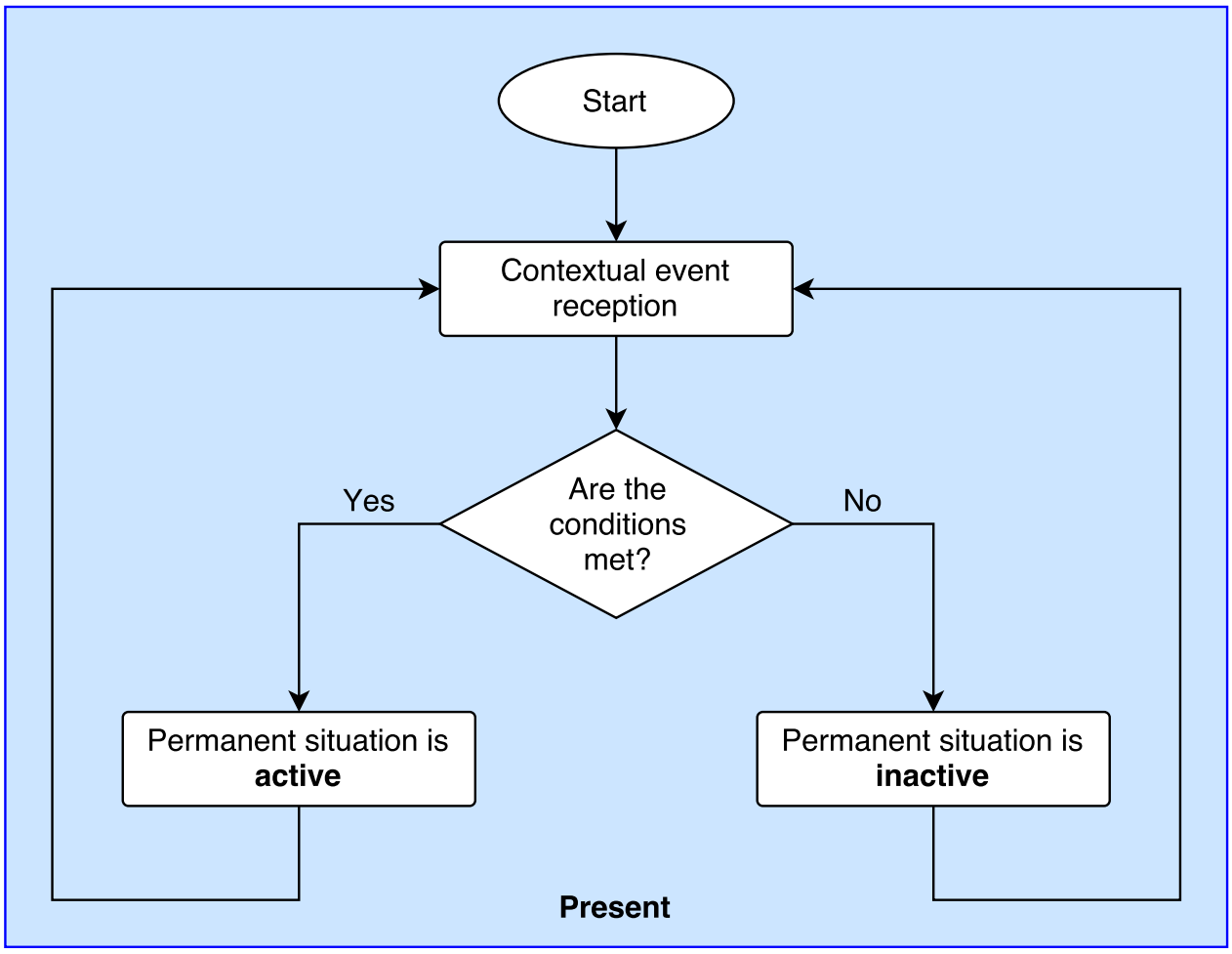


Figura 5.6: Estados de una situación permanente

Para ilustrar, considérese la situación permanente "*Collaborative edition of a document*", que es de interés para los colaboradores que trabajan en un determinado reporte (veáse la Tabla 5.10). Esta situación puede ser activada o desactivada dependiendo de si las condiciones 1) "*The number of collaborators logged in the collaborative editor is greater than one*", y 2) "*The number of collaborators working on the same document is greater than one*" se satisfacen o no.

Cuadro 5.10: Propiedades de la situación "*Collaborative edition of a document*"

Name	Collaborative edition of a document
Type	Permanent
Related entity	Document
Listeners	1) Collaborator logged in the collaborative editor.
Contextual events	1) A collaborator opens a document, *and* 2) A collaborator closes a document.
Conditions	*Active*: 1) The number of collaborators logged in the collaborative editor is greater than one *and* 2) the number of collaborators working on the same document is greater than one.

El siguiente escenario tiene el propósito de ilustrar la manera en la cual la arquitectura conceptual propuesta realiza la detección de la situación:

> {*Alice*} inicia sesión en el editor colaborativo {*Collab*} y abre el documento {*Monthly report*}. Esto activa el evento contextual *"The collaborator* {*Alice*}*" opens the document* {*Monthly report*}, se evalúa la primera condición, *"The number of collaboratos logged in* {*Collab*} *is greater than one"* y dado que no se satisface no se activa la situación "Collaborative edition of the document {*Monthly report*}". Tiempo después, {Jenny} inicia sesión en {*Collab*} y abre el documento {*Prototype presentation*}, entonces se activa el evento contextual *"The collaborator* {*Jenny*} *opens the document* {*Monthly report*}*"* y se evalúa la condición *"The number of collaborators logged in* {*Collab*} *is greater than one"* y dado que se satisface, se evalúa la siguiente condición, *"The number of collaborators working on the document* {*Monthly report*} *is greater than one"*, y al también satisfacerse, se activa la situación "Collaborative edition of the document {*Monthly report*}".
>
> Cuando {*Alice*} concluye la sección que le fue asignada y cierra el documento {*Monthly report*} se activa el evento contextual *"The collaborator* {*Alice*} *closes the document* {*Monthly report*}*"*, se evalúan las condiciones y al no ser satisfechas la situación "Collaborative edition of the document *Monthly report*" es desacti-

vada. A continuación, {*Alice*} abre el documento {*Prototype presentation*}, así se produce el evento contextual *"The collaborator* {*Alice*} *opens the document* {*Prototype presentation*}*"*, se evalúa la primera condición, *"The number of collaborator logged in* {Collab} *is greater than one"*, y dado que se satisface, se evalúa la siguiente condición, *"The number of collaborators working on the document* {*Prototype presentation*} *is greater than one"*, y dado que no se satisface, no se activa la situación "Collaborative edition of the document {*Prototype presentation*}", paralelamente, se evalúa la condición, *"The number of collaborators working on the document* {*Monthly report*}*"*, y dado que no se satisface, la situación "*Collaborative edition of the document* {*Monthly report*}" es desactivada.

Después, {*Alice*} cierra el documento {*Prototype presentation*}, entonces se activa el evento contextual *"The collaborator* {*Alice*} *closes the document* {*Prototype presentation*}*"*, se evalúa la primera condición, *"The number of collaborators logged in* {*Collab*} *is greater than one"* y dado que se satisface, se evalúa la segunda condición, *"The number of collaborators working on the document* {*Prototype presentation*} *is greater than one"*, y dado que no se satisface, no se activa la situación "Collaborative edition of the document {*Prototype presentation*}".

La Figura 5.7 muestra el diagrama de flujo para determinar el estado de la situación permanente *Collaborative edition of a document.*

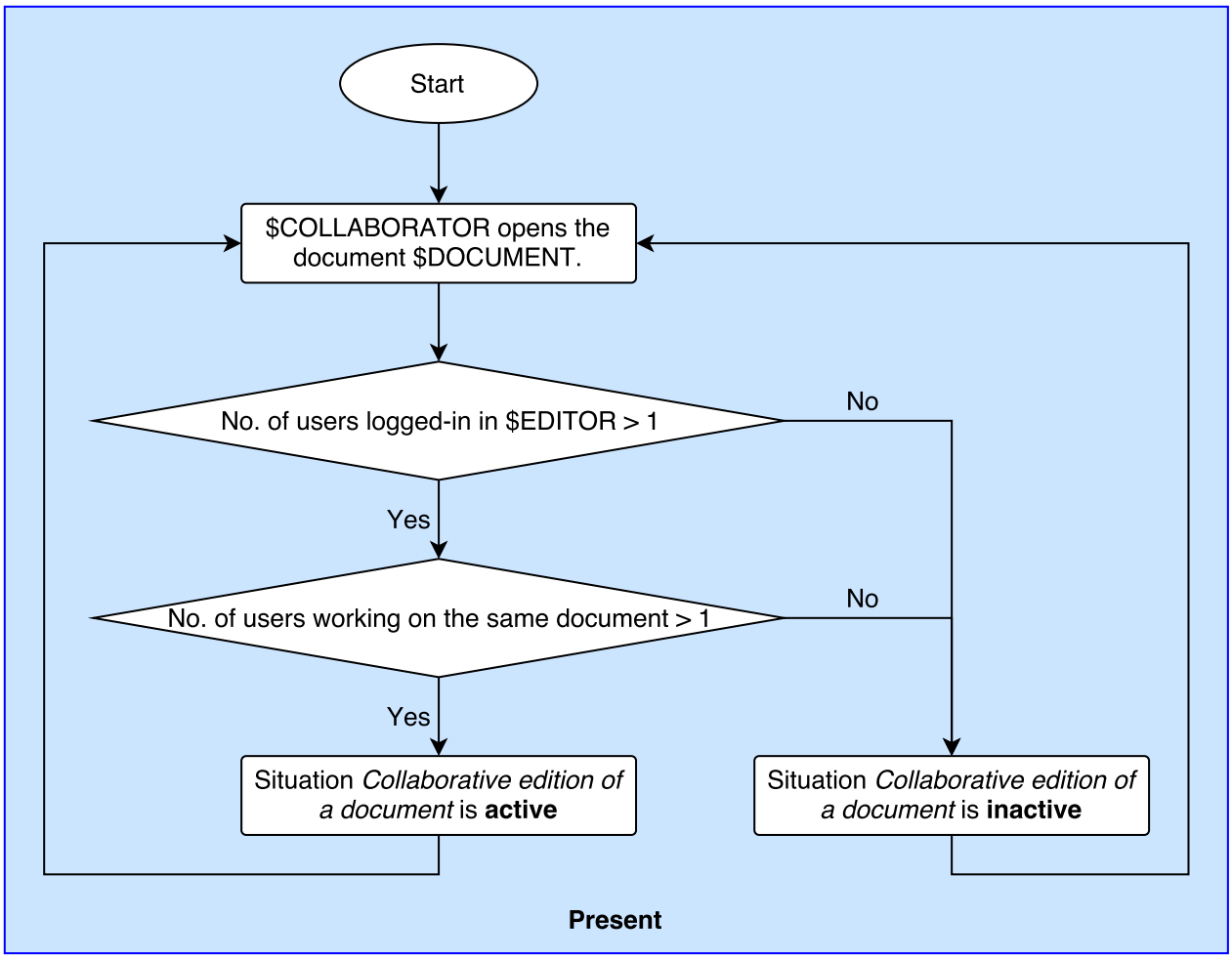


Figura 5.7: Determinación del estado de la situación "*Collaborative edition of a document*"

5.3.3. Proceso general para realizar la fase de detección de situación

El algoritmo mostrado en la Tabla 5.11 muestra el proceso general para realizar la fase de detección de situación.

Sea C el conjunto de eventos contextuales y E el conjunto de eventos contextuales detectados, tal que $E \in C$. Para cada evento contextual $e \in E$ se analiza cada una de las situaciones asociadas: 1) si la situación es *transitoria* se evaluarán sus condiciones (si

existen) y si se satisfacen, sus escuchas serán notificados, y 2) si la situación es *permanente*, se evaluarán sus condiciones para determinar su estado (*activa* e *inactiva*), si éste cambia, sus escuchas serán notificados.

Cuadro 5.11: Método para realizar la fase de detección de situaciones

Requiere: evento contextual activado e

```
1:  /* Let be S the set of situations, S_e the set of situations associated with e such that
    S_e ⊂ S, T = {active, inactive} be the set of states for the permanent situations and
    f be a partial function from S to T. */
2:  for each situation s in S_e do
3:      if s is a transient situation then
4:          if no conditions then
5:              A ← true /* the situation s is present from the contextual event e */
6:          else
7:              A ← result of evaluating the conditions of the situation s
8:          if A is true then
9:              notify s to its listeners
10:     else
11:         /* s is a permanent situation */
12:         A ← result of evaluating the conditions of the situation s
13:         if A is true and f(s) is inactive then
14:             f(s) ← active /* the situation s is activated */
15:             notify s and its state f(s) to its listeners
16:         else
17:             if A is false and f(s) is active then
18:                 f(s) ← inactive /* the situations s is desactivated */
19:                 notify s and its state f(s) to its listeners
```

5.4. Comunicación de la situación

Los cambios en las situaciones se notifican a las correspondientes entidades, bajo la premisa de diseño:

> "Los usuarios no están interesados en cada cambio que ocurre en el estado de las situaciones, sólo en aquellas que los afectan directamente".

A partir de esta premisa, se utiliza el patrón de mensajería *Publicación/Suscripción* [Eugster et al., 2003] para notificar de la presencia de situaciones transitorias y los cambios de estado en las situaciones permanentes. Las entidades bajo el rol de "Editor" emiten mensajes, mientras que aquellas bajo el rol de "Suscriptor" reciben los mensajes. No obstante, una entidad no está limitada a un sólo rol, puede tener ambos roles a la vez.

Los *editores* no envían los mensajes directamente a los *suscriptores*. Cada mensaje se asocia con al menos un tema, entonces, los *editores* emiten mensajes de cierto temas, ignorando los *suscriptores* presentes, estos últimos, expresan su interés en ciertos temas, sin el conocimiento del publicador, y consecuentemente son notificados de cualquier mensaje que coincida con su interés.

Se requiere de un punto intermedio entre *editores* y *suscriptores*, que guarde una relación entre los temas registrados y sus respectivos *suscriptores* interesados. Este componente de software es llamado el *agente de Publicación/Suscripción*. Las propiedades de un *agente de Publicación/Suscripción* son las siguientes:

1. **Puerto de editores**. Es una interfaz de software que permanece a la espera de mensajes por parte de entidades bajo el rol de *Editor*. En Java, puede implementarse mediante un servidor multi-hilo.

2. **Puerto de suscriptores**. Es una interfaz de software que permanece a la espera de peticiones de suscripción por parte de entidades bajo el rol de *Suscriptor*. En Java, puede implementarse mediante un servidor multi-hilo.

3. **Tabla de relación tema/consumidor**. Es una estructura dinámica, que asocia cada tema con una lista de los *suscriptores* interesados.

Además, un *agente de Publicación/Suscripción* puede asumir ambos roles, i.e., emitir y recibir

mensajes procedentes hacia o desde otros *agentes de Publicación/Suscripción*. Este enfoque confiere la escalabilidad de controlar varios *agentes de Publicación/Suscripción* independientemente de su implementación y es útil en escenarios altamente distribuidos, que en general, involucran *agentes de Publicación/Suscripción* múltiples ejecutándose en dispositivos diferentes (véase la Figura 5.8).

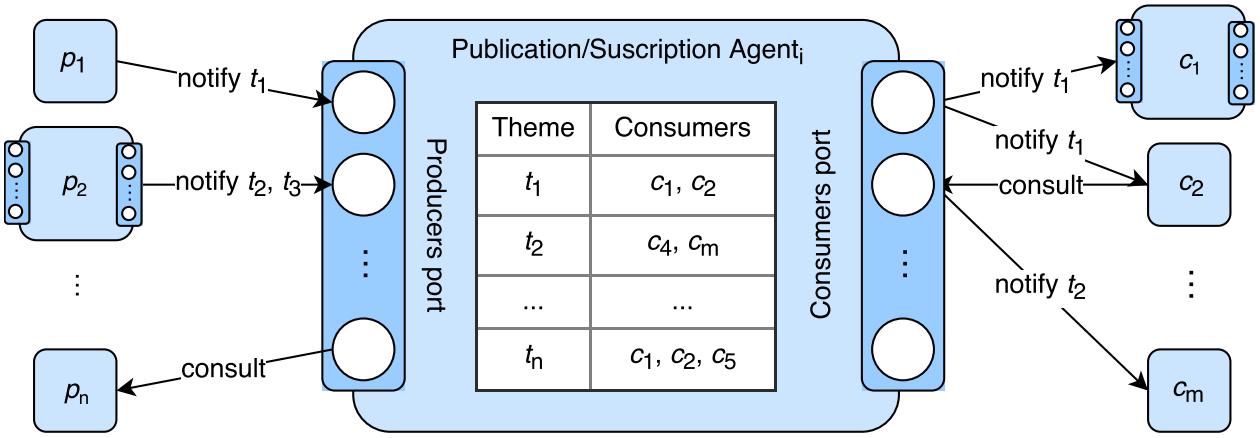


Figura 5.8: Un nodo de un sistema colaborativo que actúa como un agente de Publicación/Suscripción

El algoritmo mostrado en la Tabla 5.12 muestra el funcionamiento general de un *agente de Publicación/Suscripción*.

Cuadro 5.12: Algoritmo del funcionamiento general de un agente de *Publicación/Suscripción*

Requiere: -

1: The *Publication/Subscription agent* registers itself to the *List of Publisher/Subscriptor agents*, so that it is visible to *Publishers* and the *Subscribers*.

2: The *Publication/Subscription agent* initialize its own *associative table Topic/Subscriptors* and remains awainting messages, from the *Publishers*, or requests, from the *Subscribers*.

3: When it receives a message from a *Publisher*, its associated topic is searched in the *associative table Topic/Subscriptors* and if there is, the message is forwarded to each of the interested *Subscribers*.

4: When it receives a request from a *Subscriptor*, its associated topic is searched in the *associative table Topic/Subscriptors* and if there is, the *Subscriptor* is registered as interested in such topic; on the contrary, the topic and the *Subscriptor* are registered.

5.5. Adaptación del sistema

Cuando está presente una situación transitoria o cambia el estado de una situación permanente, el sistema colaborativo tiene que adaptarse de una forma conveniente. Para ello, la arquitectura conceptual utiliza el patrón de diseño *Observer* [Gamma et al., 2000].

El patrón *Observer* es un patrón de comportamiento, i.e., describe las estructuras de relación entre objetos y/o clases y los esquemas de comunicación entre ellos, clasificados en función de que trabajen con clases, o, u objetos, e.g., observador, visitante, cadena de responsabilidad, entre otros. Para ello, define una dependencia uno a muchos, (1:M), entre objetos y permite notificar automáticamente los cambios del objeto observado, el *observador concreto*, a todos los objetos dependientes, i.e., los *observadores concretos* (véase la Figura5.9).

El *observador concreto*, el contexto de uso, mantiene el estado de interés, las situaciones permanentes activas, para los observadores concretos. Notifica a cada uno de ellos de sus cambios de estado, i.e., de la activación de situaciones transitorias, o de la activación/desactivación de una situación permanente, mediante el envío de señales en el objeto llamado *Situation State*. Para ello, hace uso de la interfaz *sujeto*, que permite a los *observadores* suscribirse (*subscribe*) o desuscribirse (*unsubscribe*) mediante una referencia de sí mismo, de tal forma que mantiene una lista de las referencias a todos sus observadores.

El *objeto observador concreto*, e.g., un elemento de una interfaz gráfica de usuario (*G.U.I.*), mantiene una referencia al *objeto observado concreto* e implementa la interfaz de actualización, que le permite obtener datos sobre el cambio informado. Los observadores sólo se

actualizarán si reciben una señal adecuada, si es así, devolverán un objeto llamado *Observer State*. Los objetos *Observer State* permitirán conocer a los demás objetos si su actualización fue exitosa o no.

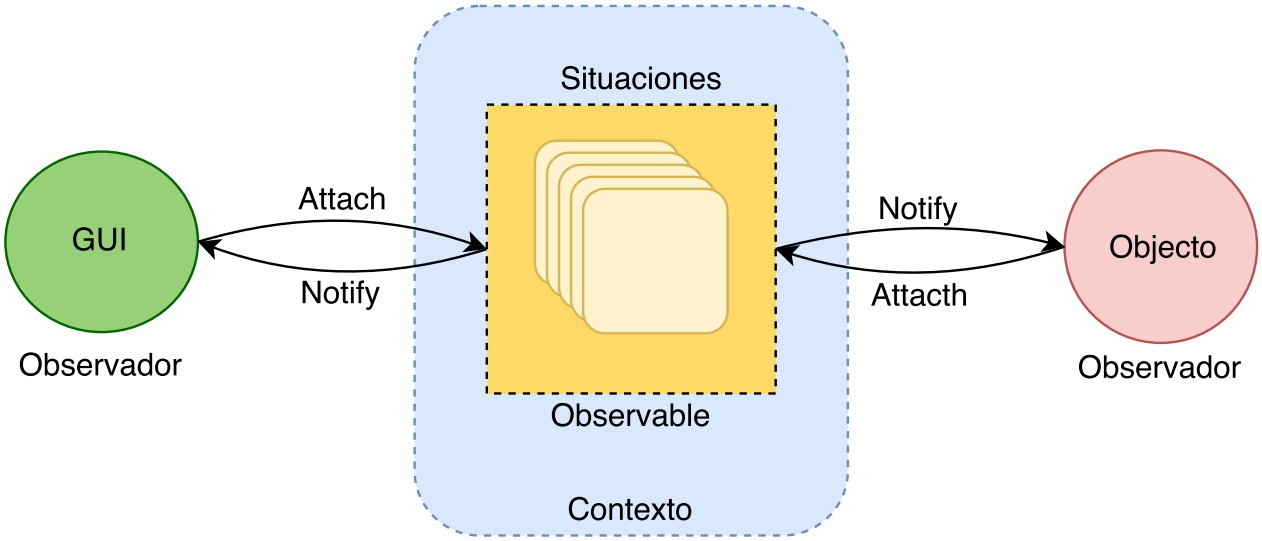


Figura 5.9: Patrón de diseño *Obsever*

Este patrón es útil para escenario que manejen transacciones, tal como en el caso de sistemas colaborativos, en donde múltiples instancias de la interfaz gráfica de usuario, u otros objetos, (*observadores concretos*) no están ligadas al mismo conjunto de situaciones del contexto (*sujetos concretos*), e inclusive, para aquellos objetos sujetos a un mismo tipo de situaciones las adaptaciones pueden variar dependiendo de las preferencias de cada usuario.

Finalmente, la adaptación de cada objeto se realizará mediante las llamadas *reglas de adaptación*, que dictaminan las modificaciones que los *observadores concretos* deben realizar. Se proponen reglas estáticas, de la forma **if** *detected situation* **then** *planned adaptation*. La Tabla 5.13 muestra un ejemplo de estas reglas.

Cuadro 5.13: Reglas de adaptación estáticas

```
1:  if End of the work day detected then
2:    show message "The work day has over"
3:    offer the collaborators save their work
```

Debido a que la forma específica en la cual el sistema colaborativo debe adaptarse está determinada por el dominio de aplicación, es útil realizar consultas a las ontologías que representan los elementos del dominio de aplicación, para inferir nueva información. Para ello, pueden realizarse consultas SPARQL. La Tabla 5.14 muestra una consulta SPARQL para obtener el nombre de todos los recursos humanos.

Cuadro 5.14: Consulta SPARQL para obtener el nombre de todos los recursos humanos

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX h: <http://www.owl-ontologies.com/humanR.owl#>
SELECT ?Z
WHERE {
  {?Z rdf:type h:HumanResource . }
  ?Z h:nameRH ?N .
};
```

5.6. Implementación

Cada fase de la arquitectura conceptual puede ser implementada de varias maneras, i.e., utilizando una variedad de lenguajes de programación, estructuras de datos, herramientas, y tecnologías existentes, o bien, diseñándolos en su totalidad. En esta tesis, la implementación utiliza el lenguaje Java porque, junto a C y C++, es uno de los lenguajes de programación más populares.

La Figura 5.10 muestra los componentes de software de la implementación de la arquitectura conceptual.

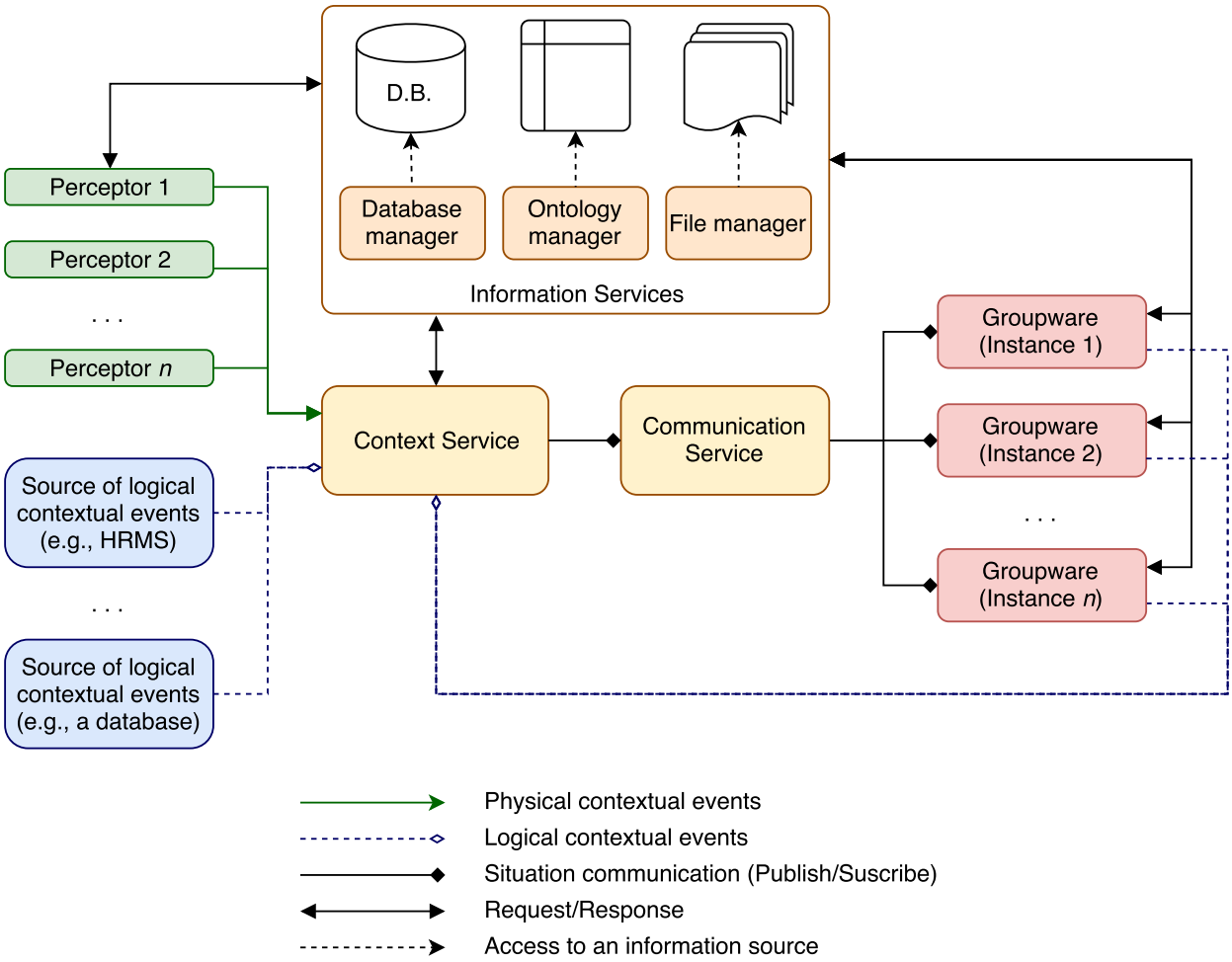


Figura 5.10: Componentes de software de la implementación de la arquitectura conceptual

A continuación se ofrece una síntesis de los componentes de software de las etapas anteriores:

1. **Perceptor**. Es un componente de software que cada cierto intervalo de tiempo realiza una medición de cada uno de sus sensores asociados y determina si se producido un cambio de estado. Entonces, determina si se ha producido un evento contextual. Hace uso del diccionario del vocabulario ontológico,

2. **Servicio de contexto**. Es un servidor, que ejecuta las etapas de *Detección de situación* y *Comunicación de la situación*. Para ello, permanece a la espera de eventos contextuales, los procesa y si se ha presentado una situación transitoria o ha ocurrido un cambio en el estado de una situación permanente, utiliza el patrón de mensajería *Publicación/Suscripción* para notificar a los nodos interesados,

3. **Servicio de comunicación**. Es un servidor, que actúa como un intermediario que recibe y redirige mensajes. Utiliza el patrón de mensajería *Publicación/Suscripción* para notificar de los cambios en el contexto (c.f. Sección 5.4),

4. **Sistema colaborativo**. Es el conjunto de herramientas que ofrecen a los usuarios mecanismos que les permiten trabajar en un proyecto común. Tiene la capacidad de adaptarse dependiendo de las notificaciones que reciba, y

5. **Servicios de información**. Es el conjunto de métodos que acceden a diversas fuentes de información para manipular datos. Por ejemplo, a través de consultas, en el caso de bases de datos, se ocupa el lenguaje SQL, y para las ontologías, se ocupa el lenguaje SPARQL.

5.6.1. Estructuras de datos propuestas

Esta sección expone las estructuras de datos desarrolladas desde cero en el lenguaje Java para los elementos software *perceptor* y *servicio de contexto*.

Para el **perceptor**:

1. **Tabla de sensores**

Se compone de una llave única, que corresponde a la variable contextual vigilada, y una lista de los identificadores de los sensores que vigilan dicha variable. `Sensor Table` es una instancia de la estructura dinámica HashMap, puesto que no acepta duplicados ni valores

nulos (véase la Tabla 5.15).

Cuadro 5.15: Estructura `SensorTable`

```
1: Map<String, ArrayList<Sensor>> SensorTable = new HashMap<>();
```

2. **Diccionario de vocabulario ontológico**

Se compone de parejas formadas por una llave única, que corresponde al nombre de una variable contextual y un objeto que representa los estados y sus umbrales. `Ontological-VocabularyDictionary` es una instancia de la estructura dinámica HashMap, puesto que no aceptar duplicados ni valores nulos (véase la Tabla 5.16).

Cuadro 5.16: Estructura `OntologicalVocabularyDictionary`

```
 1: public class State{
 2:   public String name;
 3:   public String unity;
 4:   public long lowerLimit;
 5:   public long upperLimit;
 6:
 7:   /* Constructor */
 8: }
 9:
10: ...
11:
12: Map<String, ArrayList<State>> OntologicalVocabularyDictionary
    = new HashMap<>();
```

3. **Evento contextual**

Se compone del *nombre* y sus *parámetros asociados*. `contextualEvent` es una instancia de

la estructura dinámica llamada LinkedHashMap. El elemento con la llave `Name` corresponde al nombre del evento contextual y los restantes a los parámetros asociados (veáse la Tabla 5.17).

Cuadro 5.17: Estructura `contextualEvent`

```
1: Map<String, String> contextualEvent = new HashMap<>();
2: map.put("NAME", "VALUE"); // Nombre del evento contextual
```

Para el **servicio de contexto**:

1. **Situación**

El uso de objetos es ideal para la representación de las situaciones, puesto que las propiedades pueden ser modeladas como atributos de una clase. Las situaciones se modelan como instancias de la clase *Situation*, que se muestra en la Tabla 5.18.

Cuadro 5.18: Clase *Situation*

```
 1: public class Situation
 2: {
 3:   private String name;
 4:   private boolean isTransitory;
 5:   private Boolean isActive;
 6:   private String relatedEntity;
 7:   private List<String> listeners = new ArrayList<>();
 8:   private List<String> contextualEvents = new ArrayList<>();
 9:   private List<String> conditions = new ArrayList<>();
10:
11:   public boolean CheckConditions()
12:   {
13:     boolean result = true;
14:     for( String condition : conditions )
15:       result = result && Evaluate(condition);
16:     return result;
17:   }
```

```
18:
19:     /* Permanent situation constructor */
20:     public Situation(
21:       String name,
22:       Boolean isActive,
23:       String relatedEntity,
24:       List<String> listeners,
25:       List<String> contextualEvents,
26:       List<Boolean> conditions
27:     ){
28:       this.name = name;
29:       this.isTransitory = false;
30:       this.isActive = isActive;
31:       this.relatedEntities = relatedEntities;
32:       this.listeners = listeners;
33:       this.contextualEvents = contextualEvents;
34:       this.conditions = conditions;
35:     }
36:
37:     /* Transient situation constructor */
38:     public Situation(
39:       String name,
40:       List<String> listeners,
41:       List<String> contextualEvents,
42:       List<Boolean> conditions
43:     ){
44:       this.name = name;
45:       this.isTransitory = true;
46:       this.isActive = null;
47:       this.relatedEntity = null;
48:       this.listeners = listeners;
49:       this.contextualEvents = contextualEvents;
50:       this.conditions = conditions;
51:     }
52:
53:     /* GET and SET methods */
54:  }
```

A continuación se explican cada uno de los atributos usados:

- `name`. Corresponde al *nombre* de la situación.

- `isTransitory`. Corresponde al *tipo* de la situación, si su valor es *true* es transitoria, de lo contrario, es permanente.

- `isActive`. En el caso de las situaciones permanentes, corresponde a su estado, el valor *true* representa al estado *activa*; y, *false* al estado *inactiva*.

- `relatedEntity`. Corresponde a la *entidad relacionada* de la situación.

- `listeners`. Es un arreglo dinámico que almacena los identificadores que corresponden a los *escuchas* de la situación.

- `contextualEvents`. Es un arreglo dinámico que almacena los identificadores que corresponden a los *eventos contextuales* que pueden conducir a la activación de la situación, si es transitoria, o a un cambio de su estado, si es permanente.

- `conditions`. Es un arreglo dinámico que almacena los identificadores que representan las *condiciones* asociadas a la situación. Para verificar que todas las condiciones sean satisfechas, se emplea el método *CheckConditions*, que aplica la operación lógica AND entre el resultado de evaluar cada uno sus elemento, con el método `Evaluate`.

2. **Listas de situaciones transitorias y de situaciones permanentes**

Las estructuras *transientSituations* y *permanentSituations* son listas enlazadas que permiten al servicio de contexto tener en memoria cada una de las situación transitorias y permanentes, respectivamente. Para representar el estado de cada situación permanente se utiliza el atributo `isActive` (véase la Tabla 5.19).

Cuadro 5.19: Estructuras `transientSituations` y `permanentSituations`

```
1:  /* List of transient situations */
2:  ArrayList<Situation> transientSituations = new ArrayList<>();
3:
4:  /* List of permanent situations */
5:  ArrayList<Situation> permanentSituations = new ArrayList<>();
```

5.6.2. Etapas del servicio de contexto

A continuación se detallan los pasos del funcionamiento general del servidor:

1. **Puesta en marcha de servicios**

Inicia los servicios básicos de la arquitectura, i.e., los servicios de contexto, de comunicación, y aquellos que acceden a fuentes de información del dominio de la aplicación (e.g., gestores de bases de datos, de ontologías, y de archivos).

En primer lugar, se inician los servicios que acceden a fuentes de información. Entonces, el *servicio de contexto* lee todas las situaciones registradas en el sistema colaborativo y las añade a las estructuras `transientSituations` y `permanentSituation`, según corresponda. Para las situaciones permanentes se evalúan sus condiciones para determinar su estado, i.e., si están activas o inactivas.

Después, se inicia el *servicio de comunicación* y el servicio de contexto se registra como una entidad bajo el rol Editor. Los temas de los mensajes que publicará dependen de la aplicación que se esté desarrollando.

Finalmente, el servicio de contexto permanece a la espera para recibir notificaciones sobre eventos contextuales y el servicio de comunicación, de peticiones de entidades suscrip-

toras o nuevas editoras. Particularmente, se utilizan sockets para para comunicar eventos contextuales.

2. **Detección de situaciones**

Una vez que el *servicio de contexto* recibe un *evento contextual*, se analizan cada una de las situaciones asociadas. Si ésta es una situación transitoria y no tiene condiciones asociadas, simplemente se notifica su presencia a sus escuchas, mediante el *servicio de comunicación*. Por el contrario, si tiene condiciones asociadas, para determinar si tal situación transitoria está presente, o si se requiere cambiar el estado de alguna situación permanente, se evalúa su conjunto asociado de condiciones. Para ello se realizan los siguientes pasos:

i. **Completación de parámetros**. Se determina si todos los parámetros requeridos para evaluar las condiciones son suministrados por la activación de eventos contextuales. En caso negativo, se obtienen mediante el uso de diversos métodos como consultas al estado actual de la aplicación, e.g., si se requiere el número de identificación de un colaborador, *$COLLABORATOR*, puede usarse el método `$getCollaboratorId` para obtenerla, o consultas al dominio de la aplicación, como a una base de datos o a una ontología, e.g., si se requiere el número de identificación de un colaborador, almacenado en la ontología *humanR*, puede realizarse la siguiente consulta en SPARQL:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX h: <http://www.owl-ontologies.com/humanR.owl#>
SELECT ?Z ?N ?U
WHERE {
  {?Z rdf:type h:HumanResource . }
  ?Z h:nameRH ?N .
  ?Z h:idNumber ?U .
  FILTER regex(?N, "$COLLABORATOR", "$i")
};
```

II. **Evaluación de condiciones**. Pueden programarse como métodos booleanos, e.g., la situación "*Collaborative edition of document*" puede ocupar los métodos:

- `workingOnSameApplication(List<User> users, Application a)`
- `workingOnSameDocument(List<User> users, Document d)`

El uso de métodos booleanos y operadores lógicos resulta intuitivo y permite la reutilización de código cuando se definen situaciones nuevas. Particularmente, la clase `Situation` (véase Tabla 5.18) almacena las condiciones que deben ser cumplidas para su activación (en el caso de situaciones transitorias y permanentes) o su desactivación (sólo para situaciones permanentes) en el atributo `conditions`, una lista que almacena los identificadores de las condiciones que deben ser evaluadas mediante el método `Evaluate`.

Una situación transitoria estará presente, o una situación permanente estará activada, cuando todas las condiciones de la lista `conditions` se satisfagan, i.e., el resultado de la evaluación de todas ellas sea `true`. De forma análoga, una situación transitoria no estará presente, o una situación permanente estará inactiva, cuando al menos una de las condiciones de la lista `conditions` no se satisfaga, i.e., el resultado de al menos una sea `false`.

El método `CheckConditions` es el encargado de verificar si todas las condiciones de una situación se satisfacen (véase la Tabla 5.20).

Cuadro 5.20: Método `CheckConditions`

```
1:  public boolean CheckConditions()
2:  {
3:    boolean result = true;
4:    for( String condition : conditions )
5:      result = result && Evaluate(condition);
6:    return result;
7:  }
```

El método `Evaluate` evalúa si una condición, dado su identificador, se cumple o no. El identificador se busca en una estructura de control `switch`, si éste es encontrado se ejecutan una serie de reglas de la forma *if - then* y se devuelve su resultado, i.e., `false` o `true`. De lo contrario, se devuelve el valor `false` (veáse la Tabla 5.21).

Las instrucciones *if - then* pueden hacer uso de diversos métodos como consultas al estado actual y el domino de la aplicación (e.g., métodos preestablecidos, una base de datos, una ontología) para aumentar su versatilidad.

Cuadro 5.21: Método `Evaluate`

```
1:  public boolean Evaluate(String conditionId)
2:  {
3:    switch( conditionId )
4:    {
5:      case 'conditionId1':
6:        /* if-then instructions */
7:        /* return result of evaluating code above */
8:        break;
9:
10:     case 'conditionId2':
11:       /* if-then instructions */
12:       /* return result of evaluating code above */
13:       break;
14:
15:     ...
16:
17:     case 'conditionIdn':
```

```
18:         /* if-then instructions */
19:         /* return result of evaluating code above */
20:         break;
21:
22:       default:
23:         return false;
24:
25:     }
26:   }
```

3. **Comunicación de la situación**

El *servicio de comunicación* es una implementación del patrón de mensajería *Publicación /Suscripción* para notificar de los cambios en el contexto. Si bien existen formas diferentes de implementarlo, a continuación se muestra un sencillo ejemplo para su implementación, compuesto por el paquete `patronPublishSubscribe`, y las clases `Message`, `Publisher`, `Subscriptor`, y `PublisherSubscriptorAgent`.

La clase `Message` representa a los mensajes transmitidos y se compone por las estructuras `topic` y `content`, que representan el tema asociado y el contenido del mensaje, respectivamente (veáse la Tabla 5.22).

Cuadro 5.22: Clase `Message`

```
 1: package patronPublishSubscribe;
 2:
 3: public class Message
 4: {
 5:   private String topic;
 6:   private String content;
 7:
 8:   /* Constructores */
 9:   public Message(){
10:   }
11:
12:   public Message(String topic, String content) {
```

```
13:     this.topic = topic;
14:     this.content = content;
15:   }
16:
17:   public String getContent(){
18:     return content;
19:   }
20:   public void setContent(String content){
21:     this.content = content;
22:   }
23:   public String getTopic(){
24:     return topic;
25:   }
26:   public void setTopic(String topic){
27:     this.topic = topic;
28:   }
29: }
```

La clase `Publisher` representa a los *editores* e implementa el método `Publish`, que emite un mensaje a un agente de Publicación/Suscripción.

Cuadro 5.23: Clase `Publisher`

```
 1: package patronPublishSubscribe;
 2:
 3: public class Publisher
 4: {
 5:   private String id;
 6:
 7:   public Publisher(String id){
 8:     this.id = id;
 9:   }
10:
11:   /* GET and SET methods */
12:
13:   /* Envía un mensaje al agente de
14:      Publicación/Suscripción */
15:   public void Publish(
16:     Message message,
17:     PublishSubscribeAgent publishSubscribeAgent){
```

```
18:       publishSubscribeAgent.addMessageToQueue(message);
19:     }
20:   }
```

La clase `Subscriber` representa a los *suscriptores* e implementa los siguientes métodos:

- `SubscribeToTopic`: asocia un suscriptor a un tema en un agente de Publicación/ Suscripción,
- `UnsubscribeFromTopic`: desasocia un suscriptor a un tema en un agente de Publicación/Suscripción,
- `GetMessagesForSubscriberOfTopic`: solicita los mensajes sobre un tema específico a un agente de Publicación/Suscripción, y
- `PrintMessages`: para un fin demostrativo, muestra en pantalla todos los mensajes recibidos por el suscriptor.

Cuadro 5.24: Clase `Subscriber`

```
1:    package patronPublishSubscribe;
2:
3:    import java.util.ArrayList;
4:    import java.util.List;
5:
6:    public class Subscriber
7:    {
8:      private String id;
9:
10:     public Subscriber(String id){
11:       this.id = id;
12:     }
13:
14:     /* GET and SET methods */
15:
16:     /* Almacena todos los mensajes recibidos por los editores
17:      */
```

```
18:     private List<Message> subscriberMessages =
19:       new ArrayList<>();
20:
21:     public List<Message> GetSubscriberMessages() {
22:       return subscriberMessages;
23:     }
24:
25:     public void SetSubscriberMessages(
26:       List<Message> subscriberMessages
27:       ){
28:       this.subscriberMessages = subscriberMessages;
29:     }
30:
31:     /* Asocia un suscriptor a un tema en un agente
32:        de Publicación/Suscripción */
33:     public void SubscribeToTopic(
34:       String topic,
35:       PublishSubscribeAgent publishSubscribeAgent
36:     ){
37:       publishSubscribeAgent.AssociateSubscriberToTopic
38:       (topic, this);
39:     }
40:
41:     /* Desasocia un suscriptor de un tema en un agente
42:       de Publicación/Suscripción */
43:     public void UnsubscribeFromTopic(
44:       String topic,
45:       PublishSubscribeAgent publishSubscribeAgent
46:     ){
47:       publishSubscribeAgent.DisassociateSubscriberFromTopic
48:       (topic, this);
49:     }
50:
51:     /* Solicita los mensajes sobre un tema específico a un
52:        agente de Publicación/Suscripción */
53:     public void GetMessagesForSubscriberOfTopic(
54:       String topic,
55:       PublishSubscribeAgent publishSuscribeAgent
56:     ){
57:       publishSuscribeAgent.GetMessagesForSubscriberOfTopic
58:       (topic, this);
59:     }
60:
```

```
61:     /* Imprime los mensajes recibidos por el suscriptor */
62:     public void PrintMessages(){
63:       System.out.println("Subscriber with ID: " + this.id);
64:       int i = 1;
65:       for( Message message : subscriberMessages )
66:         System.out.println(i++ + ". Topic: "
67:           + message.getTopic() + ". Content: "
68:           + message.getContent());
69:       System.out.println();
70:     }
71:   }
```

La clase `PublishSubscribeAgent` representa a los *agentes de Publicación/Suscripción* e implementa las estructuras `subscribersTopicMap` para almacenar a los suscriptores interesados en un tema, y `messageQueue` para almacenar los mensajes emitidos por los editores. Sus métodos son los siguientes:

- `AddMessageToQueue`: añade un mensaje publicado por un editor a la cola,
- `AssociateSubscriberToTopic`: asocia un suscriptor a un determinado tema,
- `DisassociateSubscriberFromTopic`: desasocia un suscriptor de un determinado tema,
- `BroadcastMessages`: transmite, y elimina, los mensajes de la cola `messageQueue` a sus respectivos suscriptores, de acuerdo a su tema, y
- `GetMessagesForSubscriberOfTopic`: envía y elimina los mensajes sobre un determinado tema a un determinado suscriptor.

Cuadro 5.25: Clase `PublishSubscribeAgent`

```
1: package patronPublishSubscribe;
2:
3: import java.util.HashMap;
4: import java.util.HashSet;
5: import java.util.LinkedList;
```

```
 6: import java.util.List;
 7: import java.util.Map;
 8: import java.util.Queue;
 9: import java.util.Set;
10:
11: public class PublishSubscribeAgent
12: {
13:   /* Almacena el conjunto de temas de interés para los
14:      suscriptores, evitando los duplicados */
15:   Map<String, Set<Subscriber>> subscribersTopicMap = new
16:     HashMap<>();
17:   /* Mantiene los mensajes publicados por los editores */
18:   Queue<Message> messagesQueue = new LinkedList<>();
19:
20:   private String id;
21:
22:   public PublishSubscribeAgent(String id){
23:     this.id = id;
24:   }
25:
26:   /* Añade un mensaje publicado por un editor a la cola */
27:   public void addMessageToQueue(Message message){
28:     messagesQueue.add(message);
29:   }
30:
31:   /* Asocia un suscriptor a un determinado tema */
32:   public void AssociateSubscriberToTopic(
33:     String topic,
34:     Subscriber subscriber
35:   ){
36:     if( subscribersTopicMap.containsKey(topic) ){
37:       Set<Subscriber> subscribers =
38:         subscribersTopicMap.get(topic);
39:       subscribers.add(subscriber);
40:       subscribersTopicMap.put(topic, subscribers);
41:     }else{
42:       Set<Subscriber> subscribers = new HashSet<>();
43:       subscribers.add(subscriber);
44:       subscribersTopicMap.put(topic, subscribers);
45:     }
46:   }
47:
48:   /* Desasocia un suscriptor de un determinado tema */
```

```
49:   public void DisassociateSubscriberFromTopic(
50:     String topic,
51:     Subscriber subscriber
52:   ){
53:     if( subscribersTopicMap.containsKey(topic) ){
54:       Set<Subscriber> subscribers =
55:         subscribersTopicMap.get(topic);
56:       subscribers.remove(subscriber);
57:       subscribersTopicMap.put(topic, subscribers);
58:     }
59:   }
60:
61:   /* Transmite, y elimina, los mensajes de la cola
62:      messageQueue a sus respectivos suscriptores, de acuerdo
63:      a su tema */
64:   public void BroadcastMessages(){
65:     while( !messagesQueue.isEmpty() ){
66:       Message message = messagesQueue.remove();
67:       String topic = message.getTopic();
68:
69:       Set<Subscriber> subscribersOfTopic =
70:         subscribersTopicMap.get(topic);
71:
72:       for( Subscriber subscriberTEMP : subscribersOfTopic ){
73:         /* Agrega el mensaje transmitido a la cola de
74:            mensajes de cada suscriptor interesado */
75:         List<Message> subscriberMessagesQueue =
76:           subscriberTEMP.GetSubscriberMessages();
77:         subscriberMessagesQueue.add(message);
78:         subscriberTEMP.SetSubscriberMessages(
79:           subscriberMessagesQueue);
80:       }
81:     }
82:   }
83:
84:   /* Envía y elimina los mensajes sobre un determinado tema
85:      a un determinado suscriptor */
86:   public void GetMessagesForSubscriberOfTopic(String topic,
87:     Subscriber subscriber) {
88:     while( !messagesQueue.isEmpty() ){
89:       Message message = messagesQueue.remove();
90:
91:       if(message.getTopic().equalsIgnoreCase(topic)){
```

```
92:
93:           Set<Subscriber> subscribersOfTopic =
94:             subscribersTopicMap.get(topic);
95:
96:           if( subscribersOfTopic != null ){
97:             for( Subscriber subscriberTEMP :
98:               subscribersOfTopic ){
99:               if( subscriberTEMP.equals(subscriber) ){
100:                /* Añade el mensaje transmitido a la cola de
101:                   mensajes de cada suscriptor interesado */
102:                List<Message> subscriberMessagesQueue =
103:                  subscriber.GetSubscriberMessages();
104:                subscriberMessagesQueue.add(message);
105:                subscriber.SetSubscriberMessages(
106:                  subscriberMessagesQueue);
107:              }
108:            }
109:          }
110:        }
111:      }
112:    }
113:  }
```

Finalmente, la clase `Main` presenta un ejemplo del uso de las clases anteriores.

Cuadro 5.26: Clase `Main`

```
1:  package patronPublishSubscribe;
2:
3:  public class Main
4:  {
5:    public static void main(String[] args){
6:      /* Declaración del agente de Publicación/Suscripción */
7:      PublishSubscribeAgent publishSubscribeAgent =
8:        new PublishSubscribeAgent("PUBSUBAGE1");
9:
10:     /* Declaración de Editores */
11:     Publisher Publisher1 = new Publisher("PUB1");
12:     Publisher Publisher2 = new Publisher("PUB2");
13:
```

```
14:     /* Declaración de Suscriptores */
15:     Subscriber Subscriber1 = new Subscriber(“SUB1”tv);
16:     Subscriber Subscriber2 = new Subscriber(“SUB2”);
17:     Subscriber Subscriber3 = new Subscriber(“SUB3”);
18:
19:     /* Declaración de algunos mensajes */
20:     Message topic1Message1 = new Message(“Topic 1”,
21:       “Message 1”);
22:     Message topic1Message2 = new Message(“Topic 1”,
23:       “Message 2”);
24:
25:     Message topic2Message1 = new Message(“Topic 2”,
26:       “Message 1”);
27:     Message topic2Message2 = new Message(“Topic 2”,
28:       “Message 2”);
29:
30:     /* Publicación en el agente de
31:        Publicación/Suscripción */
32:     Publisher1.Publish(topic1Message1,
33:       publishSubscribeAgent);
34:     Publisher1.Publish(topic1Message2,
35:       publishSubscribeAgent);
36:
37:     Publisher2.Publish(topic2Message1,
38:       publishSubscribeAgent);
39:     Publisher2.Publish(topic2Message2,
40:       publishSubscribeAgent);
41:
42:     /* Suscripción a los temas */
43:
44:     // Subscriber1 se suscribe sólo al tema 1 (Topic 1).
45:     Subscriber1.SubscribeToTopic(“Topic 1”,
46:       publishSubscribeAgent);
47:     // Subscriber2 se suscribe sólo al tema 2 (Topic 2).
48:     Subscriber2.SubscribeToTopic(“Topic 2”,
49:       publishSubscribeAgent);
50:     // Subscriber3 se suscribe a los temas 1 y 2 (Topic 1,
51:     // Topic 2).ss
52:     Subscriber3.SubscribeToTopic(“Topic 1”,
53:       publishSubscribeAgent);
54:     Subscriber3.SubscribeToTopic(“Topic 2”,
55:       publishSubscribeAgent);
56:
```

```
57:       // Se transmiten y se eliminan los mensajes a los
58:       // suscriptores.
59:       publishSubscribeAgent.BroadcastMessages();
60:
61:       // Se imprimen los mensajes que hayn recibido los
62:       // suscriptores.
63:       Subscriber1.PrintMessages();
64:       Subscriber2.PrintMessages();
65:       Subscriber3.PrintMessages();
66:
67:       /* Después de un broadcast se habrán limpiado los
68:          mensajes */
69:
70:       /* Declaración de nuevos mensajes */
71:       Message topic1Message3 = new Message(“Topic 1”,
72:         “Message 3”);
73:       Message topic2Message4 = new Message(“Topic 1”,
74:         “Message 4”);
75:
76:       /* Publicación en el agente de
77:          Publicación/Suscripción */
78:       Publisher1.Publish(topic1Message3,
79:         publishSubscribeAgent);
80:       Publisher1.Publish(topic2Message4,
81:         publishSubscribeAgent);
82:
83:       /* Se obtienen todos los mensajes de un determinado tema
84:          para Subscriber1 */
85:       Subscriber1.GetMessagesForSubscriberOfTopic(“Topic 1”,
86:         publisherSubscriptorAgent);
87:       Subscriber1.PrintMessages();
88:     }
89:   }
```

---

4. **Actualización del contexto**

Los cambios en el contexto se notifican mediante el patrón *Observer*, que permite tener desacoplados el objeto que emite el evento (sujeto concreto) de todos aquellos interesados en sus cambios (observadores concretos).

El *Java Development Kit* dispone de la clase `Observable` y la interfaz `Observer` para implementar este patrón. Por una parte, el objeto que producirá los eventos, i.e. el sujeto concreto, debe extender o contener una propiedad de tipo *Observable*. Por la otra, la clase que recibirá los eventos, i.e. el observador concreto, debe implementar la interfaz *Observer*.

La clase `Observable` dispone de métodos para añadir y eliminar observadores, contar su número, y conocer cuando un objeto ha cambiado (`hasChanged`). La interfaz `Observer` contiene sólo el método `update`, que recibe dos parámetros: 1) la instancia del objeto bajo el rol de sujeto concreto, y 2) un objecto a modo de argumento que el sujeto concreto envía.

A continuación se muestra un sencillo ejemplo para su implementación, compuesto por el paquete `patronObserver`, y las clases `ObservedObject`, y `ObserverObject`.

La clase `ObservedObject` representa a los *objetos observados* e implementa el atributo `stateOfInterest`, que representa el estado de interés para los *objetos observadores*. También implementa la estructura *stateOfInterestEvent*, que almacena el nuevo y anterior valor del atributo `stateOfInterest` y que es enviada a los objetos observadores (veáse la Tabla 5.27).

Cuadro 5.27: Clase `ObservedObject`

```
 1: package patronObserver;
 2:
 3: import java.util.Observable;
 4:
 5: public class ObservedObject {
 6:
 7:   public class stateOfInterestEvent
 8:   {
 9:     private ObservedObject observedObject;
10:     private String oldStateOfInterest;
11:     private String newStateOfInterest;
12:
13:     public stateOfInterestEvent(
14:       ObservedObject observedObject,
15:       String oldStateOfInterest,
```

```
16:         String newStateOfInterest
17:       ){
18:         this.observedObject = observedObject;
19:         this.oldStateOfInterest = oldStateOfInterest;
20:         this.newStateOfInterest = newStateOfInterest;
21:       }
22:
23:       public ObservedObject getObject() {
24:         return observedObject;
25:       }
26:       public String getOldStateOfInterest() {
27:         return oldStateOfInterest;
28:       }
29:       public String getNewStateOfInterest() {
30:         return newStateOfInterest;
31:       }
32:     }
33:
34:     private static final ObservableObject OBSERVABLE;
35:
36:     private String id;
37:     private String stateOfInterest;
38:
39:     static {
40:       OBSERVABLE = new ObservableObject();
41:     }
42:
43:     public static Observable getObservable(){
44:       return OBSERVABLE;
45:     }
46:
47:     public ObservedObject(String id, String stateOfInterest){
48:       this.id = id;
49:       this.stateOfInterest = stateOfInterest;
50:     }
51:
52:     public String getId(){
53:       return id;
54:     }
55:     public String getStateOfInterest(){
56:       return stateOfInterest;
57:     }
58:
```

```
59:    public void setStateOfInterest(String stateOfInterest) {
60:      stateOfInterestEvent event =
61:        new stateOfInterestEvent(this, this.stateOfInterest,
62:        stateOfInterest);
63:
64:      this.stateOfInterest = stateOfInterest;
65:
66:      synchronized (OBSERVABLE) {
67:        OBSERVABLE.setChanged();
68:        OBSERVABLE.notifyObservers(event);
69:      }
70:    }
71:
72:    private static class ObservableObject extends Observable
73:    {
74:      @Override
75:      public synchronized void setChanged() {
76:        super.setChanged();
77:      }
78:    }
79:  }
```

La clase `ObserverObject` representa a los *objetos observadores*, que sobreescribe el método `update`, para un fin demostrativo muestra en pantalla los valores nuevo y anterior del estado de interés `stateOfInterest` (veáse la Tabla 5.28).

Cuadro 5.28: Clase `ObserverObject`

```
 1:  package patronObserver;
 2:
 3:  import java.util.Observable;
 4:  import java.util.Observer;
 5:
 6:  import patronObserver.ObservedObject.stateOfInterestEvent;
 7:
 8:  public class ObserverObject implements Observer {
 9:
10:    String id;
11:
```

```
12:   public ObserverObject(String id){
13:     this.id = id;
14:   }
15:
16:   @Override
17:   @SuppressWarnings(\unchecked")
18:   public void update(Observable observable, Object args) {
19:     if (args instanceof stateOfInterestEvent) {
20:       stateOfInterestEvent event = (stateOfInterestEvent)
21:         args;
22:       System.out.printf("Observer" + this.id
23:         + ": the object with id " + event.getObject().getId()
24:         + " has changed its state of interest from"
25:         + event.getOldStateOfInterest()
26:         + "to" + event.getNewStateOfInterest()\n);
27:     }
28:   }
29: }
```

Finalmente, la clase `Main` presenta un ejemplo del uso de las clases anteriores.

Cuadro 5.29: Clase `Main`

```
 1: package patronObserver;
 2:
 3: import java.util.Observer;
 4:
 5: public class Main {
 6:   public static void main(String[] args) {
 7:     ObservedObject object1 = new ObservedObject("OBJECT1",
 8:       "Status A");
 9:     ObservedObject object2 = new ObservedObject("OBJECT2",
10:       "Status A");
11:
12:     Observer observer1 = new ObserverObject("1");
13:     Observer observer2 = new ObserverObject("2");
14:
15:     ObservedObject.getObservable().addObserver(observer1);
16:     ObservedObject.getObservable().addObserver(observer2);
17:
```

```
18:       object1.setStateOfInterest("Status B");
19:       object2.setStateOfInterest("Status C");
20:       object2.setStateOfInterest("Status A");
21:     }
22:   }
```

Después, en caso de las situaciones permanentes cuyo estado haya cambiado, éste se almacena en la estructura `permanentSituations`, con el fin de contestar algunas de las preguntas que el sistema colaborativo pueda requerir.

Finalmente, la adaptación de cada objeto se realizará mediante las llamadas *reglas de adaptación*, reglas estáticas de la forma **if** *detected situation* **then** *planned adaptation* (véase la Tabla 5.13). Éstas reglas, sin embargo, son particulares al dominio de la aplicación.

Capítulo 6

HERRAMIENTA DE ADMINISTRACIÓN DE CONTENIDOS VÍA *NFC*

Este capítulo describe la herramienta de administración de contenidos vía *NFC*, implementada con el fin de validar el modelado semántico para el espacio colaborativo global y la arquitectura conceptual propuestos.

La herramienta de administración de contenidos vía *NFC* permite facilitar a un grupo de colaboradores la información que se tratará en una reunión previamente planificada, i.e., el nombre, la fecha, los objetivos, los puntos a tratar, y los asistentes de la reunión, además de los documentos relevantes a la misma.

La información mencionada se registrará en una etiqueta *NFC* que será colocada en el lugar de la reunión para que, al acercar el dispositivo móvil a la etiqueta *NFC*, los colaboradores la reciban automáticamente.

La herramienta de administración de contenidos vía *NFC* está asociada al escenario *adaptación de la información* y comprende todas las dimensiones del contexto de uso de

los sistemas colaborativos, i.e., el *grupo de colaboradores*, el *conjunto de plataformas*, y el *ambiente compartido*.

En las secciones 6.1 y 6.2 se explican brevemente las tecnologías utilizadas en el desarrollo de la herramienta de administración de contenidos vía *NFC*. Posteriormente, la sección 6.3 expone los requerimientos funcionales y no funcionales de la citada herramienta, mientras que la sección 6.4 describe las ventanas de la aplicación, y las secciones 6.5, 6.6, 6.7, 6.8, y 6.9, muestra el modelado semántico, los servicios, las variables contextuales, los eventos contextuales, las situaciones y las adaptaciones consideradas, respectivamente. Finalmente, la sección 6.10 expone la evaluación de la aplicación por parte de un conjunto de usuarios.

6.1. NFC

Near Field Communication (*NFC*) es una tecnología de conectividad sin contacto de poco alcance y basada en estándares que facilita realizar transacciones, intercambiar contenido digital, y conectar dispositivos electrónicos con un toque. *NFC* es compatible con cientos de millones de tarjetas inalámbricas y lectores que han sido desarrollado en todo el mundo.

La tecnología *NFC* habilita interacciones bidireccionales simples y seguras entre dispositivos electrónicos entre dispositivos electrónicos.*NFC* utiliza elementos clave de los estándares existentes para la tecnología de tarjetas sin contacto, ISO/IEC 14443 A&B y JIS-X 6319-4, por lo cual es compatible con la infraestructura existente de tarjetas sin contacto y permite al usuario utilizar un dispositivo entre diferentes sistemas.

Extendiendo la capacidad de la tecnología de las tarjetas sin contacto, *NFC* también permite a los dispositivos compartir información a una distancia menor a 4 cm con una velocidad de comunicación máxima de 424 kbps.

La comunicación bidireccional vía *NFC* es ideal para establecer conexiones con otras tecnologías con la simplicidad de un toque, así permite al "Administrador de contenidos vía *NFC*" enviar los documentos pertinentes a los miembros de una reunión.

6.2. Android

Los dispositivos móviles tienen capacidades actualessimilares a una computadora personal pero, generalmente, siempre acompañan a su dueño, lo cual permite una gama de aplicaciones más cercanas al usuario.

Entre los sistemas operativos para dispositivos móviles, Android es la plataforma móvil más popular del mundo ya que reside en cientos de millones de dispositivos móviles en más de 190 países alrededor del mundo y ésta en constante crecimiento. Éste se debe a los elementos novedosos que incluye, desde el diseño fácil de interfaces de usuario, gráficos, posicionamiento, manejo de sensores, entre otros.

6.3. Requerimientos funcionales y no funcionales

De acuerdo con [Romero Gama et al., 2013], los requerimientos funcionales del administrador de contenidos vía *NFC* son los siguientes:

1. Permitir que un colaborador ingrese la información relacionada de una reunión y suba a un servidor de contenidos, los documentos que son relevantes a la misma,

2. Permitir que un colaborador pueda grabar en una etiqueta *NFC* la información relacionada a una reunión,

3. Permitir que un colaborador, al acercar su dispositivo a una etiqueta *NFC*, tras haber iniciado sesión, reciba automáticamente en su dispositivo la información referente a una reunión,

4. Detectar automáticamente qué formatos de documento son soportados por las herramientas de visualización instaladas en el dispositivo.

5. Iniciar la descarga de documentos que se requieren revisar durante una reunión. Durante la solicitud de documentos, se debe determinar qué tipos de formato son soportados,

de tal manera que los documentos a descargar sean en un formato soportado.

6. Mostar en la interfaz de usuario de la aplicación un enlace o ícono por cada documento descargado y permitir abrir dicho documento al hacer clic sobre su ícono.

Y, de acuerdo con [Romero Gama et al., 2013], los requerimientos no funcionales del administrador de contenidos vía *NFC* son los siguientes:

1. Habilitar el adaptador de red inalámbrica Wi-Fi, si éste se encuentra deshabilitado.
2. Permitir la apertura de los documentos descargados.
3. No permitir la transmisión de información a dispositivos de usuarios que no estén registrados.

6.4. Estructura

La herramienta de administración de contenidos vía *NFC* se compone, principalmente, por un servidor multi-hilo de contenidos, dispositivos Android que actúan como clientes y etiquetas *NFC* (véase la Figura 6.1).

El servidor *GlassFish* almacena las ontologías de la arquitectura RAMS, los documentos asociados a una reunión, y los servicios que realizan las etapas de la arquitectura conceptual descrita. Las clases `Meeting`. `User`, y `Document` del modelo semántico representan una determinada reunión y su información asociada; y, la clase `Viewer`, un determinado visor de documentos para el sistema operativo Android.

El servidor: 1) crea e inicializa las estructuras que representan el modelo de estado, las situaciones transitorias y las permanentes, 2) realiza consultas a las ontologías para autenticar el inicio de sesión de los usuarios, y obtener la información de una determinada reunión, 3) realiza la detección de situación, y 4) adapta los documentos asociados a una reunión para su despliegue óptimo en cada cliente.

El dispositivo Android ejecuta la aplicación *NFC Meeting*, que provee las ventanas para que el usuario registre una reunión, la visualice o edite tras leer su identificador asociado en una etiqueta *NFC*.

La aplicación: 1) provee la interfaz de usuario que permite el registro y/o visualización de una reunión, 2) obtiene información del dispositivo Android en el cual se ejecuta, y 3) permite la lectura de etiquetas *NFC*.

Finalmente, la etiqueta *NFC* sólo almacena el identificador de una reunión.

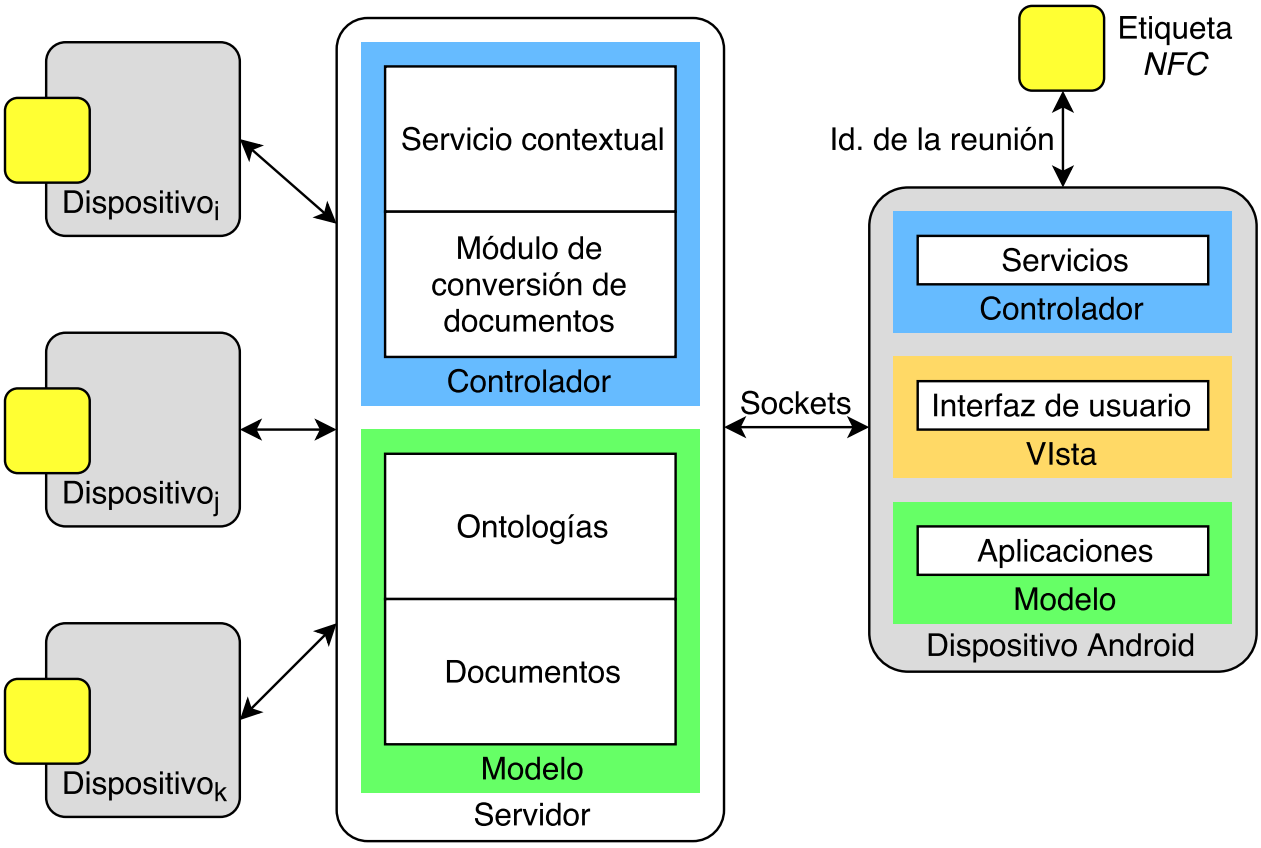


Figura 6.1: Estructura general

La aplicación *NFC Meeting* se compone de las siguientes ventanas:

- **Inicio de sesión**: en ella se ingresa el nombre del colaborador y su identificador, datos

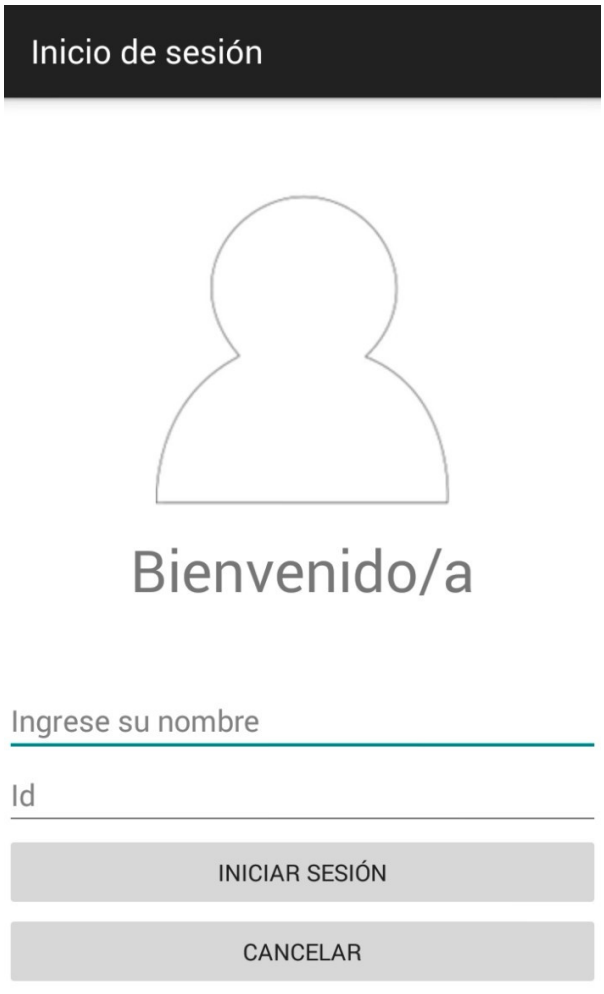
que serán corroborados con el servidor para permitir o denegar el acceso a la aplicación (véase la Figura6.2a),

- **Lectura de una etiqueta *NFC***: es la ventana principal de la aplicación, permite registrar una nueva reunión o leer el contenido de una etiqueta *NFC*, creando el evento contextual *The `$DEVICE` has read a NFC tag* (véase la Figura6.2b),

- **Registro de una nueva reunión**: muestra un formulario para ingresar los datos de una nueva reunión (nombre, fecha, objetivos, puntos a tratar, y asistentes) y adjuntar los documentos asociados, una vez guardada la reunión en las ontologías se escribirá el identificador asociado a la reunión en una etiqueta *NFC* (véase la Figura6.2c),

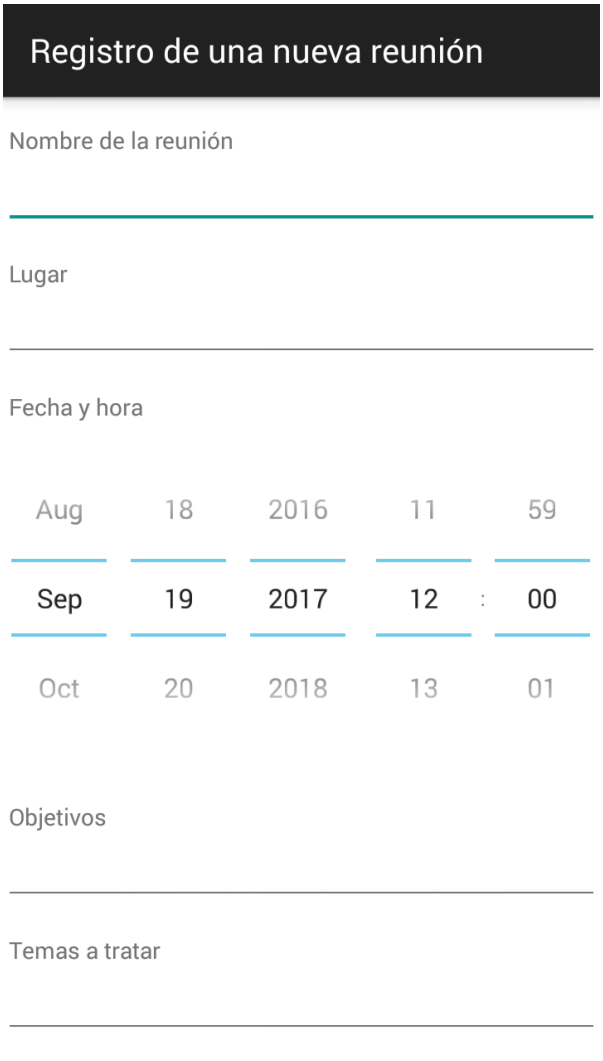
- **Consulta de una reunión**: muestra la información asociada a una reunión previamente registrada en el servidor y los enlaces a los documentos asociados en un formato que el dispositivo sea capaz de visualizar, ya sea el original o una versión en JPEG (véase la Figura6.2d), y

- **Edición de una reunión**: muestra un formulario para modificar los datos ingresados de una reunión previamente registrada en el servidor (véase la Figura6.2e).

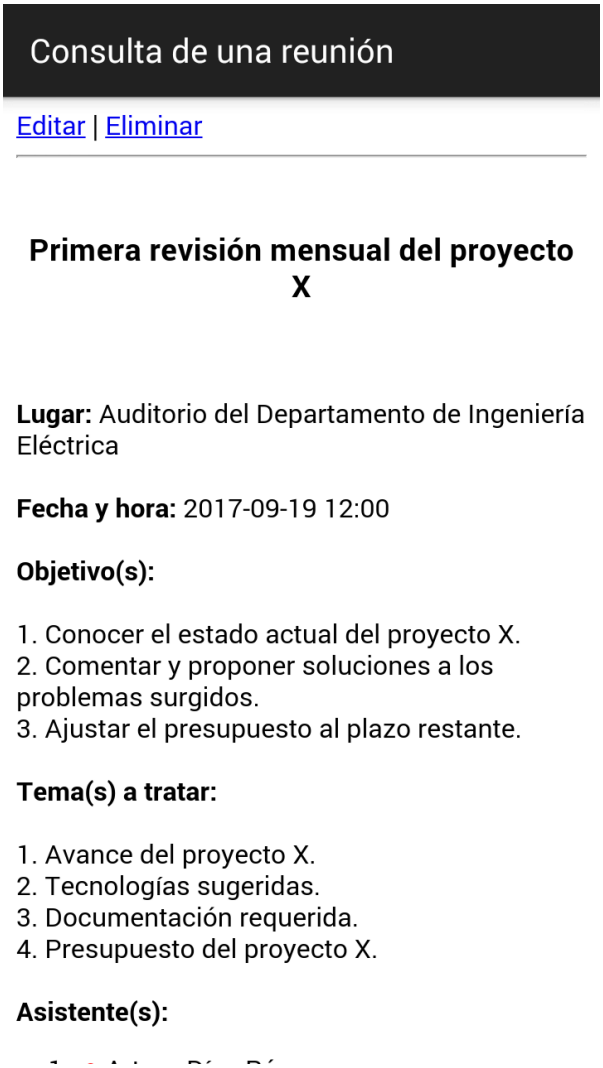


(a) Inicio de sesión

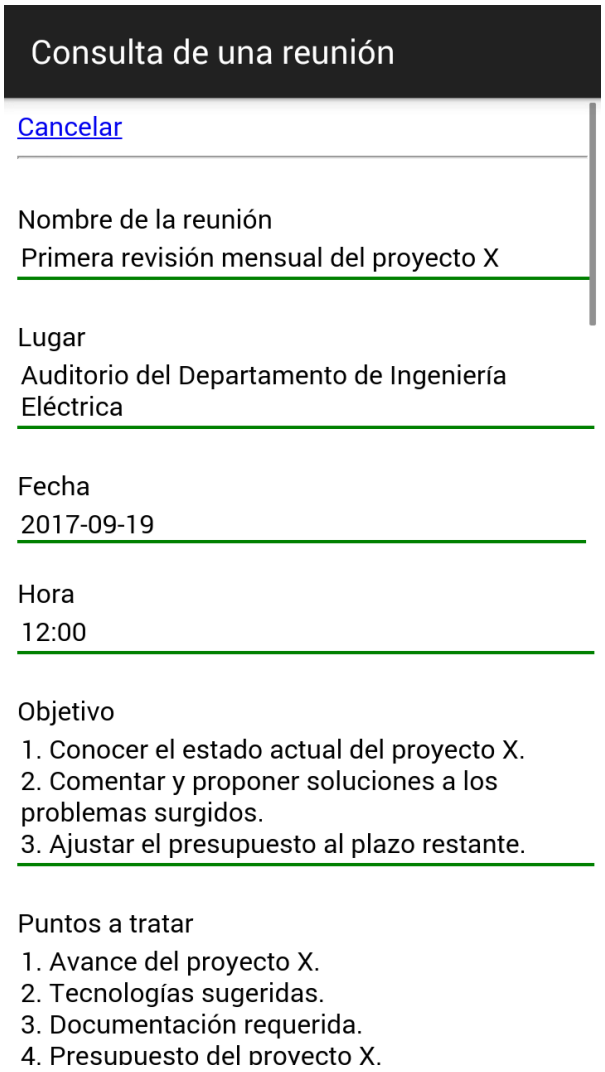
(b) Lectura de una etiqueta *NFC*



(c) Registro de una nueva reunión



(d) Consulta de una reunión

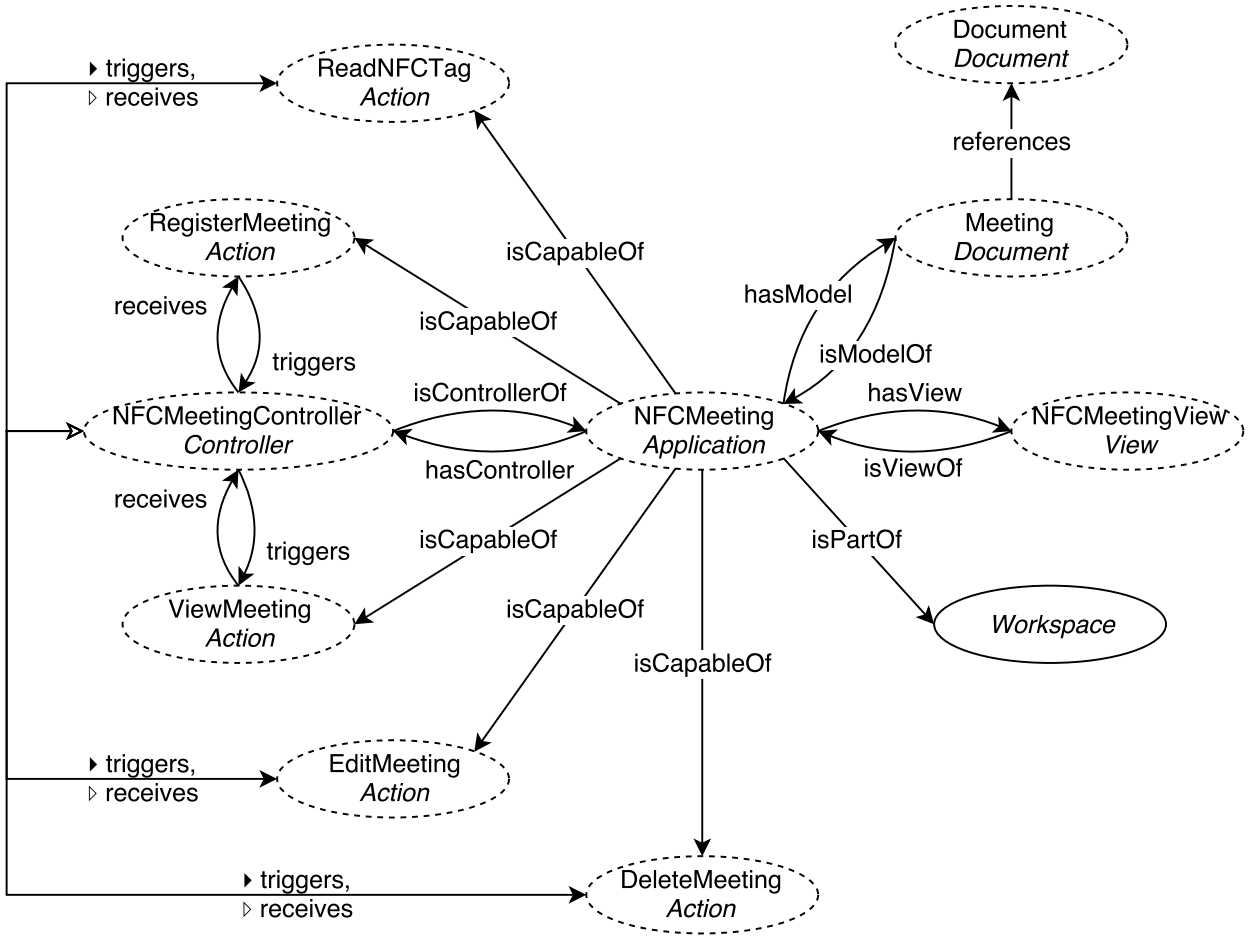


(e) Edición de una reunión

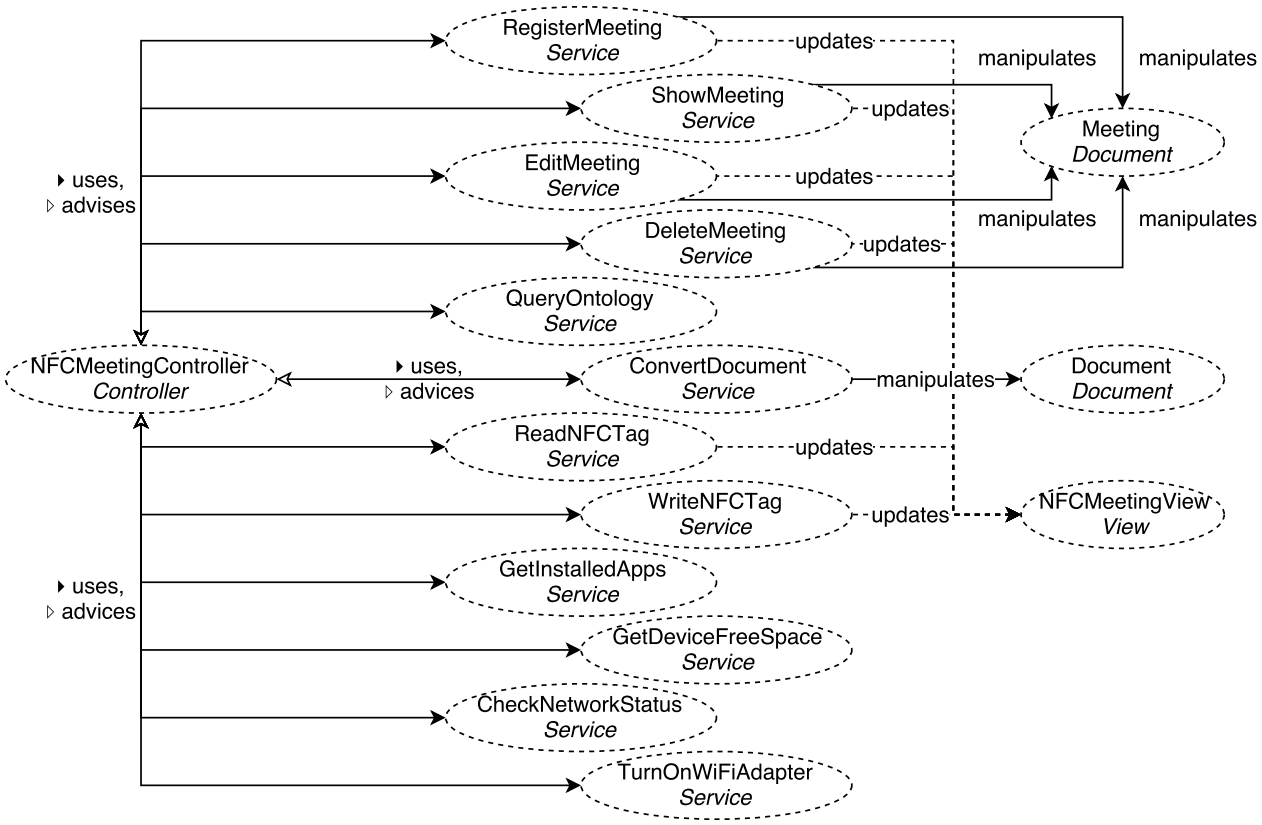
Figura 6.2: Ventanas de la aplicación *NFC Meeting*

6.5. Modelado semántico

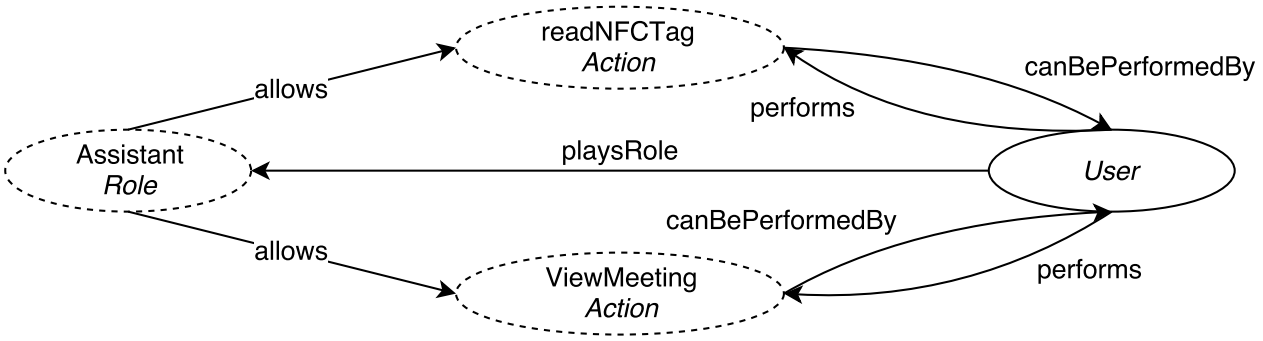
La Figura 6.3 muestra las instancias del modelado semántico propuesto en el capítulo Modelado semántico del espacio colaborativo global para la herramienta *administración de contenidos vía NFC*, en 6.3(a) se muestran las propiedades de la aplicación `NFCMeeting`; en 6.3(b), los servicios utilizados; en 6.3(c) y 6.3(d), las propiedades de los roles considerados; y, en 6.3(e), las propiedades para un usuario de la aplicación.



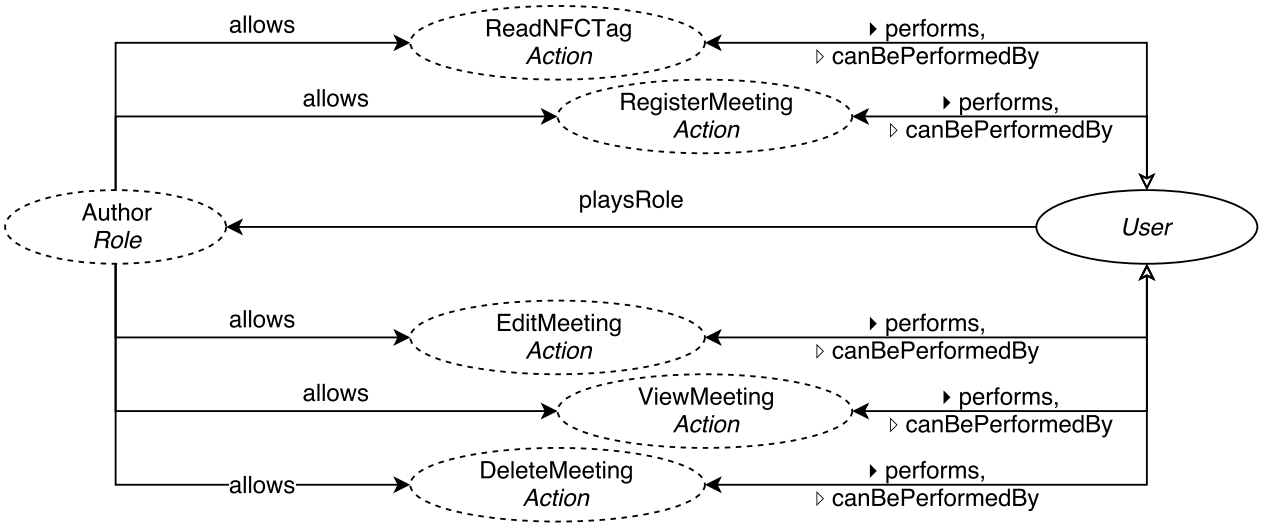
(a) Propiedades de la aplicación `NFCMeeting`



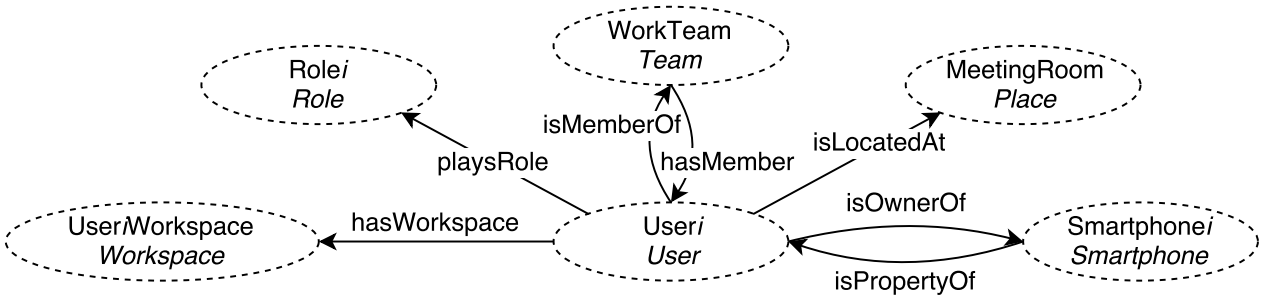
(b) Servicios de la aplicación `NFCMeeting`



(c) Propiedades del rol `Assistant`



(d) Propiedes del rol `Author`



(e) Propiedades de un usuario

Figura 6.3: Modelado semántico de la aplicación `NFCMeeting`

Adicionalmente, las clases `Meeting` y `Viewer` se añadieron al modelo semántico de la arquitectura RAMS, a fin de suministrar la información necesaria para el desarrollo de la *herramienta de administración de contenidos vía NFC*.

- `Meeting`: representa una reunión, almacena su nombre, fecha, objetivos y puntos a tratar mediante propiedades de datos; sus asistentes y documentos relacionados mediante propiedades de objeto. Es una subclase de la clase `Process`, subsumida en la clase `Organizational Context`, y a su vez, ésta subsumida en la clase `Context` (véase la Figura 6.4).

- `Viewer`: representa una aplicación capaz de visualizar o editar algún formato de documento, almacena su nombre, versión, formatos admitidos, y sistemas operativos compatibles mediantes propiedades de datos. Es una subclase de la clase `Softwarem` subsumida en la clase `VirtualResource` (véase la Figura 6.5).

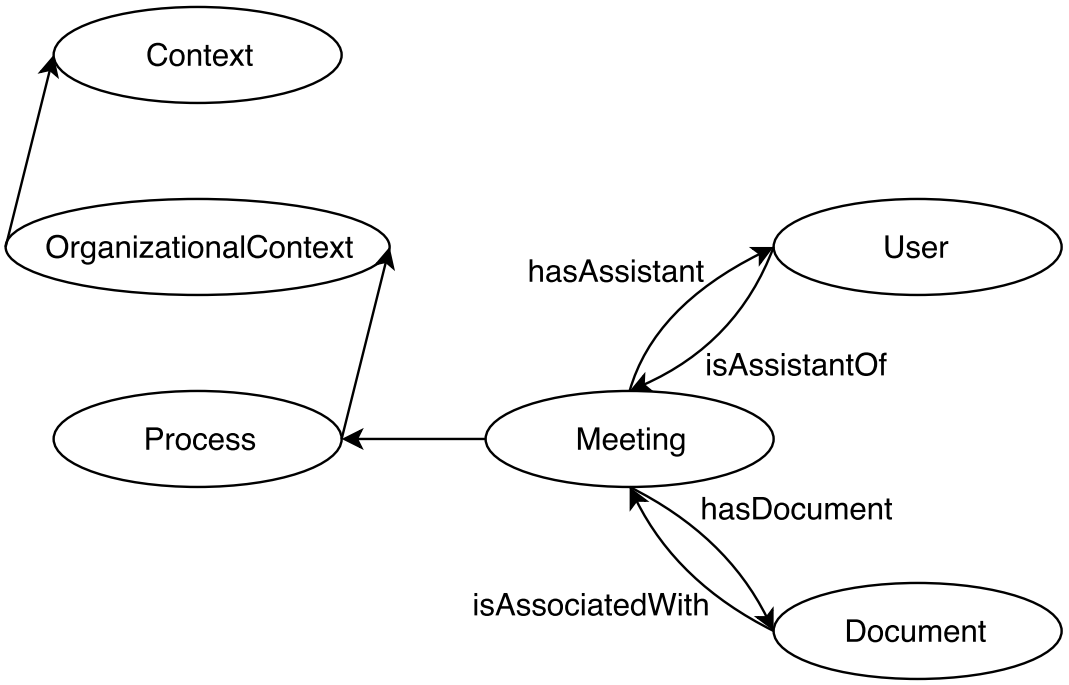


Figura 6.4: Clase `Meeting`

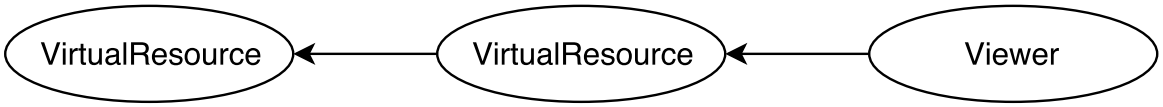


Figura 6.5: Clase `Viewer`

6.6. Servicios

En esta sección se listan y describen los servicios utilizados por el controlador de la *herramienta de administración de contenidos vía NFC*.

- `RegisterMeeting`: crea una reunión y carga al servidor sus documentos asociados. Para ello, utiliza el servicio `QueryOntology` para ejecutar las consultas SPARQL que crean las instancias de las clases `meeting` y `document` que representan a la reunión y a sus documentos asociados, respectivamente. En la instancia de `meeting` almacena su nombre, fecha, objetivos y puntos a tratar mediante propiedades de datos y, mediante propiedades de objeto, relaciona a la reunión los documentos asociados, i.e., instancias

de la clase `document`, y sus asistentes, i.e., instancias de clase `user`,

- `ShowMeeting`: presenta la información de una reunión dado su identificador en el dispositivo de un colaborador. Para ello, utiliza el servicio `QueryOntology` para ejecutar las consultas SPARQL que obtienen las propiedades de datos y propiedades de objeto del individuo que representa a la reunión,

- `EditMeeting`: modifica la información de una reunión dado su identificador. Para ello, requiere el servicio `QueryOntology` para ejecutar las consultas SPARQL que modifican las propiedades de datos y de objetos del individuo que representa a la reunión,

- `DeleteMeeting`: elimina una reunión dado su identificador. Para ello, utiliza el servicio `QueryOntology` para ejecutar las consultas SPARQL que eliminan el individuo que representa a la reunión, sus documentos asociados y los individuos que lo representan, así como las propiedades de objeto entre la reunión, los documentos y asistentes,

- `QueryOntology`: carga en memoria las ontologías *HumanResource*, *VirtualResource*, y *Context*. Permanece a la espera para ejecutar las consultas SPARQL que le sean solicitadas,

- `ConvertDocument`: genera una versión de un documento en un formato distinto al original, este servicio hace uso de los programas *imagemagick* para convertir un documento PDF a una versión en JPEG, y de *LibreOffice* versión 4.0 o superior y *unoconvert* para convertir en formato PDF los siguientes formatos de documentos:

Cuadro 6.1: Formatos soportados de gráficos

Extensión	**Formato**
fodg	Flat XML ODF Drawing
odg	ODF Drawing
otg	ODF Drawing Template

Cuadro 6.2: Formatos soportados de documentos

Extensión	**Formato**
doc	Microsoft Word 97/2000/XP/2003
docx	Microsoft Word 2007/2010/2013 XML
dot	Microsoft Word 97/2000/XP/2003 Template
fodt	Flat XML ODF Text Document
html	HTML Document
odf	ODF Text Document
ott	ODF Text Document Template
rtf	Rich Text
uot	Unified Office Format Text
txt	Plain Text
xml	Microsoft Word 2003 XML — DocBoox

Cuadro 6.3: Formatos soportados de presentaciones

Extensión	**Formato**
fodp	Flat XML ODF Presentation
odp	ODF Presentation
odg	ODF Drawing
otp	ODF Presentation Template
pot	Microsoft PowerPoint 97/2000/XP/2003 Template
potm	Microsoft PowerPoint 2007/2010/2013 XML Template
ppt	Microsoft PowerPoint 97/2000/XP/2003
pps	Microsoft PowerPoint 97/2000/XP/2003 Autoplay
pptx	Microsoft PowerPoint 2007/2010/2013 XML
ppsx	Microsoft PowerPoint 2007/2010/2013 XML Autoplay

Cuadro 6.4: Formatos soportados de hojas de cálculo

Extensión	**Formato**
csv	Text CSV
fods	Flat XML ODF Spreadsheet
html	HTML Document
ods	ODF Spreadsheet
ots	ODF Spreadsheet Template
slk	SYLK
uos	Unified Office Format Spreadsheet
xls	Microsoft Excel 97/2000/XP/2003
xlsx	Microsoft Excel 2007/2010/2013 XML
xlt	Microsoft Excel 97/2000/XP/2003 Template

Cuadro 6.5: Otros formatos soportados

Extensión	Formato
odf	ODF Database, ODF Formula
odm	ODF Master Document
otf	ODF Formula Template
oth	HTML Document Template
mml	MathML 1.01

- `ReadNFCTag`: lee la información de una etiqueta *NFC* que sea compatible con el dispositivo en el que se ejecuta la *herramienta de administración de contenidos vía NFC*,
- `WriteNFCTag`: graba la información dada en una etiqueta *NFC* que sea compatible con el dispositivo en el que se ejecuta la *herramienta de administración de contenidos vía NFC*,
- `GetInstalledApps`: obtiene la lista de todas las aplicaciones instaladas en el dispositivo en el que se ejecuta la *herramienta de administración de contenidos vía NFC*,
- `GetDeviceFreeSpace`: obtiene el espacio libre del dispositivo en el que se ejecuta la *herramienta de administración de contenidos vía NFC*,
- `CheckNetworkStatus`: determina si existe una conexión a Internet en el dispositivo en el que se ejecuta la *herramienta de administración de contenidos vía NFC* y si el adaptador de Wi-Fi está encendido o apagado, y
- `TurnOnWiFiAdapter`: enciende el adaptador de Wi-Fi del dispositivo en el que se ejecuta la *herramienta de administración de contenidos vía NFC*.

6.7. Variables contextuales consideradas

La Tabla 6.6 muestra las entidades y sus respectivas variables contextuales de la herramienta *Administrador de contenidos vía NFC*.

Cuadro 6.6: Entidades y sus variables contextuales de la herramienta *administrador de contenidos vía NFC*

Entidad	**Variables contextuales**
	Tiempo
Archivo	Formato, y tamaño
Dispositivo	Espacio libre de almacenamiento, y software disponible (visores).
Usuario	Dispositivo en uso, y rol

El **tiempo** se considera para: 1) evitar registrar reuniones cuya fecha sea anterior al día actual, 2) evitar acceder a reuniones cuya fecha sea anterior al día actual, y 3) opcionalmente, se puede ejecutar en el servidor un programa para eliminar todas las reuniones cuya fecha sea anterior al día actual. El **formato** y **tamaño** de cada uno de los archivos asociados a una reunión se consideran para mostrar una versión pertinente en el dispositivo de cada colaborador, tomando en cuenta el **software disponible** y el **espacio libre de almacenamiento** del dispositivo en el que se ejecuta la *herramienta de administración de contenidos vías NFC*. Finalmente el usuario, dependiendo de su **rol**, podrá visualizar, editar o eliminar una reunión.

6.8. Eventos contextuales considerados

La *herramienta de administración de contenidos vía NFC* utiliza los eventos contextuales:

Cuadro 6.7: Eventos contextuales considerados por la *herramienta de administración de contenidos vía NFC*

Evento contextual	Tipo	Fuente
The `$DEVICE` has read a NFC tag	Lógico	Servicio `ReadNFCTag`
The user `$USER` views the meeting `$MEETING`	Lógico	Servicio `ShowMeeting`
The user `$USER` has stopped viewing the meeting `$MEETING`.	Lógico	Servicio `ShowMeeting`
The meeting `$MEETING` is being edited	Lógico	Servicio `EditMeeting`

El primer evento contextual "***The device $DEVICE has read a NFC tag***" se activa cuando un usuario, que previamente ha iniciado sesión en la *herramienta de administración de contenidos vía NFC*, acerque su dispositivo mientras esté en la actividad *Lectura de una etiqueta NFC*, independientemente del contenido de dicha etiqueta *NFC*. Sus parámetros asociados son `$TAG_CONTENT` y `$DEVICE`, que corresponden al contenido de la etiqueta *NFC* leída y el identificador del dispositivo, respectivamente.

El segundo evento contextual, "***The $USER is viewing the meeting $MEETING***" se activa cuando un usuario, que previamente ha iniciado sesión en la *herramienta de administración de contenidos vía NFC*, visualice en su dispositivo una reunión, i.e., cuando ingrese a la actividad *Consulta de una reunión*. El tercer evento contextual, "***The user $USER has stopped viewing the meeting $MEETING***", se activa cuando un usuario, que previamente ha iniciado sesión en la *herramienta de administración de contenidos vía NFC*, deje de visualizar una reunión, i.e., cuando deje la actividad *Consulta de una reunión*. Sus parámetros asociados son `$USER` y `$MEETING`, que corresponden a los identificadores del usuario y de la reunión que dicho usuario ha empezado o dejado de ver, respectivamente.

Por último, el último evento contextual "***The meeting $MEETING is being edited***" se activa cuando un usuario, que previamente ha iniciado sesión en la *herramienta de admi-*

nistración de contenidos vía NFC, modifique los datos de una reunión, i.e., cuando ingrese a la actividad *Edición de una reunión*. Dispone de un único parámetro asociado, `$MEETING`, que corresponde al identificador de la reunión que está siendo editada.

6.9. Situaciones y adaptaciones consideradas

La siguiente lista muestra las situaciones consideradas por la *herramienta de administración de contenidos vía NFC*:

1. Un colaborador que no ha iniciado sesión acerca su dispositivo a una etiqueta *NFC*.
2. Un colaborador que ha iniciado sesión acerca su dispositivo a una etiqueta *NFC* vacía.
3. Un colaborador que ha iniciado sesión solicita visualizar una reunión inexistente.
4. Un colaborador que ha iniciado sesión no está autorizado para visualizar una reunión existente.
5. El dispositivo de un colaborador autorizado a visualizar una reunión no tiene suficiente espacio para la descarga de los documentos asociados de la reunión.
6. El dispositivo de un colaborador autorizado a visualizar una reunión tiene espacio suficiente para la descarga de los documentos asociados a la reunión pero no cuenta con un visor integrado para visualizar cierto formato de documento.
7. El dispositivo de un colaborador autorizado a visualizar una reunión tiene espacio suficiente para la descarga de los documentos asociados a la reunión y puede visualizar todos los formatos de los documentos asociados a la reunión.
8. Ejecución de la reunión.

Las situaciones primera, segunda, tercera, cuarta, quinta, sexta, séptima y octava son transitorias, en cambio, la novena situación es permanente. Seguidamente, se detalla cada situación

y las adaptaciones pertinentes.

1. **Un colaborador que no ha iniciado sesión acerca su dispositivo a una etiqueta *NFC***

Name	NOT_LOGGED_IN
Type	Transient
Listeners	1) `$DEVICE` that read the NFC tag.
Contextual events	1) The `$DEVICE` has read an NFC tag.
Conditions	1) The `$USER` is not logged in.

La función de lectura de una etiqueta *NFC* sólo puede ser utilizada en la ventana *Lectura de una etiqueta NFC*, accesible una vez que el usuario ha iniciado sesión en la aplicación. De esta forma, mientras el usuario no haya iniciado sesión. i.e., la variable de sesión `user`, el servicio `ReadNFCTag` permanece deshabilitado, y por ello el usuario no podrá hacer uso de la información almacenada en la etiqueta *NFC*.

2. **Un colaborador que ha iniciado sesión acerca su dispositivo a una etiqueta *NFC* vacía**

Name	EMPTY_NFC_TAG
Type	Transient
Listeners	1) `$DEVICE` that read the NFC tag.
Contextual events	1) The `$DEVICE` has read an NFC tag.
Conditions	1) NFC tag is empty.

Cuando un colaborador, que previamente ha iniciado sesión en la aplicación *NFC Meeting*, acerque su dispositivo a una etiqueta *NFC* vacía, ésta le propondrá registrar una reunión nueva. En caso afirmativo, se desplegará la ventana *Registro de una nueva reunión*, se guardará la reunión y sus documentos en el servidor de contenidos, y en la etiqueta *NFC* su identificador asociado (véase Figura 6.6).

3. **Un colaborador que ha iniciado sesión solicita visualizar una reunión inexistente**

La aplicación *NFC Meeting* le mostrará un mensaje de error indicando que no la reunión cuyo identificador ha leído de una etiqueta *NFC* no existe (véase la Figura 6.7).

Name	MEETING_DOES_NOT_EXIST
Type	Transient
Listeners	`$DEVICE` that read the NFC tag.
Contextual events	The `$DEVICE` has read an NFC tag.
Conditions	1) `$USER` is logged in, *and* 2) `$MEETING` does not exists.

4. **Un colaborador que ha iniciado sesión no está autorizado para visualizar una reunión existente**

Name	NOT_AUTHORIZED_TO_VIEW_MEETING
Type	Transient
Listeners	`$DEVICE` that read the NFC tag.
Contextual events	The `$DEVICE` has read an NFC tag.
Conditions	1) `$USER` is logged in, 2) `$MEETING` exists, *and* 3) `$USER` is not approved to view the `$MEETING`.

La aplicación *NFC Meeting* le mostrará un mensaje error indicando que no está autorizado para visualizar la correspondiente reunión (véase Figura 6.8).

5. **Un colaborador edita una reunión**

Name	EDITION_OF_A_MEETING
Type	Transient
Listeners	`$USERS` viewing the meeting `$MEETING`.
Contextual events	1) The meeting `$MEETING` is being edited.
Conditions	-

La función de visualización de una reunión es utilizada desde la ventana *Consulta de una reunión*, mediante el servicio `ShowMeeting`, y la función de edición de una reunión es utilizada desde la ventana *Edición de una reunión*, mediante el servicio `EditMeeting`. Para evitar que los colaboradores visualicen una versión errónea de una reunión, cuando un colaborador edita una reunión, i.e. ejecuta el servicio `EditMeeting`, evita que otro colaborador visualice dicha reunión, i.e., el servicio `ShowMeeting` evitará que se muestra en pantalla la correspondiente reunión (véase la Figura 6.9).

6. **El dispositivo de un colaborador autorizado a visualizar una reunión no tiene suficiente espacio para la descarga de los documentos asociados de la reunión**

Name	DEVICE_HAS_NOT_ENOUGH_SPACE
Type	Transient
Listeners	1) `$DEVICE` that read the NFC tag, and 2) Service `ShowMeeting`.
Contextual events	The `$DEVICE` has read an NFC tag.
Conditions	1) `$USER` is logged in, 2) `$MEETING` exists, 3) `$USER` is approved to view the `$MEETING`, *and* 4) `$DEVICE` has not enough space.

Ante esta situación, la aplicación *NFC Meeting* le mostrará al colaborador los documentos dentro de la aplicación en sus versiones JPEG, para ello usará el servicio `ConvertDocument`. La Figura 6.10 muestra la ventana *Consulta de una reunión* con varios documentos en su versión JPEG.

7. **El dispositivo de un colaborador autorizado a visualizar una reunión tiene espacio suficiente para la descarga de los documentos asociados a la reunión pero no cuenta con un visor integrado para visualizar cierto formato de documento**

Name	DEVICE_HAS_ENOUGH_SPACE_AND_DOES_NOT _SUPPORT_ALL_FORMATS
Type	Transient
Listeners	1) `$DEVICE` that read the NFC tag, 2) Service `ShowMeeting`.
Contextual events	The `$DEVICE` has read an NFC tag.
Conditions	1) `$USER` is logged in, 2) `$MEETING` exists, 3) `$USER` is approved to view the `$MEETING`. 4) `$DEVICE` has enough free space, *and* 5) `$DEVICE` does not support all formats.

Ante esta situación, la aplicación *NFC Meeting* le permitirá al colaborador descargar las versiones JPEG de los documentos cuyo formato no es soportado por los visores instalados en el dispositivo Android, ya que el sistema operativo Android soporta este formato nativamente. La Figura 6.11 muestra la ventana *Consulta de una reunión* con el enlace de varios documentos en su versión JPEG.

8. **El dispositivo de un colaborador autorizado a visualizar una reunión tiene espacio suficiente para la descarga de los documentos asociados a la reunión y puede visualizar todos los formatos de los documentos asociados a la reunión**

Name	DEVICE_HAS_ENOUGH_SPACE_AND_SUPPORTS_ALL _FORMATS
Type	Transient
Listeners	1) `$DEVICE` that read the NFC tag, 2) Service `ShowMeeting`.
Contextual events	The `$DEVICE` has read an NFC tag.
Conditions	1) `$USER` is logged in, 2) `$MEETING` exists, 3) `$USER` is approved to view the `$MEETING`, 4) `$USER` has enough free space, *and* 5) `$DEVICE` supports all formats.

Ante esta situación, la aplicación *NFC Meeting* le permitirá al colaborador descargar los documentos en sus formatos originales. La Figura 6.12 muestra la venta *Consulta de una reunión* con el enlace de varios documentos en sus versiones originales.

9. **Ejecución de la reunión**

Name	EXECUTION_OF_THE_MEETING
Type	Permanent
Related entity	`$MEETING`
Listeners	`$USERS` viewing the `$MEETING`.
Contextual events	1) The user `$USER` views the meeting $MEETING, *and* 2) The user `$USER` has stopped viewing the meeting `$MEETING`.
Conditions	1) The number of users viewing a meeting `$MEETING` is at least half of those registered.

Ante esta situación, la aplicación *NFC Meeting* destacará los colaboradores que están observando la reunión en cuestión, los mostrará en verde en la ventana *Consulta de una reunión*, en la sección *Asistentes* (véase la Figura 6.13).

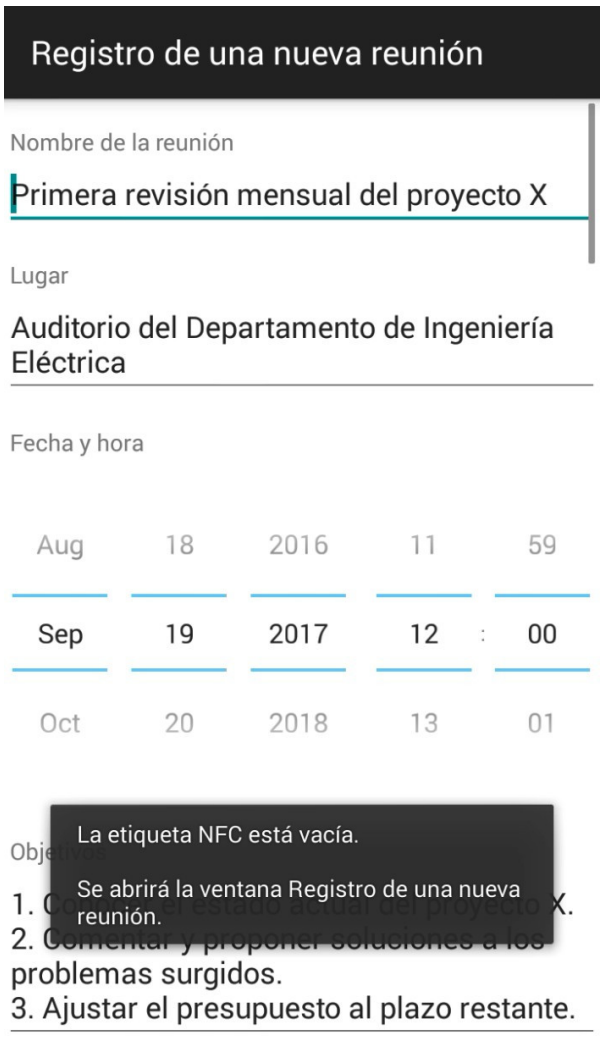
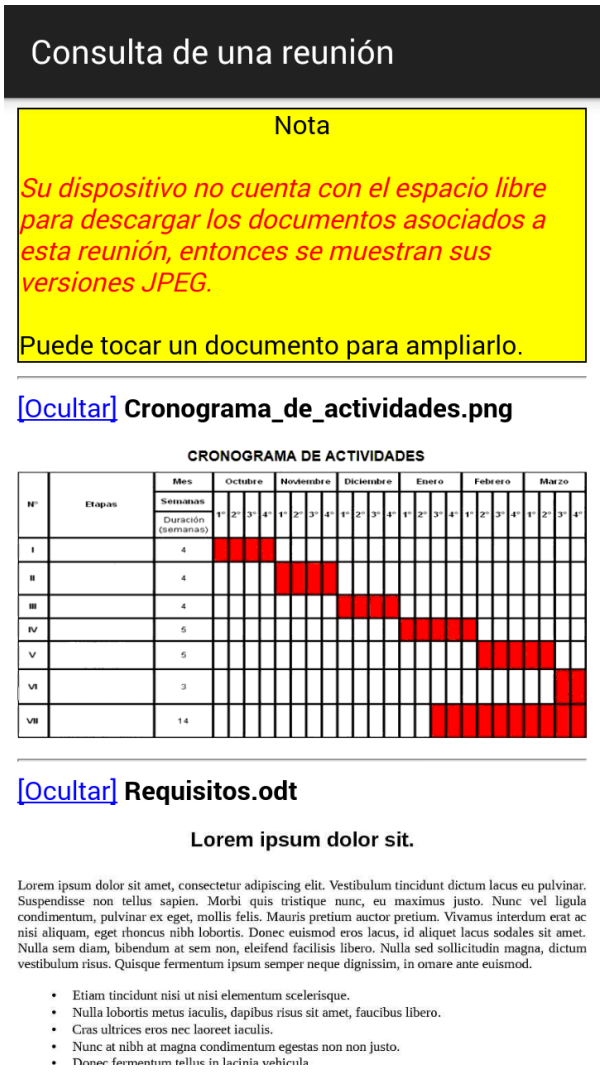


Figura 6.6: Mensaje para una etiqueta *NFC* inexistente

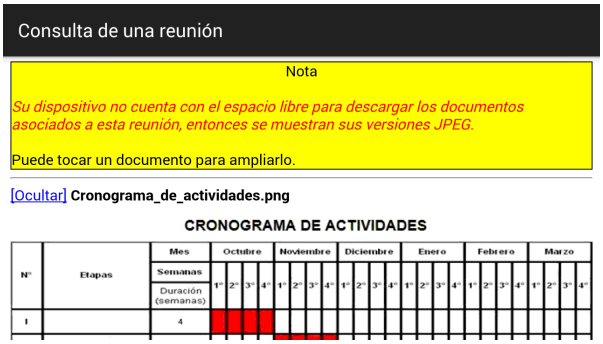
Figura 6.7: Mensaje de error para una reunión inexistente

Figura 6.8: Mensaje de error para un colaborador no autorizado a visualizar una reunión existente

Figura 6.9: Mensaje de error al visualizar una reunión que está siendo editada



Modo vertical



Modo horizontal

Figura 6.10: Ventana *Consulta de una reunión* con los documentos en sus versiones JPEG

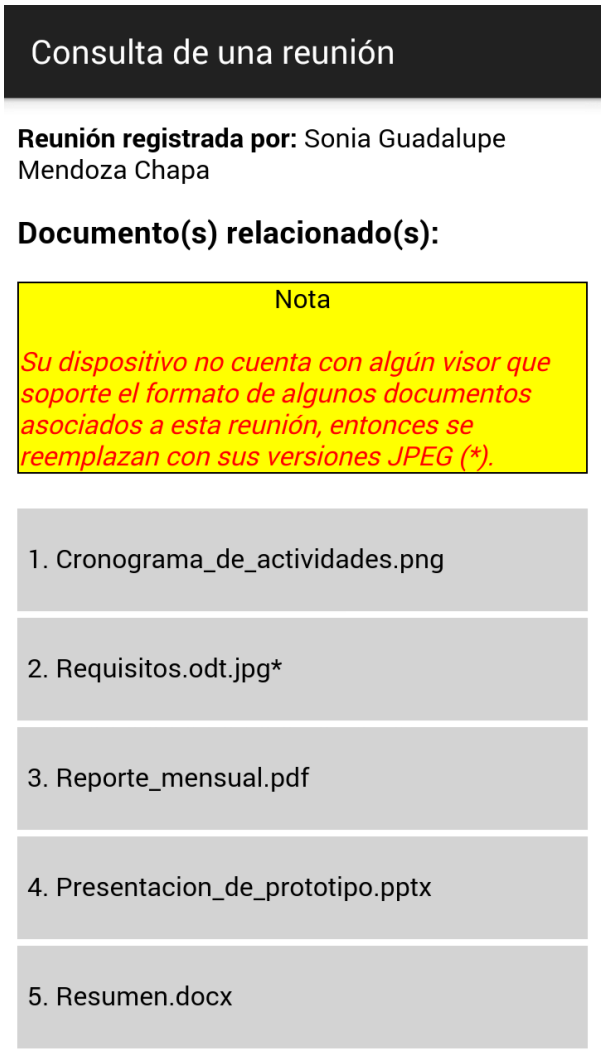


Figura 6.11: Ventana *Consulta de una reunión* con los enlaces de descarga de los documentos en sus versiones JPEG

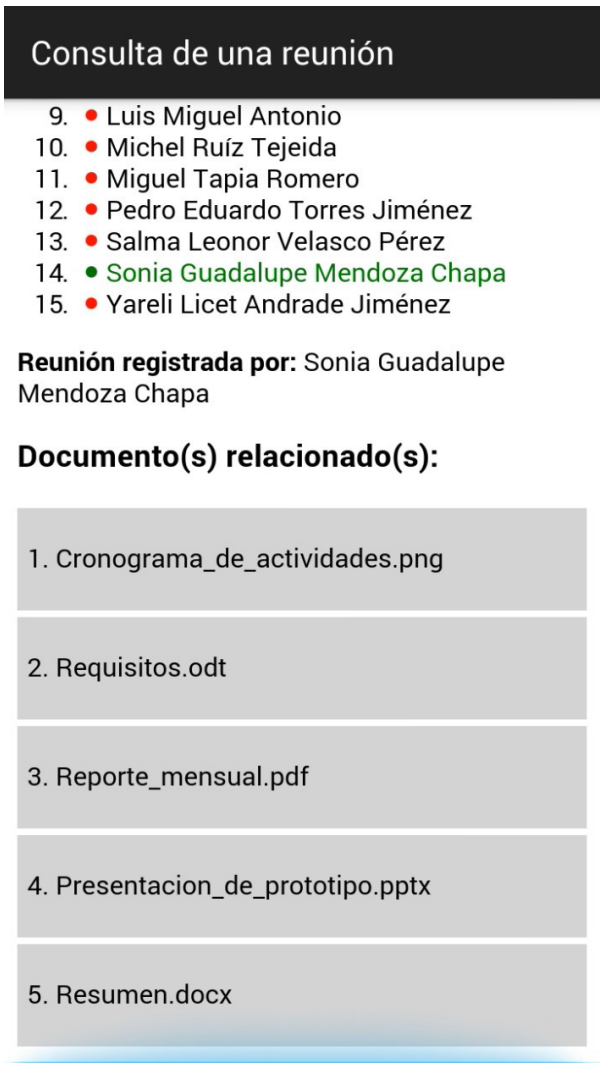


Figura 6.12: Ventana *Consulta de una reunión* con los enlaces de descarga de los documentos en sus versiones originales

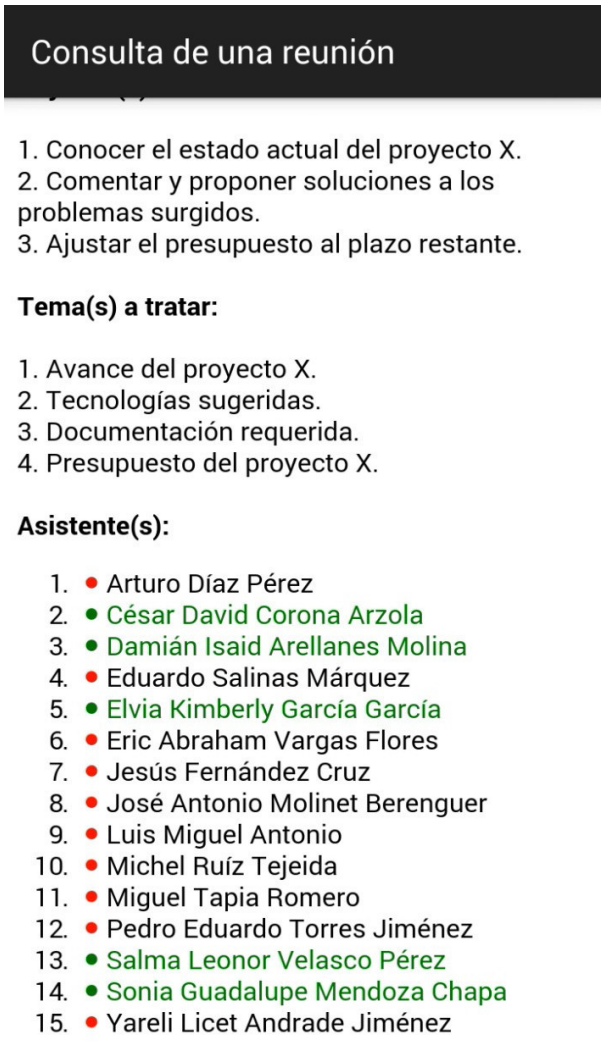


Figura 6.13: Ventana *Consulta de una reunión* con los asistentes observando una reunión

6.10. Pruebas realizadas

En esta sección se describen las pruebas a las que fue sometida la aplicación *NFC Meeting*, para verificar su comportamiento ante distintas situaciones, además se describe el flujo del programa y los resultados obtenidos. Para ello se utilizo un dispositivo Samsung Galaxy SIII con Android 4.3 Kitkat.

1. **Lectura de una etiqueta *NFC* vacía**

Se inició sesión en la aplicación con los datos del usuario "Sonia Guadalupe Mendoza Chapa" y en la ventana *Lectura de una etiqueta NFC*, el dispositivo se acercó a una etiqueta *NFC* vacía. Se ejecutó el servicio `ReadNFCTag` y se produjo el evento contextual "*A device $DEVICE has read a NFC tag*", cuyos los parámetros asociados `$TAG_CONTENT` y `$DEVICE` corresponden al contenido de la etiqueta *NFC* (nulo) y al identificador del dispositivo, respectivamente.

El servicio contextual evaluó las situaciones asociadas al evento, y se determinó presente la situación transitoria "*EMPTY_NFC_TAG*", ya que se satisfacen sus condiciones 1) *$USER is logged in*, y 2) *NFC tag is empty*. Entonces, el servidor comunicó su presencia a los escuchas interesados, el dispositivo que leyó la etiqueta *NFC*, i.e. *$DEVICE that read the NFC tag*. El dispositivo mostró la ventana de *Registro de una nueva reunión* (véase la Figura 6.14).

2. **Lectura de una etiqueta *NFC* con un identificador inválido**

A continuación, se escribió un identificador inválido, "M_A", en una etiqueta *NFC* con la aplicación para Android *NFCMeesageBoard*. Se inició sesión en la aplicación con los dartos del usuario "Sonia Guadalupe Mendoza Chapa" y en la ventana *Lectura de una etiqueta NFC*, el dispositivo se acercó a dicha etiqueta. Se ejecutó el servicio `ReadNFCTag` y se produjo el evento contextual *The $DEVICE has read a NFC tag*, cuyos parámetros asociados `$TAG_CONTENT` y `$DEVICE` corresponden al contenido de la etiqueta *NFC* (el identificador inválido) y al identificador del dispositivo, respectivamente.

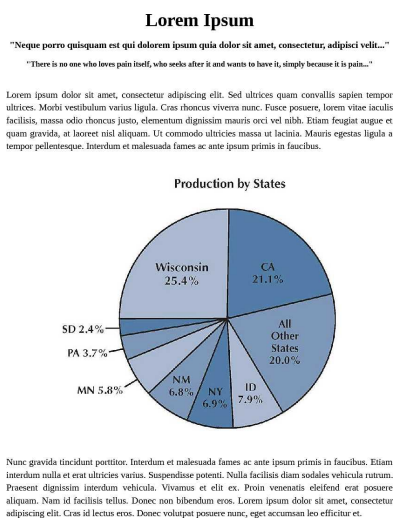
El servicio contextual evaluó las situaciones asociadas al evento, y se determinó presente la situación transitoria "*MEETING_DOES_NOT_EXIST*", ya que se satisfacen sus condiciones: 1) *$USER is logged in*, y 2) *$MEETING does not exists*. Entonces, el servidor comunicó su presencia a los escuchas interesados, en este caso, al dispositivo que leyó la etiqueta *NFC*, i.e. *$DEVICE that read the NFC tag*. El dispositivo mostró un mensaje de error indicando que la reunión solicitada no existe (véase la Figura 6.15).

3. **Registro de una reunión**

Se inició sesión con los datos del usuario “Sonia Guadalupe Mendoza Chapa” y se accedió a la ventana *Registro de una nueva reunión*, entonces se ejecutó el servicio `RegisterMeeting`. Los datos de la reunión fueron los siguientes:

Nombre:	Primera revisión mensual del proyecto X
Fecha:	2017/09/19
Objetivo:	1. Conocer el estado actual del proyexto X. 2. Comentar y proponer soluciones a los problemas surgidos. 3. Ajustar el presupuesto al plazo restante.
Puntos a tratar:	1. Avance del proyecto X. 2. Tecnologías sugeridas. 3. Documentación requerida. 4. Presupuesto del proyecto X.
Asistentes:	Andrade Jimenez Yareli Licet, Fernandez Cruz Jesus, y Miguel Antonio Luis.

Y se seleccionaron los siguientes documentos:



Lorem Ipsum

"Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit..."

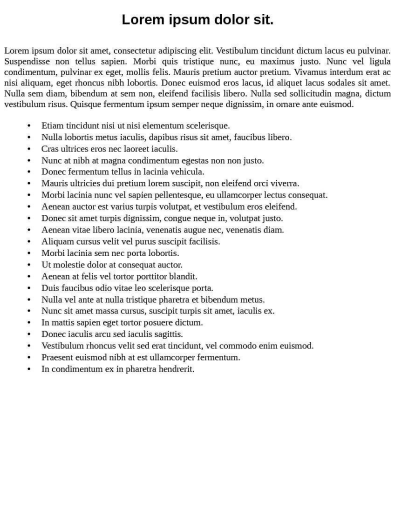
"There is no one who loves pain itself, who seeks after it and wants to have it, simply because it is pain..."

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed ultrices quam convallis sapien tempor ultrices. Morbi vestibulum varius ligula. Cras rhoncus viverra nunc. Fusce posuere, lorem vitae iaculis facilisis, massa odio rhoncus justo, elementum dignissim mauris orci vel nibh. Etiam feugiat augue et quam gravida, at laoreet nisl aliquam. Ut commodo ultricies massa ut lacinia. Mauris egestas ligula a tempor pellentesque. Interdum et malesuada fames ac ante ipsum primis in faucibus.

Production by States

Nunc gravida tincidunt porttitor. Interdum et malesuada fames ac ante ipsum primis in faucibus. Etiam interdum nulla et erat ultricies varius. Suspendisse potenti. Nulla facilisis diam sodales vehicula rutrum. Praesent dignissim interdum vehicula. Vivamus et elit ex. Proin venenatis eleifend erat posuere aliquam. Nam id facilisis tellus. Donec non bibendum eros. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras id lectus eros. Donec volutpat posuere nunc, eget accumsan leo efficitur et.

Reporte mensual.pdf

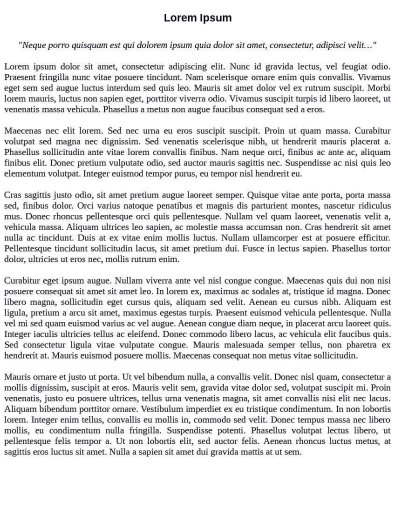


Lorem ipsum dolor sit.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum tincidunt dictum lacus eu pulvinar. Suspendisse non tellus sapien. Morbi quis tristique nunc, eu maximus justo. Nunc vel ligula condimentum, pulvinar ex eget, mollis felis. Mauris pretium auctor pretium. Vivamus interdum erat ac nisi aliquam, eget rhoncus nibh lobortis. Donec euismod eros lacus, id aliquet lacus sodales sit amet. Nulla sem diam, bibendum at sem non, eleifend facilisis libero. Nulla sed sollicitudin magna, dictum vestibulum risus. Quisque fermentum ipsum semper neque dignissim, in ornare ante euismod.

- Etiam tincidunt nisi ut nisi elementum scelerisque.
- Nulla lobortis metus iaculis, dapibus risus sit amet, faucibus libero.
- Cras ultrices eros nec laoreet iaculis.
- Nunc at nibh at magna condimentum egestas non non justo.
- Donec fermentum tellus in lacinia vehicula.
- Mauris ultricies dui pretium lorem suscipit, non eleifend orci viverra.
- Morbi lacinia nunc vel sapien pellentesque, eu ullamcorper lectus consequat.
- Aenean auctor est varius turpis volutpat, et vestibulum eros eleifend.
- Donec sit amet turpis dignissim, congue neque in, volutpat justo.
- Aenean vitae libero lacinia, venenatis augue nec, venenatis diam.
- Aliquam cursus velit vel purus suscipit facilisis.
- Morbi lacinia sem nec porta lobortis.
- Ut molestie dolor at consequat auctor.
- Aenean at felis vel tortor porttitor blandit.
- Duis faucibus odio vitae leo scelerisque porta.
- Nulla vel ante at nulla tristique pharetra et bibendum metus.
- Nunc sit amet massa cursus, suscipit turpis sit amet, iaculis ex.
- In mattis sapien eget tortor posuere dictum.
- Donec iaculis arcu sed iaculis sagittis.
- Vestibulum rhoncus velit sed erat tincidunt, vel commodo enim euismod.
- Praesent euismod nibh at est ullamcorper fermentum.
- In condimentum ex in pharetra hendrerit.

Requisitos.odt



Lorem Ipsum

"Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit..."

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc id gravida lectus, vel feugiat odio. Praesent fringilla nunc vitae posuere tincidunt. Nam scelerisque ornare enim quis convallis. Vivamus eget sem sed augue luctus interdum sed quis leo. Mauris sit amet dolor vel ex rutrum suscipit. Morbi lorem mauris, luctus non sapien eget, porttitor viverra odio. Vivamus suscipit turpis id libero laoreet, ut venenatis massa vehicula. Phasellus a metus non augue faucibus consequat sed a eros.

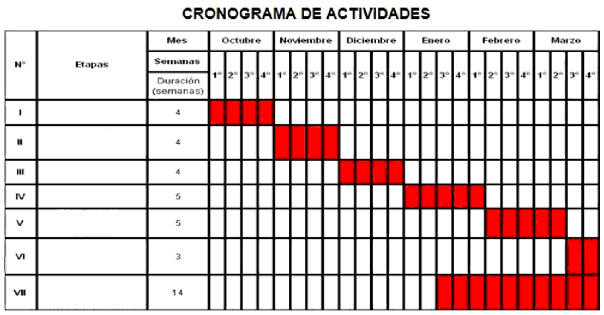
Maecenas nec elit lorem. Sed nec urna eu eros suscipit suscipit. Proin ut quam massa. Curabitur volutpat sed magna nec dignissim. Sed venenatis scelerisque nibh, ut hendrerit mauris placerat a. Phasellus sollicitudin ante vitae lorem convallis finibus. Nam neque orci, finibus ac ante ac, aliquam finibus elit. Donec pretium vulputate odio, sed auctor mauris sagittis nec. Suspendisse ac nisi quis leo elementum volutpat. Integer euismod tempor purus, eu tempor nisl hendrerit eu.

Cras sagittis justo odio, sit amet pretium augue laoreet semper. Quisque vitae ante porta, porta massa sed, finibus dolor. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec rhoncus pellentesque orci quis pellentesque. Nullam vel quam laoreet, venenatis velit a, vehicula massa. Aliquam ultrices leo sapien, ac molestie massa accumsan non. Cras hendrerit sit amet nulla ac tincidunt. Duis at ex vitae enim mollis luctus. Nullam ullamcorper est at posuere efficitur. Pellentesque tincidunt sollicitudin lacus, sit amet pretium dui. Fusce in lectus sapien. Phasellus tortor dolor, ultricies ut eros nec, mollis rutrum enim.

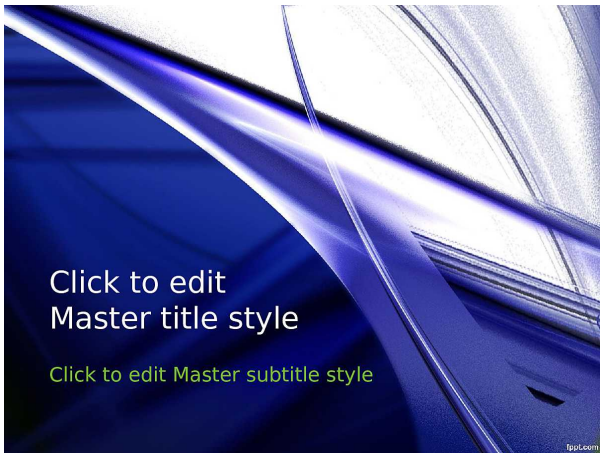
Curabitur eget ipsum augue. Nullam viverra ante vel nisl congue congue. Maecenas quis dui non nisi posuere consequat sit amet sit amet leo. In lorem ex, maximus ac sodales at, tristique id magna. Donec libero magna, sollicitudin eget cursus quis, aliquam sed velit. Aenean eu cursus nibh. Aliquam est ligula, pretium a arcu sit amet, maximus egestas turpis. Praesent euismod vehicula pellentesque. Nulla vel mi sed quam euismod varius ac vel augue. Aenean congue diam neque, in placerat arcu laoreet quis. Integer iaculis ultricies tellus ac eleifend. Donec commodo libero lacus, ac vehicula elit faucibus quis. Sed consectetur ligula vitae vulputate congue. Mauris malesuada semper tellus, non pharetra ex hendrerit at. Mauris euismod posuere mollis. Maecenas consequat non metus vitae sollicitudin.

Mauris ornare et justo ut porta. Ut vel bibendum nulla, a convallis velit. Donec nisl quam, consectetur a mollis dignissim, suscipit at eros. Mauris velit sem, gravida vitae dolor sed, volutpat suscipit mi. Proin venenatis, justo eu posuere ultrices, tellus urna venenatis magna, sit amet convallis nisi elit nec lacus. Aliquam bibendum porttitor ornare. Vestibulum imperdiet ex eu tristique condimentum. In non lobortis lorem. Integer enim tellus, convallis eu mollis in, commodo sed velit. Donec tempus massa nec libero mollis, eu condimentum nulla fringilla. Suspendisse potenti. Phasellus volutpat lectus libero, ut pellentesque felis tempor a. Ut non lobortis elit, sed auctor felis. Aenean rhoncus luctus metus, at sagittis eros luctus sit amet. Nulla a sapien sit amet dui gravida mattis at ut sem.

Resumen.docx



Cronogramas de actividades.png



Presentación de prototipo.pptx

El servicio `RegisterMeeting` se conectó al servidor, cargó los documentos seleccionados e invocó al servicio `QueryOntology` para realizar las consultas SPARQL para crear el individuo de la reunión, mediante las funciones `addDataProperty` (véase la Tabla 6.8) y `addObjectProperty` (véase la Tabla 6.9), i.e. instancia de la clase `Meeting` (véase la Tabla 6.10), los individuos para cada documento, i.e. instancias de `Document`, y las propiedades de objeto `hasAssistant` y `hasMeeting` (véase la Tabla 6.11) para relacionar la reunión con sus asistentes, i.e. instancias de la clase `User`, y con sus documentos.

Cuadro 6.8: Función para crear una propiedad de dato

```
 1:  public void addDataProperty(
 2:   OWLNamedIndividual individual,
 3:   OWLOntologyManager manager,
 4:   OWLOntology ontology,
 5:   String aux_IRI,
 6:   String propertyId,
 7:   String propertyValue
 8:  ) throws OWLOntologyCreationException,
 9:  OWLOntologyStorageException {
10:
11:    OWLDataFactory factory = OWLManager.getOWLDataFactory();
12:    OWLDataProperty OWLDP = factory.getOWLDataProperty(
```

```
13:     IRI.create(aux_IRI + 'propertyId')
14:    );
15:
16:    OWLDataPropertyAssertionAxiom OWLDPAA =
17:     factory.getOWLDataPropertyAssertionAxiom( OWLDP,
18:     individual, propertyValue);
19:    manager.addAxiom(ontology, OWLDPAA);
20:  }
```

Cuadro 6.9: Función para crear una propiedad de objeto

```
 1:  public void addObjectProperty(
 2:   String propertyName,
 3:   OWLOntologyManager manager,
 4:   OWLOntology ontology,
 5:   String IRI,
 6:   OWLIndividual individual1,
 7:   OWLIndividual individual2
 8:  ) throws OWLOntologyCreationException,
 9:  OWLOntologyStorageException {
10:    OWLDataFactory factory = OWLManager.getOWLDataFactory();
11:    OWLObjectPropertyAssertionAxiom assertion;
12:    OWLObjectProperty OWLproperty;
13:    OWLproperty = factory.getOWLObjectProperty(IRI.create(IRI +
14:     propertyName);
15:    assertion = factory.getOWLObjectPropertyAssertionAxiom(
16:     OWLproperty, individual1, individual2);
17:     manager.addAxiom(ontology, assertion);
18:  }
```

Cuadro 6.10: Registro de un reunión

```
1:  // Se crea el individuo.
2:  OWLNamedIndividual newMeeting =
3:   factory.getOWLNamedIndividual(
4:   IRI.create(aux_IRI + 'Meeting' + meetingId) );
5:
6:  // Se crean las propiedades de datos
```

```
 7: // ...generadas por el servidor.
 8: // meetingId
 9: addDataProperty( newMeeting, manager, ontology,
10:  aux_IRI + 'meetingId', 'meetingId', meetingId);
11:
12: // ... y las registradas por el usuario.
13: // meetingName
14: addDataProperty( newMeeting, manager, ontology,
15:  aux_IRI + 'meetingId', 'meetingName', meetingName);
16: // meetingPlace
17: addDataProperty( newMeeting, manager, ontology,
18:  aux_IRI + 'meetingId', 'meetingPlace', meetingPlace);
19: // meetingDate
20: addDataProperty( newMeeting, manager, ontology,
21:  aux_IRI + 'meetingId', 'meetingDate', meetingDate);
22: // meetingTime
23: addDataProperty( newMeeting, manager, ontology,
24:  aux_IRI + 'meetingId', 'meetingTime', meetingTime);
25: // meetingGoals
26: addDataProperty( newMeeting, manager, ontology,
27:  aux_IRI + 'meetingId', 'meetingGoals', meetingGoals);
28: // meetingThemes
29: addDataProperty( newMeeting, manager, ontology,
30:  aux_IRI + 'meetingId', 'meetingThemes', meetingThemes);
31: // meetingAssistants
32: addDataProperty( newMeeting, manager, ontology,
33:  aux_IRI + 'meetingId', 'meetingAssistants',
34:  meetingAssistants);
35: // meetingAuthor
36: addDataProperty( newMeeting, manager, ontology,
37:  aux_IRI + 'meetingId', 'meetingAuthor', meetingAuthor);
38:
39: // Se especifica que el individuo <newMeeting> es una
40: // instancia de la clase <Meeting>.
41: OWLClass _MeetingClass = factory.getOWLClass(
42:  IRI.create(aux_IRI + 'Meeting') );
43: OWLClassAssertionAxiom _instanceAxiom =
44:  factory.getOWLClassAssertionAxiom(_MeetingClass, newMeeting);
45: manager.addAxiom(ontology, _instanceAxiom);
46:
47: manager.saveOntology(ontology);
```

Cuadro 6.11: Creación de las propiedades de objeto `hasAssistant` y `hasDocument`

```
/* Creación de la propiedad de objeto hasAssistant */
OWLIndividual meeting = dataFactory.getOWLNamedIndividual(
 IRI.create(base + meetingId));

OWLIndividual assistant = dataFactory.getOWLNamedIndividual(
 IRI.create(base + assistantName));

addObjectProperty(
 'hasAssistant', manager, ontology, IRI meeting, assistant);

/* Creación de la propiedad de objeto hasDocument */
OWLIndividual document = dataFactory.getOWLNamedIndividual(
 IRI.create(base + documentId));

addObjectProperty('hasDocument', manager, ontology, IRI, mee-
ting, document);
```

Finalmente, el servidor devolvió el identificador de la reunión, i.e., el número de reunión con el prefijo "M_" y, mediante el servicio `WriteNFCTag` éste se almacenó en una etiqueta *NFC*.

4. **Visualización de una reunión (lectura de una etiqueta *NFC* con un identificador válido)**

En la ventana *Lectura de una etiqueta NFC*, cuando el dispositivo se acerca a una etiqueta *NFC* se ejecuta el servicio `ReadNFCTag` y se produce el evento contextual "*The `$DEVICE` has read a NFC tag*", cuyos parámetros asociados `$TAG_CONTENT` y `$DEVICE` corresponden al contenido de la etiqueta *NFC* y al identificador del dispositivo, respectivamente.

El servicio contextual utiliza el servicio `QueryOntology` para realizar las consultas SPARQL para obtener los datos de una reunión y los visores (véanse las Tablas 6.12 y 6.13) para evaluar las condiciones de las situaciones.

Cuadro 6.12: Consulta SPARQL para obtener los formatos compatibles de una reunión

```
String query = 'PREFIX rdf: <http://www.w3.org/1999/02/22
 -rdf-syntax-ns#>'
  + 'PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>'
  + 'PREFIX owl: <http://www.w3.org/2002/07/owl#>'
  + 'PREFIX c: <http://www.owl-ontologies.com/ontContext.owl#>'
  + 'SELECT ?Z ?meetingName ?meetingDate ?meetingGoals
    ?meetingThemes {'
  + '?Z c:meetingId \'' + parameter +'\' . ' // meetingId
  + '?Z c:meetingName ?meetingName . '
  + '?Z c:meetingDate ?meetingDate . '
  + '?Z c:meetingGoals ?meetingGoals . '
  + '?Z c:meetingThemes ?meetingThemes'
  + '}';
```

Cuadro 6.13: Consulta SPARQL para obtener los formatos compatibles de un visor

```
String query = 'PREFIX rdf: <http://www.w3.org/1999/02/22-
rdf-syntax-ns#>'
  + 'PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>'
  + 'PREFIX owl: <http://www.w3.org/2002/07/owl#>'
  + 'PREFIX v: <http://www.owl-ontologies.com/virtualR.owl#>'
  + 'SELECT ?Z ?N ?U'
  + 'WHERE {'
  + '{?Z rdf:type v:Viewer . }'
  + '?Z v:name ?N .'
  + '?Z v:compatibleFormats ?U .'
  + 'FILTER regex(?U, \'+ viewerName +\', \'i\')'
  + '}';
```

Dependiendo de las condiciones, puede activarse una de las siguientes situaciones transitorias:

A. **Situación *NOT_AUTHORIZED_TO_VIEW_MEETING***

Se inició sesión en la aplicación con los datos del usuario "Garcia Garcia Elvia Kimberly" y en la ventana *Lectura de una etiqueta NFC*, el dispositivo se acercó a la etiqueta *NFC* con

el identificador de la reunión previamente registrada. Se ejecutó el servicio `ReadNFCTag` y se produjo el evento contextual "*The $DEVICE has read a NFC tag*", cuyos parámetros asociados son `$TAG_CONTENT` y `$DEVICE` corresponden a los identificadores de la reunión previamente registrada y del teléfono del usuario, respectivamente.

El servicio contextual evaluó las situaciones asociadas al evento, y se determinó presente la situación "*NOT_AUTHORIZED_TO_VIEW_MEETING*", ya que se satisfacen sus condiciones:

- *$USER is logged in*: la sesión del usuario "Garcia Garcia ELvia Kimberly" estaba activa,
- *$MEETING exists*: existía una reunión cuyo identificador corresponde a `$TAG_CONTENT`, y
- *$USER is not approved to view the $MEETING*: el usuario "Garcia Garcia Elvia Kimberly" no registró la correspondiente reunión, i.e. no tiene el rol `Author`, ni estaba registrada como asistente a la misma, i.e. no tiene el rol `Assistant`.

Entonces, el servidor comunicó la presencia de tal situación a los escuchas interesados, el dispositivo que leyó la etiqueta *NFC*, i.e. *$DEVICE that read the NFC tag*. El dispositivo mostró un mensaje de error indicando que el colaborador no está autorizado para visualizar la reunión (véase la Figura 6.17).

B. **Situación *DEVICE_HAS_NOT_ENOUGH_SPACE***

Se llenó el espacio libre del dispositivo generando archivos a través del comando `dd` del sistema operativo GNU-Linux.

A continuación, se inició sesión en la aplicación con los datos del usuario "Andrade Jimenez Yareli Licet" y en la ventana *Lectura de una etiqueta NFC*, el dispositivo se acercó a la etiqueta *NFC* con el identificador de la reunión previamente registrada. Se ejecutó el servicio `ReadNFCTag` y se produjo el evento contextual "*The $DEVICE has read a NFC tag*", cuyos parámetros asociados `$TAG_CONTENT` y `$DEVICE` corresponden a los identificadores

de la reunión previamente registrada y del teléfono del usuario, respectivamente.

El servicio contextual evaluó las situaciones asociadas al evento, y se determinó presente la situación "*DEVICE_HAS_NOT_ENOUGH_SPACE*", ya que se satisfacen sus condiciones:

- *$USER is logged in*: la sesión del usuario "Andrade Jimenez Yareli Licet" estaba activa,
- *$MEETING exists*: existía una reunión cuyo identificador corresponde a `$TAG_CONTENT`,
- *$USER is approved to view the $MEETING*: el usuario "Andrade Jimenez Yareli Licet" estaba regristrado como asistente a la reunión en cuestión, i.e. tiene el rol `Assistant`, y
- *$DEVICE has not enough space*: el espacio libre de almacenamiento del dispositivo era insuficiente para la descarga de los documentos asociados a la reunión.

Entonces, el servidor comunicó la presencia de tal situación a sus escuchas interesados, el dispositivo que leyó la etiqueta *NFC*, i.e. *$DEVICE that read the NFC tag*, y el servicio `ShowMeeting`. El servicio `ShowMeeting` convirtió a imágenes JPEG todos los documentos asociados a la reunión mediante el servicio `ConvertDocument`, generó un documento HTML con los datos de la reunión y las imágenes generadas, y lo mostró en la ventana *Consulta de una reunión* del dispositivo (véase la Figura 6.18).

C. Situación *DEVICE_HAS_ENOUGH_SPACE_AND_DOES_NOT_SUPPORT_ALL_FORMATS*

Se eliminaron los archivos generados en la prueba anterior y se aseguró que el dispositivo no contara con ningún visor instalado.

A continuación, se inició sesión en la aplicación con los datos del usuario "Andrade Jimenez Yareli Licet" y en la ventana *Lectura de una etiqueta NFC*, el dispositivo se acercó a la etiqueta *NFC* con el identificador de la reunión previamente registrada. Se ejecutó el servi-

cio `ReadNFCTag` y se produjo el evento contextual "*The `$DEVICE` has read a NFC tag*", cuyos parámetros asociados y `$DEVICE` corresponden a los identificadores de la reunión previamente registrada y del teléfono del usuario, respectivamente.

El servicio contextual evaluó las situaciones asociadas al evento, y se determinó presente la situación "*DEVICE_HAS_ENOUGH_SPACE_AND_DOES_NOT_SUPPORT_ALL _FORMATS*", ya que se satisfacen sus condiciones:

- *`$USER` is logged in*: la sesión del usuario "Andrade Jimenez Yareli Licet" estaba activa,
- *`$MEETING` exists*: existía una reunión cuyo identificador corresponde a `$TAG_CONTENT`,
- *`$USER` is approved to view the `$MEETING`*: el usuario "Andrade Jimenez Yareli Licet" estaba registrado como asistente a la reunión en cuestión. i.e., tiene el rol `Assistant`,
- *`$DEVICE` has enough free space*: el espacio libre de almacenamiento del dispositivo era suficiente para la descarga de los documentos asociados a la reunión, y
- *`$DEVICE` does not support all formats*: el dispositivo no tiene instalado visores para los formatos DOCX, ODT, PDF, y PPTX.

Entonces, el servidor comunicó la presencia de tal situación a sus escuchas interesados, el dispositivo que leyó la etiqueta *NFC*, i.e. *`$DEVICE` that read the NFC tag*, y el servicio `ShowMeeting`. Este último, convirtió los documentos "Presentación de prototipo.pptx", "Reporte mensual.pdf", "Requisitos.odt" y "Resumen.docx" a imágenes JPEG mediante el servicio `ConvertDocument`, generó un documento HTML con los datos de la reunión y los enlaces de descarga a las versiones originales del documento "Cronograma de actividades.png" y a las versiones JPEG de los documentos "Presentación de prototipo.pptx", "Reporte mensual.pdf", "Requisitos.odt" y "Resumen.docx", y lo mostró en la ventana *Consulta de una reunión* del dispositivo (véase la Figura 6.19).

D. **Situación *DEVICE_HAS_ENOUGH_SPACE_AND_SUPPORTS_ALL_FORMATS***

En el dispositivo se instalaron los siguientes visores:

- Adobe Acrobat Reader: capaz de visualizar el formato PDF,
- Docs To Go: capaz de visualizar los formatos DOC, DOCX, PDF, PPT, PPTX, XLS, y XLXS, y
- OpenDocument Reader: capaz de visualizar los formatos ODT, OTT, ODF, ODG, ODP, ODS, y OTS.

A continuación, se inició sesión en la aplicación *NFC Meeting* con los datos del usuario "Andrade Jimenez Yareli Licet" y en la ventana *Lectura de una etiqueta NFC*, el dispositivo se acercó a la etiqueta *NFC* con el identificador de la reunión previamente registrada. Se ejecutó el servicio `ReadNFCTag` y se produjo el evento contextual "*The `$DEVICE` has read a NFC tag*", cuyos parámetros asociados `$TAG_CONTENT` y `$DEVICE` corresponden a los identificadores de la reunión previamente registrada y del teléfono del usuario, respectivamente.

El servicio contextual evaluó las situaciones asociadas al evento, y se determinó presente la situación *DEVICE_HAS_ENOUGH_SPACE_AND_SUPPORTS_ALL_FORMATS*, ya que se satisfacen sus condiciones:

- *`$USER` is logged in*: la sesión del usuario "Andrade Jimenez Yareli Licet" estaba activa,
- *`$MEETING` exists*: existía una reunión cuyo identificador corresponde a `$TAG_CONTENT`,
- *`$USER` is approved to view the `$MEETING`*: el usuario "Andrade Jimenez Yareli Licet" estaba registrado como asistente a la reunión en cuestión, i.e., tiene el rol `Assistant`,
- *`$USER` has enough free space*: el espacio libre de almacenamiento del dispositivo era suficiente para la descarga de los documentos asociados a la reunión, y
- *`$DEVICE` supports all formats*: el dispositivo tiene instalado visores para los formatos

DOCX, ODT, PDF, PNG, y PPTX.

Entonces, el servidor comunicó la presencia de tal situación a sus escuchas interesados, el dispositivo que leyó la etiqueta *NFC*, i.e. "*`$DEVICE` that read the NFC tag*", y el servicio `ShowMeeting`. Este último, generó un documento HTML con los datos de la reunión y los enlaces de descarga a las versiones originales de todos los documentos asociados a la reunión, y lo mostró en la ventana *Consulta de una reunión* del dispositivo (véase la Figura 6.20).

4. **Ejecución de la reunión**

Finalmente, se inició sesión en la aplicación *NFC Meeting* con los usuarios: "Jesús Fernández Cruz", "Sonia Guadalupe Mendoza Chapa", y "Yareli Licet Andrade Jiménez". Al estar conectados más de la mitad de los asistentes registrados en la reunión, se activa la situación permanente "`EXECUTION_OF_THE_MEETING`" (véase la Figura 6.21).

Figura 6.14: Ventana *Registro de una nueva reunión*

Figura 6.15: Mensaje de error para una reunión inexistente

Figura 6.16: Reunión registrada

Figura 6.17: Mensaje de error para un colaborador no autorizado a visualizar una reunión existente

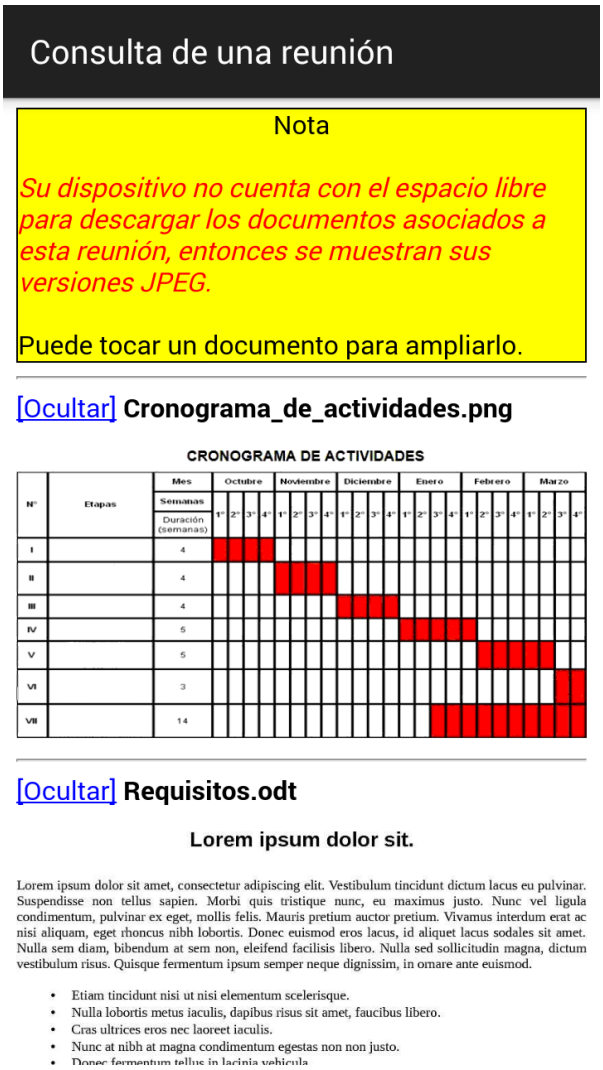


Figura 6.18: Ventana *Consulta de una reunión* con los documentos en sus versiones JPEG

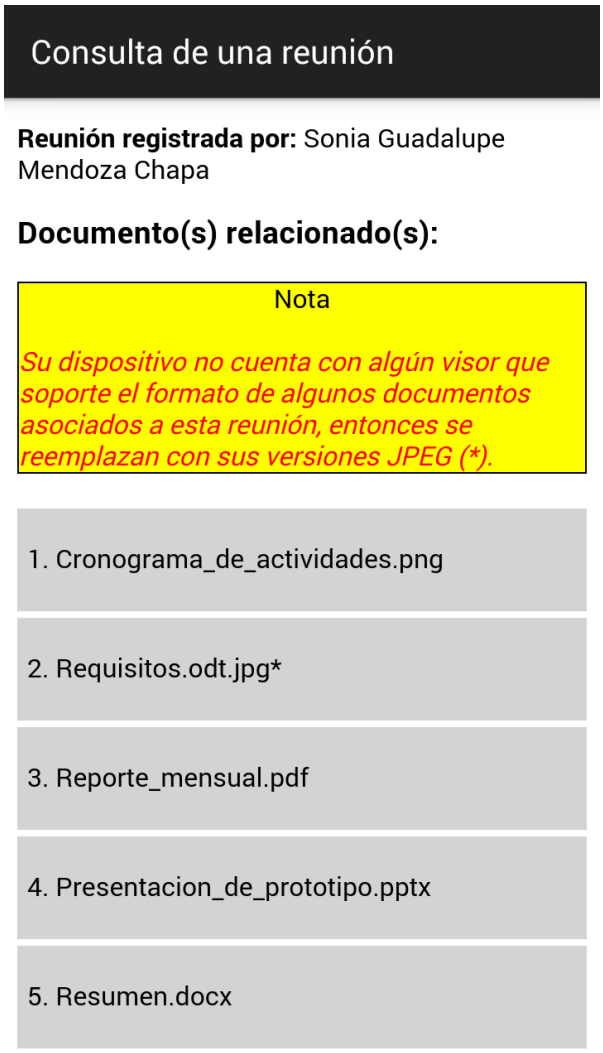


Figura 6.19: Ventana *Consulta de una reunión* con los enlaces de descarga de los documentos en sus versiones JPEG

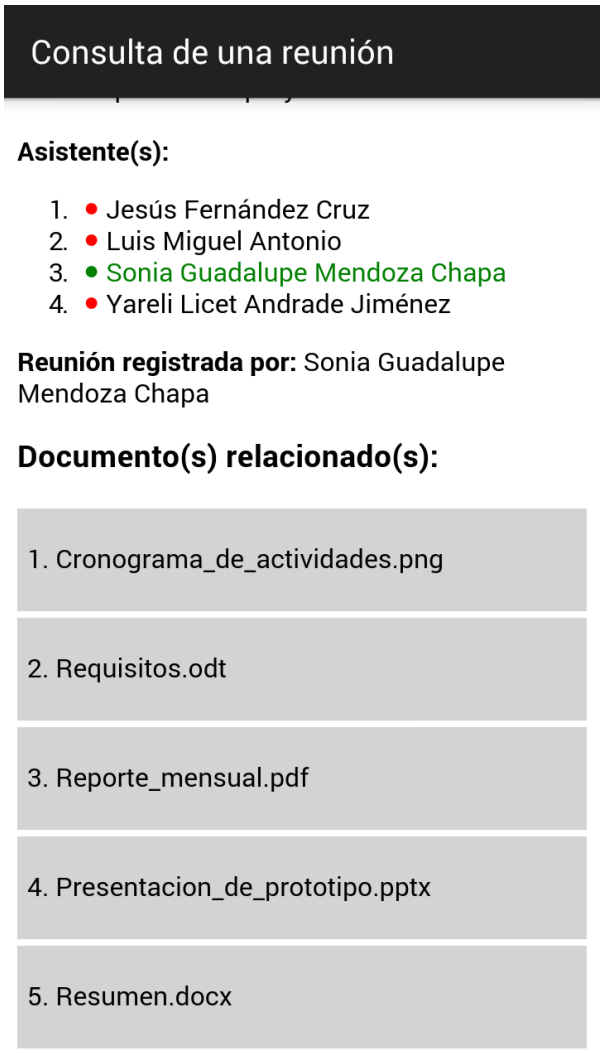


Figura 6.20: Ventana *Consulta de una reunión* con los enlaces de descarga de los documentos en sus versiones originales

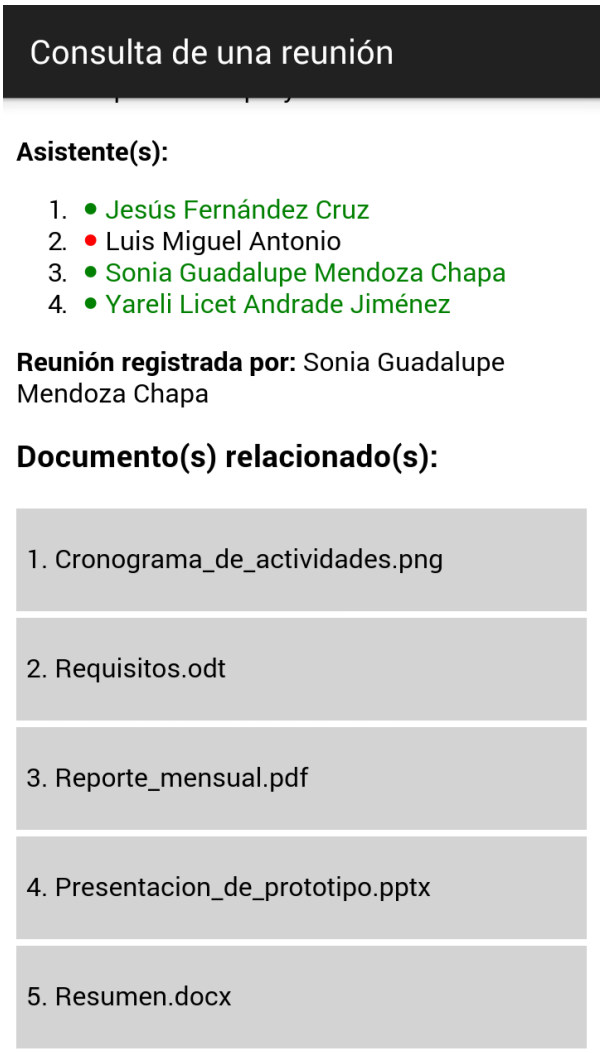


Figura 6.21: Ventana *Consulta de una reunión* con los asistentes observando una reunión

Capítulo 7

CONCLUSIONES Y TRABAJO A FUTURO

Este capítulo tiene como propósito resumir la investigación descrito en el presente documento, señalando sus contribuciones y sus limitaciones, además ofrece una guía del trabajo futuro para continuar esta investigación. Por lo tanto, la sección 7.1 presenta un resumen del planteamiento del problema. Después, en la sección 7.2, se presentan las conclusiones y contribuciones obtenidas de este trabajo. Finalmente, la sección 7.3 exhibe las limitaciones de la solución propuesta y una guía para superarlas.

7.1. Recapitulación del problema

El comportamiento humano es capaz de adaptarse a las distintas situaciones que surgen en un ambiente colaborativo; por el contrario, la mayoría de los sistemas computacionales, no se adapta al contexto del usuario y el resto, lo hace en una forma elemental.

El ajuste automático de los sistemas colaborativos de acuerdo a las situaciones presentes puede realizarse mediante el uso de un soporte adaptativo capaz de hacer frente a los cambios en el "contexto de uso". Actualmente, la investigación actual en la computación consciente

de contexto ha identificado dos problemas principales: 1) la mayoría de los estudios se han enfocado principalmente en el contexto de un único usuario, y 2) el soporte adaptativo de los sistemas consciente de contexto generalmente considera un número reducido de variables contextuales, principalmente la ubicación del usuario y la plataforma de interacción.

Por ello, se desarrolló un marco conceptual para la administración del contexto en sistemas colaborativos para suministrar a los desarrolladores de aplicaciones un soporte para integrar la adaptabilidad al contexto en sus sistemas colaborativos, y representar semánticamente los elementos del espacio colaborativo global.

7.2. Conclusiones y contribuciones

En el presente trabajo se expuso un marco conceptual compuesto por un modelo semántico para representar el espacio colaborativo global y una arquitectura conceptual para suministrar a los desarrolladores de aplicaciones un soporte para integrar la adaptabilidad al contexto en sus sistemas colaborativos.

En el modelo semántico se representan los elementos del espacio colaborativo global, i.e., el espacio de trabajo de un usuario, las aplicaciones bajo el paradigma Modelo-Vista-Controlador que lo integran, y su interacción con los recursos físicos, humanos y virtuales ubicados en una organización, mediante clases y propiedades de objetos en construcciones del lenguaje OWL. Se modelaron, a partir de éste, las aplicaciones propuestas en seis de los escenarios propuestos en el marco XARE: 1) mejoramiento del trabajo colaborativo, 2) remodelación de la interfaz de usuario, 3) redistribución de una aplicación colaborativo, 4) mejoramiento de la conciencia grupal, 5) manipulación de la información, y 6) adaptación de la información.

Las clases desarrolladas, junto a sus propiedades de objetos, se integraron a la ontología `Context` de la arquitectura RAMS. Además, varias de las clases de la arquitectura RAMS fueron incluidas en el modelo semántico, particularmente las clases `User` y `Document`.

Para la arquitectura conceptual se detalla el funcionamiento y la implementación en el lenguaje Java de cada una de sus etapas: 1) percepción de eventos contextuales, 2) detección de situación, 3) comunicación de situación, y 4) adaptación al sistema. Las clases propuestas y la arquitectura conceptual son independientes del dominio de aplicación del sistema colaborativo, y permite integrar, editar y/o eliminar nuevas variables contextuales, eventos contextuales y situaciones en un sistema colaborativo existente.

Finalmente, con el fin de validar el modelo semántico del espacio colaborativo global y la arquitectura conceptual propuesta se modeló e implementó la aplicación *herramienta de administración de contenidos vía NFC*, correspondiente al escenario *adaptación de la información* en el marco XARE. Para su desarrollo, se añadieron las clases `Meeting` y `Viewer`, junto a sus propiedades de objetos, a las ontologías `Context` y `Software` pertenecientes a la arquitectura RAMS.

7.3. Limitaciones y trabajo futuro

En este trabajo se exponen las directrices para la implementación de las etapas de la arquitectura conceptual así como un modelo semántico para representar semánticamente las aplicaciones bajo el paradigma Modelo-Vista-Controlador. Si bien, su diseño y desarrollo se ha concluido, a continuación se recomiendan algunas ideas para continuar esta investigación y mejorar el prototipo exhibido:

- El prototipo *herramienta de administración de contenidos vía NFC* sólo se consideraron eventos contextuales lógicos, por lo que se requiere el desarrollo del elemento *perceptor*, para poder trabajar con eventos contextuales físicos.

- El modelo semántico del espacio colaborativo global representa sus elementos más representativos, i.e., las aplicaciones, los usuarios, los recursos, y sus interacciones, de forma general. Entonces se precisa ampliar las clases y sus propiedades para soportar un mayor rango de las necesidades de las aplicaciones para sistemas colaborativos.

- Se requiere crear un soporte computacional que permita la interacción de las aplicaciones de un ambiente colaborativo mediante el modelo semántico, i.e., la búsqueda y ejecución de servicios o el compartimiento de recursos e información entre las aplicaciones.

- La clase `Viewer` integrada a la ontología `Virtual Resource` contiene una serie de individuos utilizados para poner a prueba el funcionamiento de la *herramienta de administración de contenidos vía NFC* y por ello, solo se añadieron aplicaciones para el sistema operativo Android. Por el gran número de aplicaciones existentes para visualizar o editar un documento, se sugiere añadir más individuos a esta ontología y ampliarla con aplicaciones para otros sistemas operativos, como iOS, Windows Phone y Windows.

- Debido a su integración con el modelo semántico de la arquitectura RAMS, se diseñó un servicio para poder realizar consultas SPARQL a ontologías diseñadas bajo el lenguaje OWL. Una desventaja en el uso de ontologías es el tiempo de respuesta para realizar operaciones en ellas, por ende, se requiere un servicio que permita realizar consultas a una base de datos.

- La arquitectura conceptual expuesta funciona bajo el modelo cliente-servidor; no obstante, para distribuir la carga del servidor es conveniente desarrollar una distribuida, considerando los mecanismos de coherencia en la manipulación de la información.

- La aplicación *NFC Meeting* fue desarrollada exclusivamente para el sistema operativo Android. Si se desea se pueden implementar versiones de esta aplicación para otras plataformas, como iOS, Windows Phone, y Windows.

Bibliografía

[Abowd et al., 1999] Abowd, G. D., Dey, A. K., J., B. P., Davies, N., Smith, M., and Steggles, P. (1999). Towards a better understanding of context and context-awareness. In Gellersen, H., editor, *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*, pages 304–307, Karlsruhe [Germany]. Springer.

[Baldauf et al., 2007] Baldauf, M., Dustdar, S., and Rosenberg, F. (2007). A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):236–277.

[Balme et al., 2005] Balme, L., Demeure, A., Calvary, G., and Coutaz, J. (2005). Sedan-bouillon: A plastic web site. In *PSMD'05: INTERACT 2005 Workshop on Plastic Services for Mobile Devices*, pages 1–3, Roma [Italia]. Citeseer.

[Calvary et al., 2001] Calvary, G., Coutaz, J., and Thevenin, D. (2001). A unifying reference framework for the development of plastic user interfaces. In Little, M. R. and Nigay, L., editors, *Proceeding of the 8th IFIP International Conference on Engineering for Human-Computer Interaction - EHCI*, pages 173–192. Springer, Londres (Inglaterra).

[Decouchant et al., 2013a] Decouchant, D., Mendoza Chapa, S. G., Sánchez Morales, G., and Rodríguez García, J. G. (2013a). Adapting groupware systems to changes in the collaborator's context of use. *Expert Systems with Applications*, 40(11):4446–4462.

[Decouchant et al., 2013b] Decouchant, D., Mendoza Chapa, S. G., Sánchez Morales, G.,

and Rodríguez García, J. G. (2013b). Adapting groupware systems to changes in the collaborator's context of use. *Expert Systems with Applications*, 40(11):4446–4462.

[Dey et al., 2001] Dey, A. K., Abowd, G. D., and Salber, D. (December 2001). A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-computer interaction*, 16(2):97–166.

[Dey et al., 1999] Dey, A. K., Salber, D., Abowd, G. D., and Futakawa, M. (1999). The conference assistant: Combining context-awareness with wearable computing. In *Digest of Papers. The 3rd International Symposium on Wearable Computers, 1999*, pages 21–28, San Francisco C.A. [E.E.U.U.]. IEEE Computer Society.

[Dourish, 2004] Dourish, P. (2004). What we talk about when we talk about context. *Personal and ubiquitous computing*, 8(1):19–30.

[Dourish, 1996] Dourish, P. (June 1996). *Open Implementation and Flexibility in CSCW Toolkits*. PhD thesis, Deparment of Computer Science, University College London.

[Dourish and Bellotti, 1992] Dourish, P. and Bellotti, V. (1992). Awareness and coordination in shared workspaces. In *Proceedings of the 1992 ACM conference on Computer-Supported Cooperative Work CSCW'92*, pages 107–114, Toronto [New York]. ACM.

[Eugster et al., 2003] Eugster, P. T., Felber, P., Guerraoui, R., and Kermarrec, A.-M. (2003). The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131.

[Forkan et al., 2015] Forkan, A., Khalil, I., Ibaida, A., and Tari, Z. (2015). Bdcam: Big data for context-aware monitoring — a personalized knowledge discovery framework for assisted healthcare. *IEEE Transactions on Cloud Computing*, PP(99):1–1.

[Forkan et al., 2014] Forkan, A., Khalil, I., and Tari, Z. (2014). Cocamaal: A cloud-oriented context-aware middleware in ambient assisted living. *Future Generation Computer Systems: the international journal of grid computing: theory, methods and applications*, 35:114–127.

[Gamma et al., 2000] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (2000). *Design Patterns: Elements of Reusable Object-Oriented Software*, volume 1. Addison-Wesley, Upper Saddle River NJ (USA).

[García García, 2013] García García, E. K. (Diciembre 2013). *Ascertaining the Availability of Shared Resources in Ubiquitous Collaborative Enviroments*. PhD thesis, Departamento de Computación, CINVESTAV.

[García García et al., 2016] García García, E. K., Mendoza Chapa, S. G., Decouchant, D., and Brézillon, P. (Octubre 2016). Facilitating resource sharing and selection in ubiquitous multi-user environments. *Information Systems Frontiers*, pages 1–21.

[García García et al., 2013] García García, E. K., Mendoza Chapa, S. G., Decouchant, D., Rodríguez García, J. G., and Pérez de los Santos Mondragon, T. (Junio 2013). Determining and locating the closest available resources to mobile collaborators. *Expert Systems with Applications*, 40(7):2511–2529.

[Ghiani et al., 2009] Ghiani, G., Paternó, F., Santoro, C., and Davide Spano, L. (2009). Ubicicero: A location-aware, multi-device museum guide. *Interacting with Computers*, 21(4):288–303.

[Gruber, 2009] Gruber, T. (2009). Ontology.

[Guarino, 1995] Guarino, N. (1995). Formal ontology, conceptual analysis and knowledge representation. *International Journal of Human-Computer Studies*, 43(5-6):625–640.

[Haake et al., 2010] Haake, J. M., Hussein, T., Joop, B., Lukosch, S., Veiel, D., and Ziegler, J. (2010). Modeling and exploiting context for adaptive collaboration. *International Journal of Cooperative Information Systems*, 19(01n02):71–120.

[Horrocks, 2011] Horrocks, I. (2011). Tools support for ontology engineering. pages 103–112.

[Jaouadi et al., 2016] Jaouadi, I., Djemaa, R. B., and Ben-Abdallah, H. (2016). A model-driven development approach for context-aware systems. *H. Softw Syst Model*.

[Kirsch-Pinheiro et al., 2004] Kirsch-Pinheiro, M., Gensel, J., and Martin, H. (2004). Representing context for an adaptative awareness mechanism. In *International Workshop on Groupware, CRIWG 04*, volume 3198, pages 339–348, San Carlos [Costa Rico]. Springer.

[McKeever et al., 2008] McKeever, S., Ye, J., Coyle, L., and Dobson, S. (2008). A context quality model to support transparent reasoning with uncertain context. In Rothermel, K., Fritsch, D., Blochinger, W., and Dürr, F., editors, *Quality of Context*, pages 65–75, Stuttgart [Germany]. Springer.

[Miao and Haake, 1999] Miao, Y. and Haake, J. M. (March 1999). Supporting concurrent design in scope. *The International Journal of Concurrent Engineering Research and Applications*, 7(1):55–65.

[Milić and Stojanović, 2017] Milić, E. and Stojanović, D. (2017). Egosense: A framework for context-aware mobile applications development. *Engineering, Technology & Applied Science Research*, 7(4):1791–1796.

[Morbach et al., 2009] Morbach, J., Wiesner, A., and Marquardt, W. (2009). Ontocape 2.0 — a (re)usable ontology for computer-aided process engineering. *Computers & Chemical Engineering Journal*, 33(10):1546–1556.

[Noy and Hafner, 1997] Noy, N. F. and Hafner, C. D. (1997). The state of the art in ontology design: A survey and comparative review. *AI magazine*, 18(3):53–74.

[Olivares Toledo, 2011] Olivares Toledo, A. (Diciembre 2011). Arquitectura para el soporte de contexto de uso en sistemas colaborativos. Master's thesis, Departamento de Computación, CINVESTAV.

[Prante et al., 2004] Prante, T., Streitz, N. A., and Tandler, P. (2004). Roomware: Computers disappear and interaction evolves. *Computer*, 37(12):47–54.

[Romero Gama et al., 2013] Romero Gama, M., Mendoza Chapa, S. G., and Sánchez Morales, G. (September 30 - October 4, 2013). Xare: A framework for developing context-aware

applications for co-located collaborative work. In *10th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE), 2013*, pages 336–341, Mexico City [Mexico]. IEEE.

[Sánchez Morales, 2013] Sánchez Morales, G. (Diciembre 2013). *XARE: marco de desarrollo para sistemas colaborativos conscientes de contexto*. PhD thesis, Departamento de Computación, CINVESTAV.

[Tandler et al., 2001] Tandler, P., Prante, T., Müller-Tomfelde, C., Streitz, N., and Steinmetz, R. (2001). Connectables: Dynamic coupling of displays for the flexible creation of shared workspaces. In *Proceedings of the 14th annual ACM Symposium on User Interface Software and Technology*, pages 11–20, Orlando F.L. [E.E.U.U.]. ACM Press.

[Vanderdonckt et al., 2008] Vanderdonckt, J., Calary, G., Coutaz, J., and Stanciulescu, A. (2008). *Multimodality for Plastic User Interfaces: Models, Methods and Principles*, chapter 4, pages 61–84. Berlin-Heidelberg, Germany: Springer.

[Vanderdonckt and González Calleros, 2008] Vanderdonckt, J. and González Calleros, J. M. (2008). Task-driven plasticity: One step forward with ubidraw. In *Proceedings of the 2nd Conference on Human-Centered Software Engineering (HCSE) and 7th International Workshop on Task Models and Diagrams (TAMODIA)*, pages 181–196. Springer, Pisa [Italia].

[Want et al., 1992] Want, R., Hopper, A., Falcao, V., and Gibbons, J. (1992). The active badge location system. *ACM Transactions on Information Systems (TOIS)*, 10(1):91–102.