TESIS DEFENDIDA POR

Gustavo Alexander Berzunza Richard

Y APROBADA POR EL SIGUIENTE COMITÉ

Dr. Jesús Favela Vara

Director del Comité

Dr. José Antonio García Macías

Miembro del Comité

Dr. Luis Armando Villaseñor González

Miembro del Comité

Dr. Pedro Gilberto López Mariscal

Coordinador del programa de posgrado en Ciencias de la Computación

Dr. David Hilario Covarrubias Rosales

Director de Estudios de Posgrado

20 de Noviembre de 2008

CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE EDUCACIÓN SUPERIOR DE ENSENADA

PROGRAMA DE POSGRADO EN CIENCIAS
EN CIENCIAS DE LA COMPUTACION

ADAPTACION DE INFORMACION EN SISTEMAS CONSCIENTES DEL CONTEXTO

TESIS

que para cubrir parcialmente los requisitos necesarios para obtener el grado de

MAESTRO EN CIENCIAS

Presenta:

GUSTAVO ALEXANDER BERZUNZA RICHARD

Ensenada, Baja California, México. Noviembre del 2008.

RESUMEN de la tesis de **GUSTAVO ALEXANDER BERZUNZA RICHARD**, presentada como requisito parcial para la obtención del grado de **MAESTRO EN CIENCIAS** en **CIENCIAS DE LA COMPUTACION**. Ensenada, Baja California. Noviembre del 2008.

ADAPTACIÓN DE INFORMACIÓN EN SISTEMAS CONSCIENTES DEL CONTEXTO

Resumen aprobado por:

1-N

Dr. Jesús Favela Vara

Director de Tesis

Los sistemas conscientes del contexto adaptan su comportamiento en base a información del ambiente para ofrecer al usuario servicios y/o información relevantes a la situación actual. En los sistemas de cómputo una posible adaptación es la de la interfaz de usuario, lo que es más relevante dada la reciente proliferación de dispositivos con capacidades y recursos diferentes. La adaptación de la interfaz de usuario generalmente está en función de las capacidades del dispositivo, sin embargo otro tipo de información contextual como la ubicación, compañía o actividad del usuario pueden tomarse en cuenta para realizar la adaptación de la interfaz de usuario.

En el presente trabajo, es de interés idear mecanismos para incorporar información contextual dentro de las posibles adaptaciones que puede sufrir la interfaz de usuario. Entre éstas, la adaptación para ser desplegada en dispositivos de cómputo heterogéneos y la actualización de la información presentada para que sea adecuada al contexto actual. Para tal tarea se parte del uso de un «lenguaje declarativo de la interfaz de usuario» existente (*XForms*), el cual captura de manera abstracta y genérica la interfaz de usuario, al indicar *qué* debe hacer la interfaz y no *cómo*. Además no hace suposición alguna sobre el dispositivo al cual será traducida. A partir de éste lenguaje y considerando sus características, se define una estrategia de adaptación que consta de tres etapas, la primera se relaciona con la generación de la interfaz abstracta que sea adecuada al contexto actual, la segunda se relaciona a la traducción de dicha interfaz abstracta a una interfaz concreta y la última etapa contempla las adaptaciones que puede sufrir la interfaz concreta en estructura e información, en base a cambios en el contexto.

La validación de la estrategia de adaptación se efectúa a partir del desarrollo de escenarios de aplicación que requieren cierto tipo de adaptación en su interfaz de usuario. El cumplimiento de los requerimientos de adaptación impuestos por los escenarios, así como la extensibilidad de los elementos propuestos, son elementos que se utilizan para validar la propuesta. Se concluye que *XForms* ofrece mecanismos de manera nativa que pueden ser utilizados en sistemas que dan soporte a la consciencia del contexto, sin embargo, es necesario hacer consideraciones relacionadas a la estructura y forma de trabajo de dichos mecanismos para ser utilizados en este tipo de sistemas.

Palabras clave: Sistemas conscientes del contexto, Interfaz de usuario, *XForms*.

ABSTRACT of the thesis presented by **GUSTAVO ALEXANDER BERZUNZA RICHARD** as a partial requirement to obtain the **MASTER OF SCIENCE** degree in **COMPUTER SCIENCE**. Ensenada, Baja California. November 2008.

INFORMATION TAILORING IN CONTEXT-AWARE SYSTEMS

Context aware systems adapt their behavior based on information from the environment to provide services and/or information to the user that are relevant to the current situation. In computer systems, the adaptation of the user interface is a possible kind of adaptation, which becomes more relevant due to the recent proliferation of devices with different capabilities and resources. The user interface adaptation is usually based on device's capabilities, though other contextual information such as location, company or user activity can be taken into account to tailor the user interface.

In this work, we aim to devise mechanisms to incorporate contextual information within any possible adaptation that the user interface could realize. Among these, the adaptation to be deployed in heterogeneous computing devices and the update of the information presented to be appropriate for the current context. For this task, we start with the use of an existing «user interface declarative language» (*XForms*), which captures the user interface in an abstract and generic way, specifying what it must do and not how it should do it. In addition it does not assume any specific characteristic of the device to which it will be translated. From this language, and considering their characteristics, we propose an adaptation strategy that consists of three stages, the first relates to the generation of an abstract interface that is appropriate to the current context, the second is related to the translation of the abstract interface to a concrete interface and the last phase includes the adjustments that may suffer the concrete interface in its structure and information, based on changes in context.

The validation of the adaptation strategy is carried out from the development of use cases that need certain type of adaptation in the user interface. The compliance with the adaptation requirements imposed by the use cases, as well as the extensibility of the proposed elements, are used to validate the proposal. We conclude that *XForms* natively provides mechanisms that can be used in systems that support context aware systems, however, it is necessary to make considerations about the structure and way of working of such mechanism to be used in this kind of systems.

Keywords: Context aware systems, User interface, *XForms*.

Dedicatorias

A mi gran familia, a todos los que forman parte de ella.

Agradecimientos

Al Dr. Jesús Favela Vara
por su invaluable dirección y apoyo.

A los miembros del comité de tesis,
Dr. J. Antonio García Macías y Dr. Luis Villaseñor González.

A Gustavo, Ma. del Carmen, Naamen y Christopher,
por el amor y compañía que me brindaron en esta experiencia que me ha tocado vivir.

A todos mis compañeros de generación y a toda la gente que tuve el privilegio de conocer durante estos dos años.

Al Centro de Investigación Científica y de Educación Superior de Ensenada (CICESE).

Al Consejo Nacional de Ciencia y Tecnología (CONACYT).

CONTENIDO

Página

CONTENIDO (continuación)

CONTENIDO (continuación)

CONTENIDO (continuación)

LISTA DE FIGURAS

LISTA DE FIGURAS (continuación)

LISTA DE TABLAS

Capítulo I

Introducción

I.1 Antecedentes

El «cómputo ubicuo», también conocido como *ubicomp*, ha sido denominado como la tercera era del cómputo (Weiser *et al*., 96), en ésta, el uso de las computadoras se extiende al hacerlas disponibles a través de todo el entorno físico, de manera invisible, es decir, que escapen de la consciencia del usuario y la interacción con éste se lleve de manera natural. En esta corriente computacional cada usuario interactúa con cientos de computadoras interconectadas en los alrededores. Aquí es necesario contar con nuevos tipos de computadoras de distintos tamaños y formas, pero que conserven su capacidad de procesamiento (Weiser, 93). La información de entrada utilizada en este tipo de entornos se debe inferir de la interacción natural de los usuarios con su ambiente, por ejemplo, la ubicación, la hora, las condiciones ambientales, los dispositivos de cómputo de los cuales hace uso de manera explícita, el estado de la red, etc. La infraestructura tecnológica se encuentra empotrada en todos los lugares posibles como las paredes, sillas, ropa, controladores de luz, etc. esto da lugar a los ambientes aumentados.

I.1.1 Sistemas Conscientes del Contexto

Más allá de la infraestructura mencionada, que entre otras tareas permite sensar el entorno, se encuentran aquellas aplicaciones que infieren, procesan y diseminan la información del ambiente para que otros sistemas hagan uso de ella. Los sistemas que hacen uso de dicha

información para adaptar y ofrecer servicios e información adecuados a las condiciones del entorno, son denominados «sistemas conscientes del contexto».

La tarea de los «sistemas conscientes del contexto» involucra la solución de problemas de adaptación multidimensional, ya que la adaptación está en función de distintos «contextos», por ejemplo, el «contexto del usuario» (ubicación, hora, capacidad del dispositivo móvil y preferencias), o el «contexto de red» (el ancho de banda). Aunado a esto se tiene el concepto de «servicio universal», en el cual la idea es permitir a un usuario acceder de manera transparente a un servicio en cualquier momento, lugar y desde cualquier dispositivo (Liota *et al*., 02).

La adaptación en función del «contexto», involucra también la información que se presenta al usuario. En los sistemas de cómputo, debido a la diversidad de dispositivos de cómputo con capacidades y recursos diferentes, es necesario adaptar la «interfaz de usuario», que es el medio por el cual un usuario se comunica con un dispositivo, por lo tal, dicha adaptación está en función de las capacidades y recursos del dispositivo. Sin embargo, el definir interfaces específicas para cada dispositivo no es una tarea escalable, una solución es la definición de mecanismos para llevar a cabo la adaptación pertinente de manera semi-automática.

I.1.2 Lenguajes Declarativos de la Interfaz de Usuario

Una propuesta que da soporte a la adaptación semi-automática de la interfaz, son los «lenguajes declarativos de la interfaz de usuario» (*UIDL*, por sus siglas en inglés) cuyo objetivo es definir de manera abstracta y genérica la «interfaz de usuario», dicha definición escapa de toda suposición relacionada al dispositivo objetivo, para el cual es traducida. La traducción da como resultado la «interfaz concreta», la cual es la manifestación física de la «interfaz abstracta» a través de los recursos y capacidades del dispositivo, incluso dependiendo del enfoque utilizado permiten generar el código fuente de la «interfaz de usuario» en un lenguaje de programación particular.

I.2 Planteamiento del Problema

El uso de un «lenguaje declarativo de la interfaz de usuario» permite transmitir la información de la «interfaz de usuario» de una manera abstracta y genérica, sin embargo, cuando se realiza una adaptación a partir de la «interfaz abstracta», normalmente se relaciona con su traducción a una «interfaz concreta», para lo cual el uso de información contextual se limita a la información de las capacidades y recursos del dispositivo, sin tomar en cuenta otro tipo de información del ambiente como la tarea, ubicación o tarea del usuario. Este tipo de lenguajes no incluyen dentro de su especificación la incorporación de información del «contexto», ya que dicha información debe ser contemplada durante el proceso de generación de la «interfaz concreta».

I.3 Propuesta

En este trabajo se propone una estrategia de adaptación para la «interfaz de usuario» que involucra el uso de información contextual como la ubicación, tarea o compañía del usuario, además de la información utilizada de acuerdo al dispositivo de despliegue, es decir, sus capacidades y recursos. El trabajo contempla la definición de los mecanismos para llevar a cabo la adaptación de la «interfaz de usuario» en conjunto con un «lenguaje declarativo de la interfaz de usuario» existente.

La estrategia de adaptación de la «interfaz de usuario», se define en tres etapas que involucran la generación de la «interfaz abstracta» descrita utilizando el «lenguaje declarativo de la interfaz de usuario», la interpretación de la «interfaz abstracta» para un dispositivo específico y por último las adaptaciones que sufre la «interfaz concreta funcional» en base a los cambios en el «contexto». Aquí la información contextual se utiliza en cada una de las etapas de adaptación.

I.3.1 Objetivos

I.3.2 Objetivo General

El objetivo general del presente trabajo consiste en establecer mecanismos para la adaptación de la interfaz de usuario de acuerdo a la información contextual como la tarea o ubicación del usuario y al dispositivo de despliegue, partiendo del uso de un «lenguaje declarativo de la interfaz de usuario» existente.

I.3.3 Objetivos Específicos

Del objetivo general se desprenden los siguientes objetivos específicos:

- Analizar los «lenguajes declarativos de la interfaz de usuario» existentes en términos de su uso o posible extensión para incorporar información contextual en las posibles adaptaciones que la «interfaz de usuario» puede comprender.
- Bosquejar las posibles adaptaciones de la «interfaz de usuario» y establecer los mecanismos para llevar éstas a cabo.
- Diseñar y desarrollar escenarios de aplicación para validar la propuesta de adaptación.

I.4 Metodología

La Figura 1 resume la metodología utilizada en este trabajo de tesis, cada etapa se describe de manera general.

Figura 1. ***Metodología propuesta para el desarrollo del trabajo de tesis.***

Revisión de la literatura, esta etapa consiste en la revisión de trabajos previos realizados en el área de investigación correspondiente, tomando en cuenta las restricciones y limitaciones presentes en dichos trabajos para encaminar la investigación.

Revisión de lenguajes declarativos de la interfaz de usuario, esta etapa involucra la revisión de «lenguajes declarativos de la interfaz de usuario» existentes en la literatura, para realizar un compendio de sus características, ventajas y desventajas. Otro elemento que se toma en cuenta en esta revisión es su posible uso en conjunto con información contextual para la adaptación de la «interfaz de usuario». De esta etapa se desprende la selección del lenguaje que sirve de base para la estrategia de adaptación propuesta.

Bosquejo de escenarios de aplicación, esta etapa consiste en visualizar posibles escenarios de aplicación en los cuales se requiera la adaptación de la «interfaz de usuario». Esto con el objetivo de encaminar el diseño de la propuesta utilizando ejemplos concretos, ya que estos escenarios dan la pauta para conocer a qué tipo de adaptación es necesario dar soporte.

Diseño de mecanismos de adaptación, esta etapa contempla el diseño de los mecanismos de adaptación que se involucran directamente con la adaptación de la «interfaz de usuario», las decisiones de diseño se basan en la revisión literaria, el «lenguaje declarativo de la interfaz de usuario» seleccionado y los ejemplos de escenarios de aplicación.

Implementación de mecanismos de adaptación, en esta etapa se implementan los mecanismos diseñados en la etapa anterior.

Diseño y desarrollo de escenarios de aplicación, en esta etapa se concretizan escenarios de aplicación que requieran dar soporte a la adaptación de la «interfaz de usuario», los cuales se implementan utilizando los mecanismos de adaptación propuestos para validar su uso.

Validación de estrategia de adaptación, a la par de la etapa descrita anteriormente se valida la estrategia propuesta para la adaptación de la «interfaz de usuario», esto se hace en base al cumplimiento de los requisitos dados por los escenarios.

I.5 Organización de la Tesis

El presente documento de tesis se divide en seis capítulos y un conjunto de apéndices. A continuación se describe de manera breve el contenido de los capítulos.

El **Capítulo II**, presenta dos temas de interés para este trabajo, el primero corresponde a las «interfaces de usuario de *software*», donde se describen conceptos básicos y se aborda el tema del desarrollo de éstas. El segundo tema corresponde a los «sistemas conscientes del contexto», en el cual se presentan conceptos básicos, su relación con las «interfaces de usuario» y se dan algunos lineamientos sobre su uso en este trabajo. Este capítulo así como el III, recaban información que encamina las decisiones de diseño para el presente trabajo.

El **Capítulo III**, recaba información sobre los «lenguajes declarativos de la interfaz de usuario», presenta una recopilación de información sobre estos, además, muestra de modo comparativo algunas características que presentan y da a conocer las decisiones para la selección del lenguaje *XForms* para ser usado en este trabajo. Por otro lado, introduce al lector de manera resumida al lenguaje *XForms*, al presentar detalles específicos sobre éste.

El **Capítulo IV** presenta el diseño e implementación de la estrategia de adaptación de la «interfaz de usuario» propuesta en este trabajo. Se describe cada elemento que se involucra dentro de la adaptación y sus fases. Además, se describe una visión general de cómo utilizar la propuesta dentro de un sistema que dé soporte a la detección y uso de información del «contexto», en otras palabras, un sistema que de soporte a la «consciencia del contexto».

El **Capítulo V** presenta dos escenarios de aplicación que reflejan la necesidad de la adaptación de la «interfaz de usuario», estos escenarios se implementan utilizando la propuesta planteada, para así validar las suposiciones hechas de la estrategia de adaptación del lenguaje *XForms*.

El **Capítulo VI**, presenta las conclusiones generales, la aportación hecha en este trabajo, las limitantes y el trabajo futuro.

El apartado de **Apéndices** incluye los documentos *XML* obtenidos del diseño e implementación de los elementos de adaptación propuestos, así como los que se utilizan en los escenarios de aplicación.

Cada capítulo presenta una sección en donde se resume y establecen las conclusiones del capítulo.

Capítulo II

Adaptación de la Interfaz de Usuario en Aplicaciones Conscientes del Contexto

II.1 Introducción

Este capítulo presenta una visión general de dos temas de interés para este trabajo, las «interfaces de usuario de *software*» y los «sistemas conscientes del contexto», la primera parte del capítulo corresponde al primer tema, la segunda aborda los «sistemas conscientes del contexto» así como su relación con las «interfaces de usuario de *software*», de este modo se provee un marco de referencia teórico para los capítulos posteriores así como una recopilación de evidencia para la toma de decisiones de diseño para la propuesta de trabajo.

II.2 Interfaces de Usuario de Software

La finalidad de una «aplicación de *software*» es permitir a un usuario realizar una tarea, que puede ir desde la recuperación y despliegue de información, la captura y el procesamiento de datos, entre otras; el cómo llevar a cabo estas tareas se encuentra plasmado en lo que se conoce como la «lógica de la aplicación». El medio a través del cual el usuario puede interactuar con dicha aplicación es la «interfaz de usuario». Por lo general la «interfaz de usuario» se desarrolla de manera independiente de la lógica y los datos de la aplicación, lo cual es una práctica recomendada dentro del desarrollo de *software*. El objetivo de la «interfaz de usuario» es actuar como un intermediario entre el usuario y los datos al

presentarlos en un formato adecuado para su interacción. Esto último está en función de los recursos del dispositivo desde el que se accede la aplicación, es decir, sus capacidades de entrada (p. ej. teclado, voz, lector digital, etc.) y salida (p.ej. audio, pantalla digital, etc.), así como sus características de *hardware* (p. ej. tamaño/resolución de pantalla, memoria, etc.), permitiendo así la comunicación con el dispositivo.

Cuando se habla de dispositivos de cómputo como computadoras de escritorio (*PC*'s), *PDA*'s, celulares, entre otros, normalmente se presenta una «interfaz de usuario gráfica». Este tipo de interfaces utiliza un conjunto de componentes gráficos para representar la información y permitir la interacción al usuario, a estos componentes se les conoce como *widgets* o *controles*, estos son los botones, ventanas, cuadros de texto, menús desplegables y demás elementos que conforman dicha interfaz. Aunque el término *widget* se relaciona comúnmente con una «interfaz de usuario gráfica», también se puede emplear de manera más general para denotar cualquier bloque de construcción primitivo dentro de una «aplicación de *software*», por ejemplo, módulos con una funcionalidad específica. Seffah en (Seffah *et al*., 04a), distingue tres tipos de interfaces, según su plataforma de cómputo:

- Interfaces de usuario gráficas (*GUI*, por sus siglas en inglés), éstas se utilizan mayormente en computadoras de escritorio. Emplean cuatro elementos fundamentales: ventanas, iconos, menús y punteros.
- Interfaces de usuario basadas en web (*WUI*, por sus siglas en inglés), por lo general este tipo de interfaces se componen de una mezcla de etiquetas (p. ej. *HTML*, sintaxis *XML*), hojas de estilo, *scripts*, etc. Normalmente se presentan a través de aplicaciones conocidas como «navegadores», las cuales proveen la interacción y navegación básica para éstas.
- Interfaces de usuario para dispositivos de bolsillo (*HUI*, por sus siglas en inglés), aquí se contemplan los dispositivos como celulares y *PDA*'s. Dichos dispositivos se pueden catalogar en dos clases, aquellos con una *GUI* en apariencia y comportamiento y aquellas que utilizan un subconjunto de *GUI* o *WUI*. Ambos tipos hacen uso de un dispositivo tipo pluma (*stylus*) y/o una pantalla sensible al tacto (*touch screen*) para llevar a cabo la interacción.

La Figura 2 muestra ejemplos de «interfaces de usuario de *software*» de una a) *PC*, b) *PDA* y c) celular. Cada una está desarrollada en base a los recursos del dispositivo, por ejemplo, el área de despliegue y medio de interacción (*mouse*, teclado, pantalla sensible al tacto, etc.).

Figura 2. ***Ejemplo de Interfaces de usuario para una a) PC, b) PDA y c) celular.***

Los apartados anteriores describen «interfaces de usuario de *software*», las cuales permiten interactuar con una aplicación de cómputo, pero dado el cometido de una «interfaz de usuario», la de ser mediadora entre cualquier interacción hombre-máquina, no sólo se tienen «interfaces de usuario» implementadas como aplicaciones de *software*. La Figura 3 muestra una «interfaz de usuario» desarrollada como un dispositivo de *hardware* a modo de pulsera, llamado *Maior Vocce*[1], la cual permite a través de comandos de voz controlar la red domótica de la que forma parte. De esta manera permite a los usuarios interactuar con los servicios ofrecidos por dicha red, por ejemplo controlar luces, calefacción, electrodomésticos, sistemas de riego, etc.

[1] http://www.fagor.com/domotica/_bin/cast/maiorvoicce.php

Figura 3. ***Ejemplo de una «interfaz de usuario» implementada como un dispositivo de hardware.***

Del apartado anterior se puede hacer una distinción entre «interfaces de usuario de *software*» e «interfaces de usuario de *hardware*». En las siguientes secciones se hace énfasis en las primeras, pero es conveniente tener en cuenta las segundas a modo de referencia. Por último es conveniente mencionar que dadas las capacidades de cada dispositivo, hay tareas que son más adecuadas para realizar con unos que con otros, inclusive hay tareas que no son factibles de realizar con algún dispositivo, de aquí que es necesario ser conscientes de las posibilidades y objetivo de cada dispositivo.

II.2.1 Desarrollo de Interfaces de Usuario para Dispositivos Heterogéneos

La diversidad de dispositivos de cómputo existentes (p. ej. *PC*'s, celulares, *PDA*'s, laptops, etc.) que se fabrican sin seguir un estándar de recursos, conectividad, interacción, etc., da lugar a que las personas cuenten con dispositivos heterogéneos. Una consideración propia de los dispositivos de bolsillo como *PDA*'s y celulares, son las capacidades inferiores de procesamiento y recursos en comparación con otros equipos, como una *PC* de escritorio, ejemplo de esto es la fuente de energía, pantalla de despliegue, entre otras.

Dado que diferentes dispositivos tienen distintas capacidades de interacción de usuario y despliegue de información, la mayoría de las aplicaciones o servicios no pueden adaptar sus interfaces de usuario a estas diferencias. Esto significa que el usuario tiene que utilizar diferentes versiones de un servicio de distintos proveedores para acceder a la misma funcionalidad (Nylander *et al.*, 05). Asociado a lo anterior se tiene el desarrollo de aplicaciones o servicios que se enfocan a un dispositivo específico, lo que limita el acceso de las personas a dichas aplicaciones y servicios, ya que se diseñan bajo las restricciones de

las características de dispositivos particulares. Lo anterior obliga a las personas a hacer uso de dispositivos específicos para acceder a una aplicación particular. Por ejemplo, una aplicación para controlar un proyector electrónico en una sala de juntas.

La posibilidad de tener acceso a tales servicios o aplicaciones independientemente del dispositivo con el que se cuente requiere, de la adaptación de la «interfaz de usuario» a las características del dispositivo, para lo que se debe tomar en cuenta la forma de interacción (pluma, voz, teclado), la forma de intercambio de datos, entre otros. Sin embargo, definir una interfaz específica para cada tipo de dispositivo existente a partir de una aplicación en particular no es una tarea escalable para los desarrolladores de aplicaciones. En base a esto es necesario definir mecanismos semi-automáticos para la adaptación de la «interfaz de usuario» sin que la funcionalidad provista por la aplicación se vea afectada. Independientemente que las interfaces de usuario se diseñen e implementen o sean generadas en tiempo de ejecución, se requiere cierto conocimiento *a priori* sobre las capacidades del dispositivo de usuario así como de las preferencias personales (Repo *et al*., 04).

Dos metodologías para el desarrollo de interfaces de usuario son: el enfoque basado en modelos y el enfoque basado en patrones.

II.2.1.1 Desarrollo de Interfaces Basado en Modelos

El objetivo del enfoque basado en modelos es identificar abstracciones útiles que resalten los aspectos básicos de una «interfaz de usuario» que deberían ser tomados en cuenta al momento de diseñar aplicaciones interactivas. Un enfoque basado en modelos, describe la «interfaz de usuario» desde diferentes perspectivas (Witt, 05). Algunos modelos existentes en la literatura son (Puerta, 97; Puerta *et al*., 01):

- Modelo del Usuario, el cual captura una jerarquía de usuarios, definiendo por ejemplo niveles de privilegios.
- Modelo de Diálogo, describe la interacción hombre-máquina, al especificar en qué momento el sistema debe capturar una entrada del usuario, realizar consultas a éste o presentar información.

- Modelo de Presentación, especifica cómo los elementos de interacción (o *widgets*) aparecen en distintos estados del diálogo. Generalmente consiste en una descomposición jerárquica de las posibles pantallas a presentar. Normalmente el modelo de diálogo y de presentación están estrechamente relacionados. Ambos se pueden considerar como elementos concretos dentro de la interacción del usuario y la aplicación.
- Modelo de tareas, se enfoca a las actividades/tareas que se han de realizar. El modelo involucra entidades como metas, acciones y objetos de dominio. Las metas especifican cuándo se ha alcanzado un estado o meta, las acciones definen los procedimientos para alcanzar dichas metas, y los objetos de dominio representan los elementos que han de presentarse en la interfaz para completar cada tarea en el modelo. Ejemplos de tareas pueden ser: "ingresar fecha", "ver mapa" o "redactar documento".
- Modelo de dominio, conocido también como «modelo de aplicación». Generalmente la interfaz de usuario se utiliza para presentar o modificar datos. El modelo se relaciona en gran medida con el modelo de tareas ya que las tareas involucran la presentación o manipulación de datos (Van den Bergh *et al*., 04). De aquí que este modelo represente los objetos y clases de objetos con los que el usuario interactúa en las tareas. Además describe las relaciones entre los objetos dentro de un dominio. Ejemplos de objetos de dominio pueden ser: una "fecha", un "mapa" o un "documento", los cuales pueden verse como elementos abstractos.

La estrategia con el enfoque de modelos es describir distintos aspectos de la aplicación que permitan generar una «interfaz de usuario» que reflejen los requisitos de la aplicación. Dichos aspectos pueden ser la presentación, el diálogo, las tareas, etc. Lo que aquí se tiene es la generación de una «interfaz de usuario concreta» (la interfaz con la que interactúa el usuario) a partir de la representación abstracta y genérica de la «interfaz de usuario». Esto a través del mapeo de los modelos abstractos en interfaces concretas o elementos de éstas (Bomsdorf *et al*., 98). Dependiendo de los modelos que se utilicen, el alcance puede ir más allá de sólo la interfaz. La Figura 4 muestra el flujo seguido en el proceso de generación de la interfaz concreta.

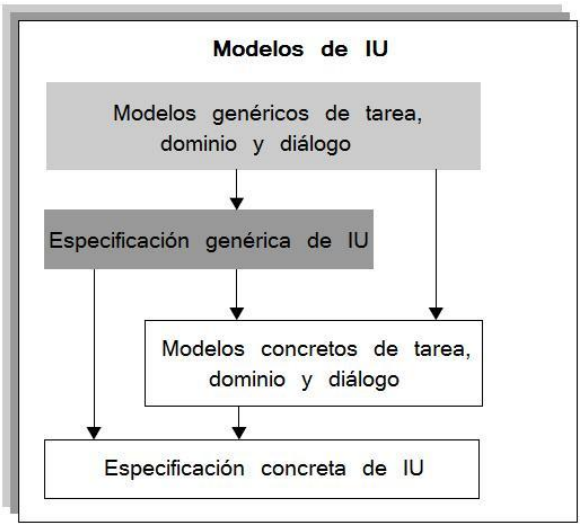

Figura 4. ***Modelos y mapeos en el desarrollo basado en modelos.***

El enfoque basado en modelos para el desarrollo de interfaces de usuario requiere de (Luyten, 04)**:**

- Interfaces centradas a la tarea.
- Soporte multiplataforma.
- Adaptación de la interfaz.
- Interfaces multimodales.
- Soporte para interfaces sensibles al contexto.

Los desafíos antes mencionados se han atacado a partir de la definición de nuevos modelos, la extensión de los modelos existentes, la definición de nuevos tipos de relaciones entre los modelos, así como de información que permite incorporar nuevo conocimiento a los modelos. Ejemplo de esto es el trabajo que se presenta en (Kavaldjian, 07).

El *modelo de tareas* tiene una gran aceptación, debido al soporte que brinda a la generación de interfaces de usuario en una forma orientada a tareas y centrada a la interacción (Forbig *et al*., 04). La notación de Árbol de Tareas Concurrentes (Paterno, *et al*. 97), (CTT, por sus siglas en inglés) implementa un modelo de tareas, el cual describe la funcionalidad de la aplicación por medio de un árbol jerárquico de tareas. Cada tarea se clasifica en uno de cuatro tipos («Abstracción», «Interacción», «Usuario» y «Aplicación»). Éstas se transforman en una interfaz o en un elemento de ella como lo muestra la Figura 5, en la cual se describe la tarea abstracta de *Redactar Correo* descompuesta en sub-tareas

concretas para llevar a cabo su objetivo. El conjunto de las sub-tareas definen la ventana y elementos de interacción que conforman la interfaz de usuario.

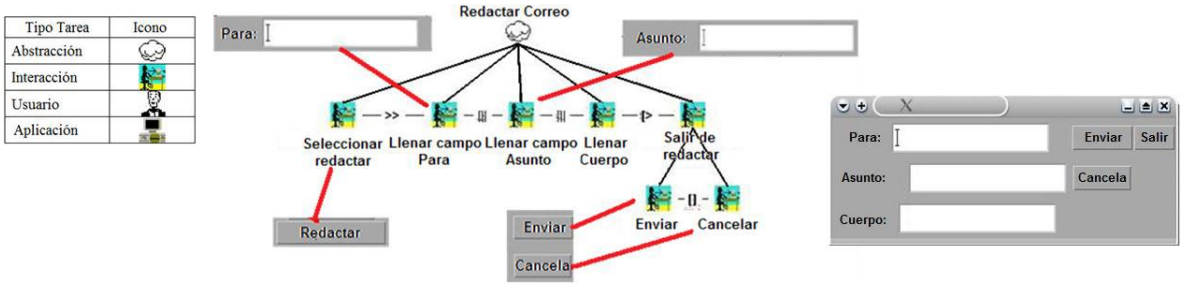

Figura 5. ***Ejemplo gráfico de la representación de la tarea Redactar Correo utilizando la notación CTT.***

El uso de modelos es un componente esencial de cualquier desarrollo de adaptación de interfaces de usuario (Seffah, *et al*., 04a), además el enfoque basado en modelos permite al diseñador enfocarse en sub modelos, más que en la apariencia de la interfaz y además permite la reutilización de código (Hanumansetty, 04).

El enfoque alternativo que se presenta en (Ding *et al*., 06) para el desarrollo de interfaces para varios dispositivos, no parte de modelos formales, sino que se parte del diseño existente de una interfaz de usuario contemplada para dispositivos con pantalla de despliegue grande como una *PC* de escritorio. La idea aquí es que una interfaz final contempla todas las descripciones hechas en los modelos abstractos, de forma que dicha información se puede extraer de la interfaz final a través de un mecanismo, un ejemplo sería el número de botones en la interfaz, los cuales pueden ser contados. Además de esta estrategia se contemplan anotaciones en la interfaz hechas por el diseñador, lo cual provee cierta semántica y fundamentos de diseño.

II.2.1.2 Desarrollo de Interfaces Basado en Patrones

En el área de diseño de interfaces de usuario, un «patrón» es aquel que encapsula una solución probada para un problema de usabilidad. Un desarrollo basado en el enfoque de patrones puede complementar a un «modelo de tareas» al proveer las experiencias acumuladas a través de la retroalimentación del usuario final (Seffah *et al*., 04a). La solución propuesta al problema recurrente debe ser lo suficientemente genérica para poder ser aplicada a diferentes situaciones (Sinng *et al*., 04). Un ejemplo sería la implementación de una «barra de herramientas» (conjunto de *widgets* que presentan las funciones básicas de

una aplicación. En un procesador de texto éstas pueden ser: copiar, cortar, pegar, etc.) para una aplicación en una *PC* y en una *PDA*. En el caso de la *PC*, se puede implementar como la «barra de herramientas» tradicional en una aplicación de escritorio. Pero en el caso de la *PDA* por la disponibilidad de espacio en la pantalla de despliegue se puede implementar como un menú desplegable. Debido a que la solución que provee el enfoque basado en patrones es genérica, ésta puede implementarse de más de una manera. En (Seffah *et al*., 04b) se presenta una lista resumida de estrategias en el uso de patrones.

Tanto el enfoque basado en modelos como el de patrones pueden guiar el proceso de generación de la interfaz para una plataforma determinada o el proceso para generar el «código fuente» de la interfaz en un lenguaje de programación específico. En el desarrollo de interfaces para dispositivos heterogéneos no sólo se involucra la adaptación de la interfaz, cuando se traslada esta idea a dispositivos heterogéneos accediendo recursos en la web se tiene el concepto de adaptación de contenidos.

II.2.2 Adaptación de Contenidos

En el ámbito web el diseño de la adaptación de la interfaz, requiere también la capacidad de adaptación del contenido para presentarse en varios dispositivos mientras se mantiene la consistencia y usabilidad del servicio (Menkhaus *et al*., 02). Un método de adaptación transforma el contenido/información de un estado a otro a fin de satisfacer las restricciones del cliente (Held *et al*., 02; Lemlouma *et al*., 04). Este proceso de adaptación puede incluir transcodificación de formatos (p. ej. *XML* a *WML*, *JPEG* a *WBMP*), redimensión de imágenes, así como de audio y video, conversión de medios (p. ej. texto a voz), omisión o sustitución de partes de un documento (p. ej. la sustitución de una imagen por una representación textual), fragmentación de documentos, traducción de lenguaje, etc. Proveer una lista completa de todos los tipos de adaptación de contenidos no es posible ya que nuevos tipos de adaptaciones surgen con nuevos formatos de datos, aplicaciones, etc. Esta característica es una de las razones por las cuales se dificulta el desarrollo de un sistema único de adaptación que contemple todos los tipos de adaptación (Berhe *et al*., 04). Según su enfoque, relacionado al lugar en el cual se lleva a cabo la adaptación del contenido, las propuestas existentes se pueden clasificar en tres categorías: en el *servidor*, en el *cliente* o a

través de *proxys*. La Figura 6 muestra el punto de vista del desarrollador sobre la «independencia de dispositivos» que es una de las ideas detrás de la «adaptación de contenidos».

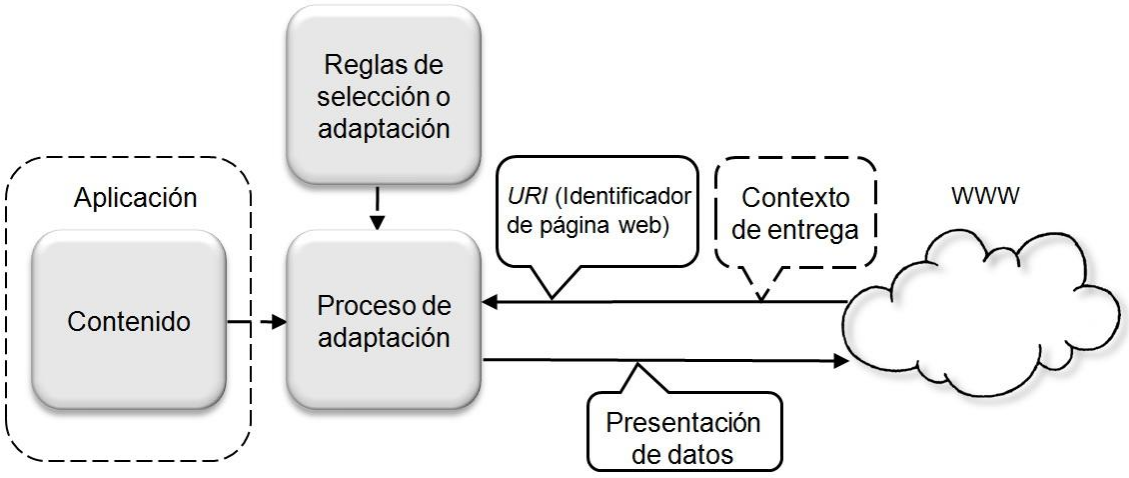

Figura 6. ***Punto de vista del desarrollador de la independencia de dispositivos.***

II.2.2.1 Adaptación de Contenidos en el Servidor

En este enfoque la adaptación se realiza en el servidor, para lo cual se basa en información acerca de las capacidades del cliente y preferencias del usuario, a la cual se puede acceder a través de una *URL* o un repositorio, un punto a considerar aquí es que dicha información pudiera estar desactualizada o errónea.

Sin embargo, la capacidad de procesamiento de cómputo del servidor lo hace más adecuado para llevar a cabo las transcodificaciones necesarias así como otro tipo de adaptaciones.

II.2.2.2 Adaptación de Contenidos por *Proxys*

El concepto de *proxy* en el área de cómputo, se relaciona a una entidad que hace el papel de intermediario entre dos entidades que se comunican. En este enfoque se utiliza de la misma manera, el *proxy* se encarga de mediar el acceso al contenido y determina cómo debe de adaptarse. Aquí el *proxy* realiza las solicitudes al servidor en nombre del cliente, intercepta la respuesta del servidor, realiza la adaptación y envía el contenido adaptado al cliente. Una limitante de este enfoque es que la mayoría de los *proxys* se especializan en algún tipo de adaptación, por ejemplo a la transcodificación de imágenes.

II.2.2.3 Adaptación de Contenidos en el Cliente

Este tipo de adaptación se realiza en el lado del cliente. Debido a que la adaptación se lleva totalmente en el cliente se espera que este tipo de adaptación sea más adecuada que la que haría un servidor, ya que el cliente tiene un conocimiento más detallado de sus características. Este enfoque ya no requiere que las capacidades del dispositivo tengan que estar almacenadas en un repositorio o ser enviadas al servidor cada vez que se realice la solicitud de un servicio. La tarea de transcodificación en el cliente cuando se trata de dispositivos como por ejemplo, *PDA*'s, es limitada debido al poder de procesamiento así como del ancho de banda requerido para el envío de la información.

Algo interesante de este tipo de adaptación, es que se pudieran incluir tomas de decisión del usuario cuando alguna opción adecuada de adaptación no es factible de inferir. Sobre esto último Held en (Held *et al*., 02) menciona que el usuario debe tomar parte del proceso de adaptación para satisfacer sus preferencias, al proveer las prioridades personales. Esto va más allá de tomar parte del proceso de adaptación relacionada a las capacidades del dispositivo, sino que contempla aspectos de usabilidad.

Una vez descritos los enfoques para la adaptación de contenidos, otro enfoque podría contemplar el trabajo en conjunto de los anteriores, el desafío aquí sería la estrategia para discernir cómo asignar quién realiza qué tipo de adaptación, contemplando las posibles adaptaciones necesarias, así como las capacidades de los elementos participantes para llevar a cabo la adaptación asignada.

II.2.3 Adaptando la Interfaz de Usuario

En relación al responsable de la adaptación de la interfaz de usuario en (Thevenin *et al*., 04) se menciona que esto depende de la fase del proceso de desarrollo. En la etapa de diseño, ésta se puede realizar por los diseñadores del sistema o los desarrolladores. En tiempo de ejecución se distingue entre una «interfaz adaptable» y una «interfaz adaptativa». La primera se define como aquella que se adapta a petición del usuario, normalmente se hace a través de menús de preferencias. Por otro lado se dice que es «adaptativa» cuando la interfaz de usuario se adapta bajo su propia iniciativa. Grundy en su trabajo con interfaces de usuario adaptativas (Grundy *et al*., 04) reporta que los usuarios encuentran adecuadas

este tipo de interfaces para sus tareas de aplicación, y en algunos casos las prefieren sobre aquellas implementadas en base al dispositivo o usuario. En su trabajo define la Tecnología de Interfaz de Usuario Adaptable (*AUIT*, por sus siglas en inglés), la cual tiene como objetivo proveer un diseño genérico de pantalla que aumente las implementaciones de servidor web como *JSP* o *ASP*. Se provee un conjunto de etiquetas *XML* independientes de dispositivo para describir los elementos de la pantalla, de esta manera se proporciona una interfaz adaptable para sistemas basados en web. En tiempo de ejecución las etiquetas se transforman al lenguaje particular para el dispositivo objetivo. Las adaptaciones están en función del usuario, tarea y dispositivo. De aquí que a las etiquetas se les definan anotaciones relacionadas con estos últimos para así manipular los elementos.

Como se ha visto tanto en los enfoques para el desarrollo semi-automatico de interfaces de usuario para aplicaciones, así como en la «adaptación de contenidos» en la web es necesario contemplar varios aspectos relacionados con la interfaz de usuario. Un elemento que se puede utilizar en ambos es una descripción que contemple únicamente a la interfaz de usuario, la cual especifique cómo está constituida y posiblemente cómo se relaciona con otros elementos de la aplicación. Esta descripción debe ser abstracta y genérica ya que se busca generar una interfaz concreta funcional para cualquier dispositivo a partir de ella. Una propuesta para dicha tarea son los «lenguajes declarativos de la interfaz de usuario», que se definen en la siguiente sección. A diferencia del enfoque de modelos donde los objetivos a alcanzar dan indicios para definir los elementos a incorporar, aquí se define la «intención» o «propósito» de los elementos que se incorporan a la interfaz.

II.2.4 Lenguajes Declarativos de la Interfaz de Usuario

Este apartado únicamente introduce el concepto, en el Capítulo III se extiende la definición y se presentan algunos lenguajes existentes. Los «lenguajes declarativos de la interfaz de usuario» proponen la descripción de la interfaz de usuario de manera abstracta a través de una estructura y elementos genéricos relacionados a los elementos que conformarán la interfaz que utilizará un cliente, conocida como la «interfaz de usuario concreta», que es aquella con la que el usuario final interactúa y es la manifestación física de la «interfaz abstracta» a través de los recursos propios del dispositivo. Este tipo de lenguajes deben

capturar la «intención» de la interfaz, definiendo *Qué* debe hacer y no *Cómo*, esto se designa como un enfoque declarativo. Estos lenguajes generalmente se basan en el lenguaje de etiquetas *XML*.

Este tipo de lenguajes definen una especificación que rige la forma de estructurar los documentos que describen la interfaz de usuario. Algunos de estos dan soporte para la integración con otros modelos abstractos (usuarios, tareas, diálogo, etc.).

Generalmente la descripción de la interfaz hecha con este tipo de lenguajes se traduce o interpreta a las características del dispositivo específico a través de un procesador dedicado o «interprete», lo cual puede realizarse en el servidor o en el cliente.

Hasta este punto se han descrito adaptaciones enfocadas a las capacidades de los dispositivos, esta situación refleja la consciencia del «Contexto de Uso» para llevar a cabo la adaptación. Trewin en (Trewin *et al*., 04) define el «Contexto de Uso» como los usuarios, tareas, equipo de cómputo, así como el ambiente físico y social en el cual una aplicación se usa. Thevenin en (Thevenin *et al*., 04) lo define como la situación en tiempo de ejecución que describe las condiciones actuales de uso del sistema. El término también se utiliza como «Contexto de Entrega». En el siguiente apartado se introduce el área de «sistemas conscientes del contexto».

II.3 Sistemas Conscientes del Contexto

Se dice que un sistema es «consciente del contexto» si utiliza la información de su entorno para adaptar su comportamiento y brindar así un servicio adecuado a la situación actual. Para dar una definición más formal es necesario establecer en primera instancia lo que es el «contexto».

II.3.1 Contexto

El «contexto» se puede definir como todo aquello que pueda servir para describir la situación actual de una entidad. Acerca de la interpretación que se debe tener de dicho término Dey en (Dey *et al*., 00) indica que se debe relacionar a la relevancia de la información para la situación actual. En su trabajo hace una revisión de algunas definiciones hechas en la literatura, encontrando que algunos autores proporcionan una

definición que resulta muy particular o específica a ciertos casos, lo que hace que se acote el alcance de lo que puede o no ser relevante en una situación, ya que no para todas las situaciones aplican las mismas condiciones y varían de situación a situación. En contraste con estas definiciones, el «contexto» se relaciona a todo sobre la situación actual relevante a la aplicación y a los usuarios. De aquí surge su definición de «contexto», la cual se adopta para este trabajo:

"*El contexto es cualquier información que se puede utilizar para identificar la situación de una entidad. Una entidad es una persona, lugar u objeto que se considera relevante a la interacción entre un usuario y una aplicación, incluyendo al usuario y a la aplicación*".

A partir de esta definición, se define el término «sistema consciente del contexto»:

"*Un sistema es consciente del contexto si utiliza el contexto para proveer información y/o servicios relevantes al usuario, donde la relevancia depende de la tarea del usuario*".

Esta definición se enfoca únicamente a la respuesta que debe tener un sistema en base a su «contexto», dejando a un lado los conceptos relacionados a la recolección e interpretación, los cuales se contemplan como tareas que se deben llevar a cabo por entidades externas. Este término se adopta también para el presente trabajo.

Dourish en (Dourish, 04) hace una distinción en relación a la forma en la que el «contexto» se utiliza en los sistemas de «cómputo ubicuo». Una forma, que es el enfoque más común, es utilizar el «contexto» para adaptar dinámicamente el comportamiento del sistema o su respuesta a patrones de uso. Este enfoque es el de interés para el presente trabajo.

II.3.2 Consciencia del Contexto en la Adaptación de Interfaces de Usuario

Si bien la definición de «consciencia del contexto» dada en el apartado anterior hace referencia a la respuesta que tiene un sistema en función de su «contexto», el uso de la información relacionada a las capacidades del dispositivo así como de las preferencias del usuario al momento de adaptar una interfaz de usuario denota justamente esa «consciencia del contexto». Dentro del ámbito web se denomina «contexto de uso» o «contexto de entrega». Como se menciona, la definición anterior deja a un lado toda tarea que no se relacione con el uso de la información para adaptar su comportamiento. Las tareas de

recolección, interpretación, diseminación, etc. de la información contextual son propias de la arquitectura de la cual es parte el sistema.

Un término propicio a mencionar en este punto es la «plasticidad» de la interfaz de usuario. El término está inspirado de los materiales que se expanden y contraen bajo sus límites naturales sin romperse, preservando así su uso continuo. Aplicado al área de Interacción Humano-Computadora (*HCI*, por sus siglas en inglés), la plasticidad es la 'capacidad de un sistema interactivo de dar soporte a varios contextos de uso mientras preserva su usabilidad' (Thevenin *et al*., 99; Thevenin *et al*., 04).

Bajo los enfoques ya presentados, las adaptaciones que se contemplan, incluyendo la de la interfaz de usuario se basan generalmente en las capacidades del dispositivo (tamaño/resolución de pantalla, soporte de color, etc.), características de la red (ancho de banda, retardo, tasa de error, etc.), y preferencias del usuario (tipo de servicio, formato de presentación, idioma, etc.). Hanumansetty en (Hanumansetty, 04) reporta que pocos enfoques toman en cuenta aspectos contextuales como la ubicación, actividad o compañía de la persona para la adaptación de la interfaz de usuario.

Debido a la diversidad de dispositivos existentes y de esto el conjunto de consideraciones a realizar no es factible definir una estrategia única que contemple todo el rango de adaptaciones posibles, de aquí que se sigan proponiendo estrategias de adaptación que abarcan un rango específico de alcance, es decir, las «plataformas» contempladas, con esto se hace referencia a una combinación de *hardware*, capacidades de cómputo, sistema operativo y un *toolkit* para la interfaz de usuario. Ejemplo de esto son los trabajos que se presentan en (Mohomed *et al*., 06), (Butter *et al*., 07) y (Feuerstack *et al*., 08).

II.3.3 Propuesta de Adaptación

El interés del presente trabajo es la adaptación de la «interfaz de usuario» en relación al «contexto» actual más allá de sólo las capacidades del dispositivo y preferencias de usuario, para incluir aquellas relacionadas con la ubicación, compañía, actividad, etc. No sólo la «plataforma» se debe considerar como un factor de influencia contextual en las aplicaciones (Clerckx *et al*., 05). Para tal efecto, el enfoque propuesto parte del uso de un «lenguaje declarativo de la interfaz de usuario» existente, y a partir de éste se propone una

estrategia que involucra tres etapas de adaptación relacionadas a: la generación de la «interfaz abstracta», la traducción de dicha «interfaz abstracta» a una «interfaz concreta» y las adaptaciones predefinidas que pueda sufrir la «interfaz concreta» que son disparadas en función de los cambios en el «contexto» durante el tiempo de ejecución. La primera adaptación se lleva a cabo en un servidor y las otras dos se llevan a cabo en el cliente.

Una de las principales consideraciones a tomar, es que los «lenguajes declarativos de la interfaz de usuario» no contemplan información explícita acerca del «contexto», ya que ésta debe tomarse en cuenta durante el proceso de generación de la interfaz funcional (Trewin *et al*., 04).

En la estrategia propuesta, la adaptación que se relaciona con las capacidades del dispositivo tal como se maneja en el enfoque tradicional se deja a la especificación del *UIDL* y un «intérprete» correspondiente existente, de esta manera las adaptaciones de la «tercera etapa de adaptación» se enfocan a la estructura e información que se presenta en la «interfaz de usuario concreta» funcional una vez que la descripción abstracta se adapta para el dispositivo. La idea es proponer un medio para vincular los cambios en el «contexto» y las adaptaciones predefinidas para la interfaz de usuario.

Bajo la premisa anterior, se selecciona *XForms* (Boyer, 07) como el lenguaje base para la definición abstracta de la interfaz de usuario, la idea general es crear dinámicamente el documento que especifica la «interfaz abstracta» en el lenguaje *XForms*, a partir de un modelo que represente el «contexto» actual, un modelo que defina las características de los elementos a incluir en la «interfaz abstracta» incluyendo si es dependiente del «contexto» y un conjunto de reglas específicas que guíen la generación del documento *XForms*. Tanto los modelos como las reglas se especifican en documentos *XML*. Estos tres documentos sirven de entrada para una aplicación encargada de generar el documento *XForms*. Se elige *XForms* en base a una revisión de la literatura, la cual se presenta en el Capítulo III. Dicha revisión refleja su ventaja sobre otros lenguajes de su tipo, aunado a esto se toma en consideración los mecanismos que provee para definir un comportamiento dinámico de la interfaz.

Justamente los mecanismos provistos por la especificación *XForms*, dan la pauta para describir las adaptaciones predefinidas que ha de sufrir la «interfaz de usuario concreta»,

dichas adaptaciones se definen en la «interfaz abstracta». De esta manera el mismo «intérprete *XForms*» que se encarga de la conversión de la interfaz a la representación adecuada del dispositivo en cuestión, es el encargado de ejecutar las adaptaciones predefinidas.

Se parte de la visión de que la estrategia trabaja bajo una arquitectura cliente-servidor, en la cual un servidor junto con una infraestructura adecuada es el encargado de monitorear el «contexto» y hacerlo conocer al cliente.

La idea detrás de esta estrategia es poder integrarla dentro de una arquitectura que dé soporte a aplicaciones «conscientes del contexto». La definición de una arquitectura de este tipo escapa del alcance del trabajo, sin embargo es necesario hacer una revisión de trabajos relacionados con la definición de este tipo de arquitecturas para así tomar decisiones de diseño para encaminar la definición de la estrategia de adaptación.

II.3.4 Frameworks para el Manejo de Información Contextual

Uno de los desafíos dentro del desarrollo de aplicaciones «conscientes del contexto» es la estrategia para el manejo de la gama de información contextual, de manera que las aplicaciones sean capaces de utilizar dicha información. La división de tareas para administrar el contexto contempla la recolección, análisis, conversión, inferencia, manejo de incertidumbre, diseminación, entrega, entre otras.

Algunos trabajos proponen arquitecturas de *software* que brindan un marco de referencia para guiar la implementación de aplicaciones «conscientes del contexto». Una característica común en estos trabajos es justamente los componentes que se encargan de comunicar los cambios en el «contexto», que actúan como un puente entre los componentes encargados de la recolección, interpretación, etc. y los componentes que consumen o utilizan dicha información. El enfoque que presentan para dicha tarea se describe como un modelo de programación basado en eventos.

Un *framework* que dé soporte a la «consciencia del contexto» debe satisfacer (Hannumansetty, 04):

- Contar con tecnología que permita el sensado de información del ambiente.

- Debe dar soporte para un modelo de programación basado en eventos así como brindar la habilidad de disparar eventos cuando cierto cambio en el contexto se observe.
- Proveer un medio para comunicar los datos contextuales sensados a otros elementos en el ambiente y un mecanismo para interpretar los datos recolectados.

II.3.4.1 Trabajos Relacionados

Esta sección recopila algunos trabajos propuestos en la literatura para el desarrollo/soporte para aplicaciones y sistemas conscientes del contexto. Dentro de estas propuestas se contemplan *frameworks* para asistir al desarrollo de aplicaciones y diseños arquitectónicos para guiar la construcción de aplicaciones que utilicen información contextual.

- *Context Toolkit* (Salber *et al*., 99; Dey *et al*., 01) es una infraestructura que permite el desarrollo de servicios «conscientes del contexto». Dicha infraestructura asume una descripción explicita del «contexto». Su arquitectura permite a las aplicaciones conocer la información contextual que requieran sin la necesidad de saber cómo fue sensada u obtenida. La información se hace accesible al entorno de ejecución de la aplicación y permite al diseñador decidir qué información es potencialmente relevante y cómo manejarla. La propuesta se constituye de: los *controles de contexto* que sensan el «contexto» de manera implícita, los *agregadores* que recolectan el «contexto» relacionado, los *intérpretes* que realizan conversiones entre tipos de «contexto» y lo interpretan, las *aplicaciones* que utilizan el contexto y una *infraestructura de comunicación* que se encarga de entregar el «contexto» a estos componentes distribuidos. El *toolkit* facilita la incorporación de uso del «contexto» o entradas implícitas a aplicaciones existentes. En relación a la respuesta a cambios en el «contexto» manejada a través de eventos, se indica que un enfoque secuencial para atender las notificaciones para la adaptación no es apropiado cuando se trata del «contexto». Un enfoque en el cual cada notificación se maneje por un hilo de ejecución independiente no es una buena alternativa para la escalabilidad. El enfoque adoptado, es que los controles actúen como fuentes de «contexto» entregando la información a las aplicaciones interesadas, de esta manera se tiene un

enfoque de flujo de datos. Este trabajo hace una clara división de componentes que incorporan, recolectan e interpretan el «contexto». Clerckx en (Clerckx *et al*., 05), menciona que aunque el método propuesto contempla el soporte para la construcción de aplicaciones «conscientes del contexto», no se da énfasis al efecto y uso del «contexto» en la interfaz de usuario.

- Hanumansety en (Hannumansetty, 04) presenta un *framework* para dar soporte al desarrollo de aplicaciones de *software* interactivas y ubicuas, así como para la creación de interfaces de usuario sensibles al contexto. Son cinco los componentes que constituyen el *framework*: *Servidor de información y lógica de aplicación*, *Servidor de contexto*, *Herramienta de modelado del contexto*, *Servidor de Interfaz y Modelado* y el *Dispositivo cliente*. El *Servidor de contexto* es el encargado de administrar la información contextual y comunicar los cambios en el «contexto» al *Servidor de Interfaz y Modelado*, el cual se encarga de los mapeos correspondientes entre los modelos contemplados. La estrategia de construcción y adaptación de la interfaz se basa en cuatro modelos de abstracción, y los mapeos entre estos. El *Servidor de contexto* comunica los cambios en el «contexto» a través de eventos, cada tipo de evento notificado da la pauta para saber en qué etapa de la generación de la interfaz se debe comenzar la regeneración de ésta en caso de requerir una adaptación. La adaptación se realiza en su totalidad en el servidor y la interfaz es reenviada al cliente.
- Alamán en (Alamán *et al*., 03) presenta la propuesta de un *framework*, para la generación dinámica de interfaces de usuario el cual utiliza un protocolo basado en eventos, que permite la interacción entre la interfaz de usuario y el ambiente. La idea es comunicar todos los cambios en un repositorio común al cual esté suscrito todo aquel interesado en conocer los cambios. Esta arquitectura permite un bajo acoplamiento entre las capas pertinentes.
- En Lei (Lei, 05) se define un servicio general de «contexto», el cual provee una infraestructura para la recolección y diseminación de información contextual. Provee un nivel de abstracción de programación para dar soporte a la composición de datos contextuales así como la vinculación de diferentes fuentes. Uno de los

cuatro componentes denominado el Motor de Eventos es el encargado de verificar por correspondencias entre los eventos contextuales y aquellos de interés registrados por una aplicación. Para tener acceso a la información contextual un cliente debe establecerlo a través de una plantilla, la cual es manejada por el servicio de contexto que realiza las correspondencias en cuestión.

- En (Indulska *et al.*, 03) se propone un sistema administrador del «contexto» para dar soporte a una infraestructura de sistema ubicuo. Para tal se define una arquitectura de tres capas cada una correspondiente a la aplicación «consciente del contexto», la aplicación gestora del «contexto» y la recolección de la información contextual. Las aplicaciones se suscriben a eventos relacionados a los cambios en el «contexto» de interés.
- DynaMo-AID (Clerckx *et al.*, 05), provee un ambiente de desarrollo integrado para la implementación, y demás tareas afines, de sistemas interactivos «conscientes del contexto». Se define una arquitectura para dar soporte en tiempo de ejecución. La división de tareas asignadas a los componentes de la arquitectura refleja una delegación de responsabilidades según sus capacidades. Definir responsabilidades en base a las capacidades presenta ciertas ventajas, por ejemplo el componente denominado «Unidad de Control de Contexto», que es el encargado de sensar el «contexto» apoyado de otros componentes, se implementa en un servidor externo, de esta manera puede encargarse de tareas complejas, liberando a cualquier dispositivo móvil de dichas tareas. Además se puede establecer una estrategia basada en eventos que permite dar soporte a «interfaces de usuario» distribuidas.

II.3.5 Describiendo más que la Interfaz de Usuario

Hasta este punto se han contemplado únicamente lenguajes de etiquetas propios para representar la interfaz de usuario de manera abstracta, sin embargo es necesario contemplar otro tipo de elementos que afectan a la interfaz de usuario que se entrega al cliente. Por ejemplo, en la «adaptación de contenidos», una imagen de una resolución apropiada para un dispositivo como lo sería una *PC* de escritorio, no es necesariamente adecuada para un dispositivo con una pantalla reducida como lo sería la de una *PDA*. Aquí se puede

considerar más de un camino para la resolución de dicha cuestión, si el dispositivo no soporta imágenes ésta se puede omitir, por el contrario si ofrece soporte, es posible realizar una transcodificación de este medio, ya sea convertirlo a un formato que el dispositivo sea capaz de manejar o una posible redimensión de ésta. Lo mismo con otro tipo de medios como audio, video etc. Dichas decisiones se pueden plasmar en el código de la aplicación, especificadas como reglas, o a través de algún «lenguaje independiente de dispositivo», cuyo cometido va más allá de la interfaz de usuario ya que definen descripciones que contemplan otro tipo de adaptaciones incluyendo las mencionadas anteriormente.

Un ejemplo es el «Lenguaje de Autoría Independiente de Dispositivo» (*DIAL*, por sus siglas en inglés) (Smith, 07) del Consorcio de la *World Wide Web* (*W3C*, por sus siglas en inglés). El propósito de *DIAL* es proporcionar un lenguaje de etiquetas para el filtrado y presentación de contenido de una página web a través de diferentes «contextos de entrega». Es un lenguaje basado en vocabularios *XML* existentes y módulos de «hojas de estilos en cascada» (*CSS*, por sus siglas en inglés). Estos proveen los mecanismos estándar para representar la estructura de la página web, presentación y forma de interacción. *DIAL* se apoya en el vocabulario de metadatos *DISelect* (Lewis *et al*., 07) otra propuesta de la *W3C*, para sobrellevar los desafíos propios de la autoría para múltiples «contextos de entrega». *DISelect* especifica una sintaxis y un modelo de procesamiento para un filtrado y selección de contenido de propósito general. La selección se realiza sobre un conjunto de información *XML*, a través de un procesamiento condicional en base al resultado de la evaluación de expresiones. Usando este mecanismo algunas partes de información se pueden seleccionar para un procesamiento posterior y otras se pueden suprimir. La especificación de las partes del conjunto de información *XML* afectada y las expresiones que dictan el procesamiento se hacen a través de una sintaxis *XML*. Esto incluye elementos, atributos y expresiones en *XPath* (Clark *et al*., 99), el cual es un lenguaje que permite construir expresiones para recorrer y procesar un documento *XML*. *DISelect* especifica cómo los componentes trabajan en conjunto para proveer una selección de propósito general. Estas iniciativas se encuentran en desarrollo a cargo del grupo *Ubiquitous Web Applications Actvity*[2] de la *W3C* antes

[2] http://www.w3.org/2001/di/

denominado *Device Independence*[3], cuyo cometido es buscar la extensión de la web para permitir aplicaciones distribuidas en una diversidad de dispositivos.

II.4 Resumen y Conclusiones

En este capítulo se hace una breve descripción de la adaptación de la «interfaz de usuario» desde el punto de vista del desarrollo de una aplicación para distintos dispositivos así como la relacionada a la «adaptación de contenidos» cuando se acceden servicios de la web y el papel de los «lenguajes declarativos de la interfaz de usuario» en ambos enfoques. Una vez hecha la distinción se introduce el concepto de «sistemas conscientes del contexto» a partir del cual se hace noción de cómo el «contexto» se utiliza para llevar a cabo la adaptación pertinente.

Como se hace evidente no es posible una estrategia única de adaptación lo que hace necesario una división de áreas de adaptación, ejemplo de esto es el área de «adaptación de contenidos» cuando se contempla la web.

En el caso concreto de la propuesta de trabajo, ésta se relaciona a la definición de una interfaz de usuario adaptativa usando como base el lenguaje *XForms*. La adaptación de interés es la que contempla el «contexto» del usuario, es decir su compañía, actividades, etc., sin dejar a un lado las capacidades de los dispositivos.

[3] http://www.w3.org/2007/uwa/

Capítulo III

Lenguajes Declarativos de la Interfaz de Usuario

III.1 Introducción

El Capítulo anterior presenta de manera breve los «lenguajes declarativos de la interfaz de usuario» y su importancia para el desarrollo de interfaces para dispositivos heterogéneos así como para la «adaptación de contenidos».

En este capítulo se aborda el tema y se presenta una recopilación de algunos lenguajes declarativos existentes, así como de trabajos que realizan revisiones y comparaciones entre estos, destacando sus ventajas y desventajas. Los resultados expuestos en dichos trabajos así como la revisión literaria definen la decisión de la elección de *XForms* como el lenguaje base para definir la «interfaz de usuario abstracta genérica» de la estrategia de adaptación.

Para el presente trabajo, es de interés la propuesta presentada por los lenguajes declarativos para describir «interfaces de usuario abstractas genéricas» basados en *XML*.

III.1.1 Enfoque Declarativo

La naturaleza de los lenguajes declarativos permite describir la «intención» de la interfaz de usuario, al definir *Qué* debe hacer y no *Cómo*, de esta manera escapa de cualquier interpretación particular. La idea es describir la interfaz de usuario de manera abstracta y genérica. La representación no supone alguna modalidad en particular (visual, auditiva, táctil, etc), dispositivo, lenguaje (de programación o etiquetas), plataforma, forma de presentación al usuario, etc.

A partir de la descripción dada por la «interfaz de usuario abstracta», se puede generar la «interfaz de usuario concreta» en algún lenguaje de etiquetas particular (p. ej. *HTML*, *VoiceXML*) o el código fuente en algún lenguaje de programación.

Con respecto al concepto de «independencia de dispositivos» dentro de la *W3C* se relaciona a la habilidad de poder acceder a alguna información independientemente del dispositivo que se utilice y ésta siempre estará disponible y accesible al usuario. Es justamente el enfoque declarativo de este tipo de lenguajes relacionado a la captura de la «intención» de la interfaz de usuario lo que permite dar ese soporte a la «independencia de dispositivos».

El uso de un lenguaje declarativo para describir la interfaz de usuario supone los siguientes beneficios (Menkhaus *et al*., 02):

- Separación natural del código de la interfaz de usuario y el código relacionado a la funcionalidad de la aplicación.
- Facilita un rápido prototipado.
- Permite la creación de una familia de interfaces que mantienen sus características comunes.

III.1.2 Intérpretes

Un «intérprete» es una aplicación de *software* que lee la descripción abstracta de la «interfaz de usuario» hecha con el *UIDL* y realiza la traducción a una «interfaz concreta funcional» o a código fuente. Dicho «intérprete» conoce la plataforma y demás características del dispositivo objetivo.

Cuando se dice que un *UIDL* provee un «mecanismo» para una situación o caso particular, lo que se indica es que su estructura permite reflejar dicha situación, ya que está contemplado dentro de su especificación.

La implementación de dichos mecanismos para la plataforma a la que se traduce la descripción abstracta se realiza por el «intérprete». La traducción se puede llevar a cabo en el servidor o en el cliente, para lo cual hay que dotarlos con un «intérprete» correspondiente. Las ventajas y desventajas de uno y otro enfoque se relacionan directamente a temas relacionados con el poder de procesamiento de cómputo y limitaciones afines.

La Figura 7 ejemplifica el concepto de traducir la representación abstracta de un control a las capacidades de un dispositivo. En el ejemplo se presenta un control que permita realizar la selección de un elemento de una lista. En el ejemplo se debe seleccionar el tipo de tarjeta de crédito dentro de un conjunto definido. En primera instancia se tiene la traducción hecha para un celular y por otro lado su contraparte en una aplicación para *PC* respectivamente.

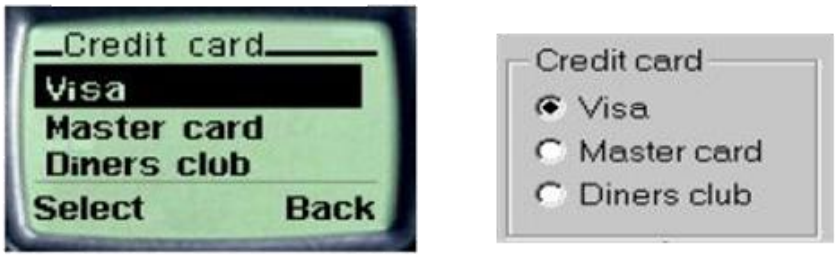

Figura 7. ***Selección de un elemento en una lista y su interpretación en un celular y en una aplicación para PC respectivamente.***

Este tipo de lenguajes se basa generalmente en *XML*, el cual es un lenguaje de marcas o etiquetas. Un lenguaje de etiquetas permite definir un documento, ya sea su estructura o presentación, al incorporar junto al texto, etiquetas o marcas que contienen la información de la estructura o presentación. *XML* permite estructurar la información en un archivo además de dar indicios de cómo las aplicaciones deberían interpretarlo. *XML* se define como un metalenguaje ya que su estructura permite definir otros lenguajes. *XML* permite separar la funcionalidad de la aplicación de la estructura de datos.

En la literatura existen un conjunto de *UIDL*'s, sin embargo, algunos de ellos se encuentran en una etapa experimental, de desarrollo, han sido desarrollados para condiciones específicas (escenarios de aplicación, lenguaje, aplicación, etc.), o simplemente no cuentan con soporte por parte de terceros lo cual los hace dependientes de ciertas tecnologías. Cada lenguaje ha sido creado con un objetivo común, sin embargo, cada uno tiene un enfoque diferente, de aquí que algunos sean más adecuados que otros a ciertas condiciones impuestas por el desarrollo de una aplicación o la plataforma, por ejemplo, si estos dan mayor peso a la presentación, a los elementos de interacción o incluso el mismo soporte para ciertas tecnologías o lenguajes de programación. Además, algunos de estos lenguajes

cuentan con un gran trasfondo de investigación y desarrollo que los respalda, incluso permitiendo soporte por parte de terceros.

Una de las mayores desventajas de describir la «interfaz de usuario» por medio de un lenguaje declarativo es la necesidad de un motor de presentación o «intérprete», otra desventaja de este tipo de lenguajes es que se requiere que el desarrollador aprenda un nuevo lenguaje o herramienta, así como el trabajar a un nivel más abstracto del que acostumbra. Además, las interfaces generadas automáticamente a partir de dicha descripción conllevan problemas de usabilidad, ya que el proceso de generación desconoce del significado real y carece de una comprensión profunda de la funcionalidad del servicio o aplicación (Göschka *et al*., 01; Trewin *et al*., 04), a pesar de esto su uso presenta ventajas destacables (Göschka *et al*., 01):

- La descripción de la interfaz de usuario es independiente de cualquier plataforma o lenguaje de programación.
- El uso de un *framework* estandarizado internacionalmente, simplifica el proceso de definición y fomenta el uso del lenguaje por otras compañías.
- La interpretación personalizada de una descripción, permite alcanzar un uso más adecuado de la funcionalidad específica de una plataforma sencilla, ya que las transformaciones no se limitan a relaciones 1 a 1.
- La personalización de la interfaz de usuario puede realizarse fácilmente al modificar las características del «intérprete».

Conocer el contexto ayudaría a definir interfaces personalizables, que permitirían alcanzar las metas de efectividad y usabilidad de éstas, tal como lo demuestra Shankar en (Shankar, *et al*., 07), quien utiliza técnicas de aprendizaje de máquina para aprender del comportamiento de las personas cuando son expuestas a diferentes versiones de una misma interfaz de usuario. Su estudio le permite descubrir las preferencias de los usuarios con lo que puede definir interfaces personalizables para cada individuo. Incluso además del contexto, la ubicación física, juega un papel importante en la manera en la que una persona percibe y maneja información, como se muestra en (Elliot, *et al.*, 07), donde se ilustra cómo estos dos elementos influyen en la forma en la que los miembros de una familia perciben la información desplegada en un dispositivo en un momento dado.

III.2 Revisión de Lenguajes Declarativos Existentes

En primera instancia se presentan de manera general cuatro lenguajes declarativos existentes, después de esto se presenta la recopilación de un conjunto de trabajos que realizan revisiones a algunos de estos lenguajes, desde el cumplimiento de ciertos estándares hasta evaluaciones para determinar en qué medida dan soporte a escenarios de aplicación planteados. Dichos trabajos realizan evaluaciones más exhaustivas a estos lenguajes, respecto a sus características, ventajas, soporte, etc., que la revisión que contempla este trabajo, por lo tal se hace uso de los resultados expuestos. Como se ha mencionado *XForms* se selecciona entre un conjunto de *UIDL*'s en base a los resultados reportados en los trabajos mencionados anteriormente, así como en la revisión de dichos lenguajes en relación a su posible extensibilidad, tecnologías existentes, acceso, etc. La decisión de utilizar un lenguaje existente y no proponer un lenguaje desde su inicio, se debe al trabajo de investigación y desarrollo que existe detrás de estos. Obviamente la selección es una de las decisiones principales de este trabajo ya que a partir del lenguaje se define la estrategia de adaptación que contempla además, el modelado del contexto, la definición de un modelo que defina las características de los elementos a incluir en la «interfaz abstracta», la definición de las reglas de adaptación que relacionan el modelo del contexto con el modelo anterior, así como la aplicación que genera el documento *XForms*.

XIML

XIML (eXtensible Interface Markup Language), es un lenguaje que permite representar y manipular elementos de interacción tanto a nivel abstracto como concreto. Básicamente *XIML* es una colección organizada de elementos de la interfaz que se categorizan dentro de uno o más componentes mayores. Está diseñado para dar soporte a todo el ciclo de vida del desarrollo de la interfaz, para esto define cinco modelos declarativos que considera básicos para la interfaz: tarea, dominio, usuario, diálogo y presentación. Cada uno de estos define su propia jerarquía y representa una entidad abstracta (tarea, dominio y usuario) o concreta (diálogo y presentación), para la generación de la interfaz de usuario.

UIML

UIML (User Interface Markup Language) (Phanouriou, 00; Trewin, *et al.*, 04), se propone como un formato de intercambio universal que puede representar cualquier interfaz de usuario, independiente del modo de interacción, lenguaje, sistema operativo, y plataforma. *UIML* no define etiquetas para generar interfaces de usuario. En cambio, la especificación se limita a definir las partes de un documento UIML, y las reglas mediante las que se puede establecer un lenguaje de etiquetas para una determinada aplicación, utilizando la estructura proporcionada para tal. La idea es tener una especificación por cada lenguaje de programación o etiquetas objetivo que se desee y definir los mapeos entre estos y los elementos *UIML*.

XForms

XForms (Dubinko, 03; Cagle, 05), se ha propuesto como la tecnología para remplazar los formularios web actuales, los cuales incorporan muchos elementos *script* para llevar a cabo validaciones y cálculos. *XForms* minimiza el uso de *scripts* al incorporar este tipo de acciones de manera nativa en su especificación. *XForms* en su versión 1.0 es una recomendación de la *W3C*.

XForms separa los datos y la lógica de una forma de su presentación. De esta manera los datos de la forma pueden definirse de manera independiente de la interacción del usuario final. *XML* se utiliza para definir las reglas que describen y validan los datos de la forma. Los datos que son desplegados en una forma, y los datos que son ingresados a la forma, son enviados al servidor utilizando *XML*. Además *XForms* depende de Hojas de Estilo en Cascada (*CSS*, por sus siglas en inglés), para definir la presentación, de tal manera que el desarrollador tiene total control sobre ésta (Lamadon *et al*., 04). *XForms* se aborda con más detalle en la sección III.3.

XUL

XUL (XML-based User Interface Language), tecnología propuesta por la fundación Mozilla para definir un sistema multiplataforma de descripción de interfaces de usuario, a través de *XML*. *XUL* Permite crear interfaces que son interpretadas y mostradas al usuario por *Gecko*, el motor gráfico integrado en el navegador *Mozilla Firefox*.

III.2.1 Revisión de Lenguajes

Esta sección presenta un recopilado de trabajos que realizan una revisión a algunos de los lenguajes declarativos existentes. Cada trabajo analiza distintos aspectos de los lenguajes, de aquí que no sea posible definir una referencia homogénea sobre los resultados presentados en ellos, por tal, los resultados se sintetizan en base a los puntos de interés de este trabajo. Los lenguajes a los que se les da mayor énfasis y sobre los que se presentan dichos resultados son los presentados anteriormente: *XIML*, *UIML*, *XForms* y *XUL*.

En (Souchon *et al*., 03) se hace una recopilación sobre la metodología de trabajo (herramientas, modelos, lenguajes soportados, etc.) y características relacionadas a la especificación y su alcance (abstracción, número de etiquetas manejadas, expresividad, etc.) de un conjunto de lenguajes declarativos. Debido al alcance de la revisión hecha se limita a comentar que la adopción de alguno de estos lenguajes debe regirse por los objetivos de la aplicación a desarrollar. De los lenguajes de interés sólo contempla *XIML*, *UIML* y *XUL*, *XForms* no lo consideran argumentando su desarrollo incipiente al momento de desarrollar el estudio, y su enfoque al desarrollo de formas web.

En (Cogliati, 06) se reportan ventajas y desventajas y se revisan los requisitos de los distintos tipos de interfaces de usuario existentes, y en base a la revisión de las capacidades de algunos lenguajes reportan que las herramientas de desarrollo y metodologías no conducen de manera adecuada la diversidad de desafíos para la adaptación de la interfaz de usuario. Esta observación se relaciona como se menciona en el Capítulo II, con la falta de una metodología única para el soporte de todas las adaptaciones posibles. En dicho trabajo se abarcan los cuatro lenguajes de interés.

En (Lamadon, *et al*., 04) se evalúan tres lenguajes (*XIML*, *UIML* y *XForms*) en base a requisitos técnicos y generales tanto para el usuario como para el desarrollador. A partir de dos de los lenguajes (*UIML* y *XForms*) se desarrolla una implementación para un escenario real. Se reporta a *XForms* como el lenguaje que mejor se apegó a los requisitos dados.

En (Hurtado *et al*., 04) se hace una comparación entre lenguajes destacando su relación con el modelado de interfaces de usuario basado en modelos declarativos y sus fases o niveles de abstracción. En dicho trabajo se abarcan los cuatro lenguajes de interés.

En (Trewin, *et al*., 03) se analiza si las arquitecturas propuestas por *UIML*, *XIML*, *XForms*, *AIAP* dan soporte a los requisitos de usabilidad universal al permitir la definición de interfaces personalizables al usuario. *XForms* y *AIAP* se reportan como los dos lenguajes que mejor cumplen con los requisitos expuestos.

En (Trewin, *et al*., 04) se evalúan cuatro lenguajes (*UIML*, *XIML*, *XForms*, *URC*) en relación a un conjunto de requisitos definidos por un escenario real. Se reporta a *XForms* y *URC* como los dos lenguajes que mejor cumplen con los requisitos propuestos.

En (Pohja, *et al*., 06) se hace una comparación entre *XForms* y *XUL*, en base a la implementación de dos casos de uso y los requisitos para estos. Además, se valida el cumplimiento de los requisitos que debe cumplir un *UIDL* según la literatura. *XForms* se reporta como el lenguaje que mejor cumple los requisitos impuestos.

En relación a los requisitos que debe cumplir un *UIDL*, (Trewin *et al*., 04), (Lamadon *et al*., 04) y (Pohja *et al*., 06), adoptan los propuestos por (Trewin *et al*., 03) e incluso extienden o ajustan a su propósito, y justamente a partir de estos realizan parte del análisis presentado en sus trabajos.

La Tabla I presenta una síntesis comparativa de las características de los lenguajes, adoptando y extendiendo los trabajos que se revisan. Las ventajas y desventajas se describen textualmente más adelante. A continuación se definen las características reflejadas en la tabla.

- Datos y Presentación, denota si la definición de los controles de la interfaz de usuario se hace de manera independiente de la presentación.
- Expresividad, se relaciona con la capacidad para definir en menor número de sentencias un concepto, así como de los mecanismos para su manejo.
- Extensible, se define como la opción para la introducción de nuevos conceptos o etiquetas que no han sido definidos en la especificación.
- Interfaz, este punto indica si el tipo de interfaz generada a partir de su especificación es únicamente una «interfaz de usuario gráfica» (GUI) o contempla otro tipo de interfaces (IU).

- Soporte a modelos, este punto sólo se limita a mencionar si la especificación contempla soporte a algún modelo declarativo de la interfaz de usuario sin entrar en detalle sobre cuáles.

Tabla I. Tabla comparativa de las características de los lenguajes declarativos.

Característica/Lenguaje	**XIML**	**UIML**	**XForms**	**XUL**
Datos y Presentación	Separado	Combinado	Separado	Separado
Expresividad	Media	Media	Alta	Baja
Extensible	Si	No	Si	No
Interfaz	UI	UI	UI	GUI
Soporte a Modelos	Si	Si	No	Si

Algunas de las desventajas encontradas se enumeran a continuación:

- *XIML*, está sujeto a licencias para su uso, además no ofrece soporte a tecnologías de terceros. No está enfocado a una gran diversidad de dispositivos, y el kit de desarrollo incluye un intérprete único para dicha tarea.
- *UIML*, es un lenguaje de gran complejidad. Cuando fue diseñado se contempló la generación de la interface concreta, por lo tal, es necesario estar definiendo vocabularios que contienen información de un lenguaje particular, así como las reglas de mapeo para vincular la interfaz abstracta con el vocabulario.
- Tanto *XIML* como *UIML*, requieren de mecanismos externos para conocer el estado de la interfaz en un momento dado.
- *XUL*, depende de *Gecko* el cual se encuentra integrado en el navegador *Mozilla*, además, depende del lenguaje *javascript*, lo que obliga a desarrollar ciertas características de las aplicaciones utilizando dicho lenguaje *script*.
- *XForms* no provee soporte para otros modelos declarativos (como el de tareas, dialogo, etc.) para la generación de la interfaz.

Para la selección del «lenguaje declarativo de la interfaz de usuario» se decide adoptar *XForms* como el lenguaje base de la propuesta de trabajo. *XUL* se descarta debido a su alta dependencia de *scripts*, *UIML* debido a la falta de separación de los datos y presentación, además de requerir la definición de vocabularios específicos para cada lenguaje objetivo. *XIML* a pesar de presentar varias ventajas se descarta debido a la necesidad de contar con

otras aplicaciones externas a las que provee su *kit* de desarrollo y por carecer de soporte a terceros.

Los trabajos que se presentan en esta sección proveen un conjunto de fundamentos, en base a evaluaciones técnicas, sobre la ventaja que presenta *XForms* sobre otros *UIDL*'s. La tabla comparativa presentada anteriormente únicamente resume algunas características de *XForms* en comparación con otros lenguajes. Dichas características involucran aquellas que idealmente debería cumplir un *UIDL*.

Independientemente del cumplimiento de dichas características, uno de los principales objetivos de este tipo de lenguajes es dar soporte para la independencia de dispositivos. Esta idea surge a partir de la variedad tecnológica de dispositivos de cómputo heterogéneos en capacidades y la necesidad de los usuarios de poder utilizar cierto tipo de aplicaciones o acceder a recursos en la web independientemente del dispositivo utilizado. Contemplando esta necesidad, este tipo de lenguajes generalmente es utilizado en dos posibles situaciones:

- En primer lugar, pueden asistir en la generación semi-automática del código fuente de la «interfaz de usuario» en un lenguaje de programación especifico, con la idea de facilitar el proceso de generación de interfaces de usuario como módulos independientes dentro de una aplicación.
- En segundo lugar, pueden ser utilizados dentro de ambientes tecnológicos donde el enfoque principal requiera generar la interfaz de usuario en un «lenguaje de etiquetas» y entregarla al cliente como podría darse en la web, pero este tipo de ambientes no se limitan únicamente a la web, inclusive este mismo enfoque puede darse dentro de una red de área local donde se tenga un ambiente aumentado con tecnología ubicua, en el cual se cuente con terminales a modo de clientes delgados no necesariamente con navegadores, sino con un entorno de ejecución para el despliegue de la interfaz de usuario. Aquí el patrón de diseño cliente-servidor es utilizado tanto en la web, en redes de área local o aplicaciones centralizadas donde los módulos desarrollados implementen este patrón de diseño.

XForms es un *UIDL* el cual puede ser utilizado en entornos como los que se plantean en el segundo caso. A pesar que *XForms* nace del seno de la web, bajo la encomienda de sobrellevar muchas de las deficiencia que presentan las formas web, incluyendo la

dependencia de lenguajes *script*, los elementos que conforman su especificación pueden ser desacoplados y utilizarse de manera independiente con otras tecnologías (Rivera *et al*., 02), además *XForms* no está limitado a ser utilizado en navegadores (Boyer, 07). Por otro lado, debido al ambiente web de donde surge, se utiliza generalmente en conjunto con el lenguaje de etiquetas *XHTML* y el protocolo *HTTP*. Sin embargo, no son exclusivos, ya que *XForms* puede ser utilizado en conjunto con otro lenguaje *XML* y utilizar cualquier otro protocolo adecuado a los requerimientos de la aplicación (B'Far, 05).

Un error es encasillar a *XForms* como un lenguaje únicamente para definir formas web, ya que puede ser utilizado para desarrollar otro tipo de aplicaciones debido al conjunto de elementos que define (Cagle, 05), sin embargo las formas son un medio que proveen una interacción entre el usuario y la aplicación, más allá del enfoque tradicional en el cual las formas se utilizan para recolectar y enviar datos a un servidor. Las formas proveen un medio de interacción, y *XForms* enriquece esta interacción al incluir en su especificación mecanismos para definir una interfaz que reaccione tanto a interacciones del usuario como del sistema, además de aquellos mecanismos que permiten tener una interfaz que se modifica de manera dinámica tanto en estructura como en los datos. Por otro lado el uso de lenguajes web para desarrollar aplicaciones de escritorio, al ser incorporados en entornos de ejecución, se ilustra por algunas implementaciones de *XForms* existentes, tal es el caso de *Sidewinder*[4], el cual provee un *framework* que permite implementar páginas web como aplicaciones de escritorio al dotarlas de todas las características que una aplicación de escritorio ofrece.

Las premisas establecidas en los párrafos anteriores permiten visualizar cómo *XForms* puede ser explorado en otro tipo de aplicaciones más allá de aquellas para las que fue concebido, es decir, las formas web, así como trabajar en otro tipo de ambientes como la web. Por último, *XForms* es un estándar, lo cual trae la ventaja de su posible interoperabilidad con otras tecnologías, además de ser un lenguaje en el que se sigue trabajando aún.

---

[4] http://sw.swcube.com/

Independientemente del lenguaje seleccionado es necesario contar con algún tipo de «intérprete» propio del lenguaje. En el caso de *XForms* existen intérpretes para el lado del cliente y del lado del servidor.

III.3 XForms

XForms implementa parte de su especificación apoyándose en algunos estándares de la *W3C* existentes, para modelar, recolectar y transmitir el flujo de interacción. Entre los estándares se tienen:

XPath, es una tecnología utilizada para recorrer la estructura jerárquica de documentos *XML*, y definir expresiones relacionadas a los nodos del documento. *XForms* la utiliza para realizar el vínculo entre el modelo *XForms* y los controles de la forma, declarar propiedades al modelo y expresar dependencias entre elementos relacionados.

XML Events, es una tecnología que provee una abstracción de la especificación de «eventos *DOM2*» para implementar escuchadores y manejadores de eventos. *XForms* utiliza dichos escuchadores y manejadores para definir comportamientos específicos.

XML Namespaces y *XML Schema*. Las aplicaciones *XForms* utilizan *XML* 1.0 para la recolección de datos de la forma. *XML Namespaces* se utiliza para evitar conflictos entre campos de nombres similares que se utilizan para recolectar los datos. *XML Schema* se utiliza para definir restricciones a los datos y validar la instancia que los contiene.

Un «documento *XForms*» se compone de dos partes principales, el «modelo *XForms*» y los «controles de la forma». El diseño detrás de *XForms* es análogo al patrón de diseño denominado Modelo Vista-Controlador (MVC) (Dubinko, 03), en este patrón un «objeto modelo» contiene todos los datos esenciales, que es la información del sistema, uno o más «objetos vista» cuyo objetivo es presentar el modelo en un formato adecuado para interactuar, usualmente es la «interfaz de usuario», y un «objeto controlador» que está relacionado a la respuesta a eventos, normalmente acciones del usuario, y es el responsable de invocar cambios en el modelo y probablemente en la vista. En *XForms* los «controles de la forma» hacen la función del «objeto vista», aunque no existe nada que mapee directamente a un controlador en *XForms*, se pueden contemplar los mecanismos para el

procesamiento del modelo en conjunto con los «eventos *XForms*» como elementos que juegan un papel similar. *XForms* se utiliza dentro de algún otro lenguaje *XML*, denominado el lenguaje anfitrión, por ejemplo *XHTML*. Dicho lenguaje es utilizado para proveer la estructura en la cual son incorporados los «controles de la forma». Por otro lado se utilizan hojas de estilo para definir la presentación visual, en el caso de tratarse de una «interfaz gráfica de usuario».

III.3.1 Modelo XForms

El «modelo *XForms*» se compone de cuatro elementos:

- El elemento *model*: indica la raíz de la definición del «modelo *XForms*».
- El elemento *instance*: Sirve como un contenedor para los datos iniciales. El contenido de este elemento son simplemente datos que serán escritos y leídos durante la interacción con la forma, la cual se define en una estructura *XML*.
- El elemento *bind*: Establece condiciones que se aplican de manera continua a los datos en el elemento *instance*, éstas pueden ser restricciones o propiedades. De igual manera sirve para vincular un control de la forma con un elemento de la instancia de datos.
- El elemento *submission*: Define los parámetros para serializar y enviar los datos del elemento *instance*.

El modelo captura todos los aspectos no relacionados con la presentación.

III.3.2 Controles de la Forma

Los «controles de la forma» son los que definen la «interfaz de usuario», es por esta razón que también se les denomina bajo el mismo denominativo de «interfaz de usuario». En una página web, los controles pueden ser mapeados a instancias como botones, campos de texto, menús, etc. Estos controles están vinculados al «modelo *XForms*» para manipular los datos que contiene en su elemento *instance*.

Entre los controles de la forma que *XForms* provee, se pueden citar algunos, como es el caso de *input*, que en una página web permite la entrada de caracteres. El elemento *output*, se utiliza para presentar los datos del «modelo *XForms*». El elemento *trigger* es un

disparador de «acciones *XForms*», un botón es una de sus posibles presentaciones. El elemento *select1*, representa la selección de exactamente un elemento de una lista. Por otro lado el elemento *select*, se utiliza para la selección de ninguno, uno o muchos elementos.

Estos controles son interpretados para definir la «interfaz concreta funcional» a la plataforma objetivo. Dependiendo del dispositivo en cuestión y el soporte que brinde, deben traducirse a un elemento adecuado, en el caso del *input* pudiera manifestarse como un elemento para aceptar entradas por comandos de voz, por otro lado el elemento *output* pudiera manifestarse como una salida de audio en un dispositivo de sonido. La Figura 8 ilustra la estructura general de un «documento *XForms*», a modo general se muestra la jerarquía entre los elementos y la manera en la que el elemento *bind* relaciona los «controles de la forma» y la «instancia de datos» definida por el elemento *instance*.

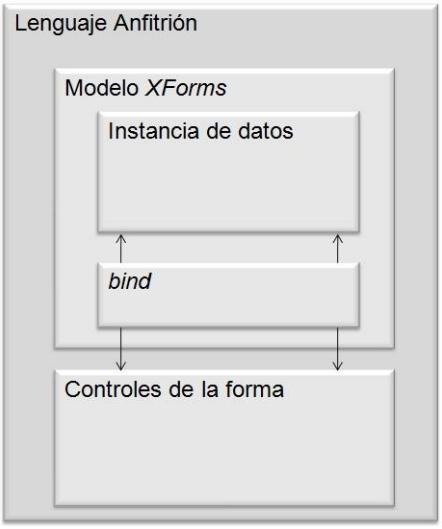

Figura 8. ***Composición general de un «documento XForms» y las relaciones entre los elementos.***

III.3.3 Eventos y Acciones XForms

Un «evento» puede verse como la representación de un acontecimiento asíncrono, por ejemplo, un «evento» en una «interfaz de usuario» sería hacer *click* con el *mouse* o empezar a introducir comandos de voz.

La parte de «*eventos*» y «*acciones*» que *XForms* provee, elimina el uso de lenguajes *script*. Ya que incorpora un conjunto de «manejadores de eventos» declarativos, que cubren la mayoría de los casos comunes que se realizan con los lenguajes *script*. Un «manejador de eventos» define la «acción» asociada cuando un «evento» ocurre. Por tal cada «evento» se

asocia con una «acción» y con un «elemento objetivo» que representa el punto donde se lleva a cabo la «acción» principal. Un «mecanismo de eventos» permite a una aplicación detectar «eventos» del usuario y del sistema.

Un «evento», en lo que a *XForms* concierne, es una estructura de datos que se pasa a los «escuchadores de eventos». Un «escuchador de eventos» u «observador de eventos» es el elemento designado para detectar la ocurrencia del «evento» para así invocar al «manejador de eventos» correspondiente.

Los «eventos *XForms*» se basan en la especificación de «Eventos *XML*» (*XML Events*, por su nombre en inglés). Dicha especificación brinda un marco de referencia para la definición de «eventos», ya que provee un mecanismo genérico extensible para crear «escuchadores de eventos *DOM2*» y asociarlos a «manejadores de eventos» en distintos vocabularios *XML* (*HTML*, *XHTML*, *SVG*, etc.).

El estándar de la *W3C*, *Document Object Model* (*DOM*, por sus siglas en ingles), permite definir un «modelo de objetos» para representar documentos *HTML*, *XML* y formatos relacionados de manera independiente de plataforma y lenguaje. Provee una representación estructural del documento, que permite a programas y *scripts* modificar el contenido, estructura y presentación visual (estilo). *DOM* está diseñado en tres niveles:

Nivel 1, se enfoca al modelo de los documentos, y especifica funcionalidades para la navegación y su manipulación.

Nivel 2, define un modelo para la hoja de estilos del documento, y funcionalidades para manipular la información de estilo. Permite además la navegación del documento, define un modelo de eventos y da soporte a *XML Namespaces*.

Nivel 3, se enfoca al modelo del contenido (como *DTD* y *XML Schema*) con soporte a la validación del documento.

Los «eventos» de *DOM* permiten el procesamiento y envío de «eventos» de una manera consistente. La interfase *EventListener* de «Eventos *DOM2*» define un mecanismo genérico para el manejo de «eventos» en navegadores *XML*. Dicha funcionalidad es implementada por «*XML Events*» a través de una sintaxis genérica para *XML*. De este modo los lenguajes basados en *XML* pueden definir una semántica personalizada de «eventos» sin la necesidad de realizar extensión alguna al lenguaje de etiquetas anfitrión. Es decir, la estructura dada

por la especificación de «*XML Events*» permite describir un «evento», el elemento objetivo y su manejador, de una manera consistente. Obviamente el navegador web donde se ejecute debe implementar las especificaciones mencionadas.

Los «manejadores de eventos» pueden ser escritos en un lenguaje *script*, ej. *javascript* o *Python*, en cuyo caso el navegador *XML* requiere acceso a un lenguaje intérprete apropiado para invocar al manejador; de manera alternativa, los manejadores estándar pueden ser prescritos como una etiqueta *XML* declarativa, como en el caso de *XForms*. En este caso dicha declarativa debe ser implementada por el intérprete.

El «modelo de eventos» definido en *DOM2* permite el manejo de «eventos genéricos»; de este modo los «eventos» generados por el sistema o por el usuario pueden ser manejados de una manera consistente. Por lo tal se utiliza el mismo mecanismo para reaccionar a ambos tipos de «eventos». «*XML Events*» permite el acceso a nivel de etiquetas a este mecanismo de procesamiento de «eventos», permitiendo así la creación de un comportamiento declarativo para responder a «eventos genéricos».

Crear un «escuchador de eventos» requiere especificar tres elementos de información: el «evento» a escuchar, el «elemento objetivo» y el manejador. Para tal, hay tres maneras de definirlo:

1. Definición independiente.
2. Adjuntar el «evento» y «manejador» al «elemento objetivo».
3. Adjuntar el «elemento objetivo» y «evento» al «manejador».

La Figura 9 muestra un ejemplo de cada uno de los casos expuestos, en el inciso 1), a través de la instrucción *listener* se declaran los tres elementos, aquí se espera por la activación de un botón para realizar una acción determinada. En el inciso 2), una liga es el «elemento objetivo» a la que se le adjuntan la declaración del «evento» y del «manejador», aquí se espera por la activación de dicha liga haciendo *click* con un *mouse* para realizar una acción determinada En el inciso 3), dentro de la definición de la «acción» (escrita en un lenguaje *script*), se define el «elemento objetivo» y el «evento» a escuchar, aquí se espera a que se solicite el envío de una forma al servidor para que sea validada. Como se ha mencionado para este último caso puede tratarse de una «acción» declarativa, como es el caso de *XForms*, lo cual se muestra en el inciso 4), aquí se define que cuando el «documento

XForms» termine de ser cargado en el navegador, se asocie al elemento de la «instancia de datos» definida como *mes* el valor de *enero*. La referencia a cualquiera de los elementos puede hacerse a través de un identificador único asignado y en el caso de los «datos de instancia» en *XForms* a través de referencias definidas con *XPath*.

Figura 9. ***Ejemplos de la definición de eventos.***

La Tabla II tomada de (Raman, 03**)** hace una correspondencia entre algunos «Eventos *XForms*» y «Acciones *XForms*».

Tabla II. Correspondencias entre un conjunto de «Eventos XForms» y «Acciones XForms».

Evento	Acción
xforms-delete	Delete
xforms-insert	Insert
xforms-rebuild	Rebuild
xforms-reset	Reset
xforms-revalidate	revalidate
xforms-submit	Send
xforms-setfocus	Setfocus

A pesar que «*XML Events*» está diseñado principalmente para ser utilizado en ambientes web, *XForms* adopta el modelo de «*eventos DOM2*» de la manera en que «*XML Events*» lo implementa.

La Figura 10 muestra el ejemplo de un «documento *XForms*» sencillo, en el cual se declara una «instancia de datos» conteniendo dos datos: el nombre y el apellido de una persona. Estos datos se asocian a dos «controles de la forma» de tipo *input* a través del elemento *bind*, que además permite definir restricciones a dichos datos. Cuando el documento se termina de cargar, a dichos controles se les asigna el valor contenido en los datos de instancia. Por otro lado se define un elemento *trigger* («elemento objetivo») el cual al ser

activado («evento» *DOMActivate*) invoca a la «acción XForms» *setvalue* («manejador de eventos»), cuya función consiste en asignar un valor específico al elemento de la instancia referenciado. Para el documento se utiliza *XHTML* como lenguaje anfitrión.

```
1  <?xml version="1.0" encoding="iso-8859-1"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
3  transitional.dtd">
4  <html xmlns="http://www.w3.org/1999/xhtml"
5    xmlns:xf="http://www.w3.org/2002/xforms"
6    xmlns:ev="http://www.w3.org/2001/xml-events"
7    xmlns:xs="http://www.w3.org/2001/XMLSchema">
8    <head>
9          <title>Ejemplo XForms</title>
10
11     <xf:model id="idModelo">
12
13       <xf:instance id="idInstancia" xmlns="">
14        <data>
15          <nombre>Gabriel</nombre>
16          <apellido>García</apellido>
17        </data>
18       </xf:instance>
19
20      <xf:bind id="idBind1" nodeset="instance('idInstancia')/nombre" readonly="true()"/>
21      <xf:bind id="idBind2" nodeset="instance('idInstancia')/apellido"/>
22
23     </xf:model>
24     </head>
25     <body>
26
27     <xf:input bind="idBind1">
28      <xf:label>Nombre: </xf:label>
29    </xf:input>
30     <br/>
31
32     <xf:input bind="idBind2">
33      <xf:label>Apellido(s): </xf:label>
34    </xf:input>
35    <br/>
36
37    <xf:trigger>
38      <xf:label>Activador</xf:label>
39      <xf:setvalue id="idAccion1" ev:event="DOMActivate" bind="idBind2">García Marquez</xf:setvalue>
40    </xf:trigger>
41
42     </body>
43 </html>
44
```

Definición de la instancia de datos

Definición del modelo XForms

Definición de elementos *bind* asociados a los datos de instancia

Control de la forma vinculado al elemento *nombre* a través de un elemento *bind*

Control de la forma vinculado al elemento *apellido* a través de un elemento *bind*

Definición de los controles de la forma

Control de la forma asociado a una acción vinculada al elemento *apellido* a través de un elemento *bind*

Figura 10. ***Ejemplo de un «documento XForms».***

La Figura 11 muestra el documento interpretado utilizando el navegador *Mozilla Firefox 2.0.0.14*, y el intérprete *Mozilla XForms 0.85* para una *PC* de escritorio. El inciso 1) muestra la interpretación del documento al momento de terminar de cargarse y el inciso 2) muestra el documento cuando el elemento *trigger* representado como un botón es activado a través del mouse de la *PC*, lo que origina que .se desencadene la acción que modifica el valor de la instancia de datos, que requiere actualizar el control de la forma con el nuevo valor.

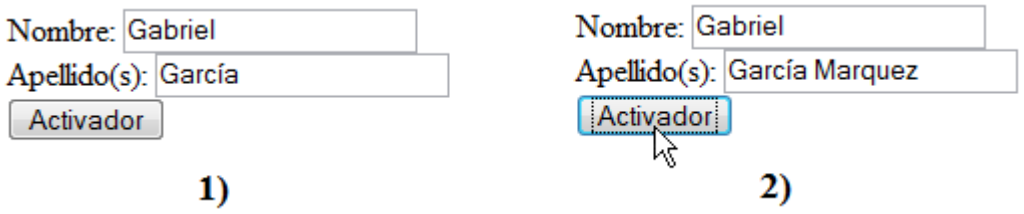

Figura 11. ***Interpretación del «documento XForms» para el navegador de PC Mozilla Firefox.***

III.3.4 *XForms* en el Área de Consciencia del Contexto

La literatura presenta algunos trabajos en los que *XForms* ha sido utilizado, para dar soporte a escenarios donde se involucra la «consciencia del contexto». En este apartado se comentan algunos de estos.

Braun *et al.* exploran la posibilidad de crear una interfaz de usuario distribuida entre un conjunto de dispositivos que forman un dispositivo virtual (Braun *et al.*, 04b). A esta asociación se le denomina *federación de dispositivos*. Un escenario de aplicación es aquel donde una persona tiene a su disposición una *PDA* y una pantalla pública, de aquí el usuario accede a una aplicación relacionada a un reproductor de canciones. La interfaz presentada se distribuye de la siguiente manera: en la pantalla pública se despliegan texto e imágenes relacionadas a la canción que se reproduce y en la *PDA* un conjunto de botones que actúan como un control para el reproductor. En (Braun *et al.*, 04a), se explora la idea de sesiones móviles entre dispositivos, pero cada dispositivo brinda ventajas y desventajas en base a sus recursos, de aquí se vislumbra la idea de la *federación de dispositivos*, la cual se apoya en extensiones a *XHTML* y *XForms*.

En (Barton *et al.*, 03) se propone la incorporación y soporte a un mayor número de formatos específicos por parte de *XForms*, con la idea de que un dispositivo capaz de sensar, publique en un ambiente inteligente, a través de una forma, los formatos que él genera y acepta, para así buscar comunicación con otro dispositivo en el ambiente, para solicitar un servicio particular. Por ejemplo, una cámara digital se puede considerar como un dispositivo que puede sensar su ambiente, a través de la captura de imágenes en un formato particular, de aquí que ésta busque comunicación con alguna impresora en el ambiente que entienda el formato de las imágenes generadas, para así poder realizar impresiones.

En el enfoque planteado por (Nathanail *et al.*, 07), *XForms* se utiliza como un elemento temporal dentro del proceso de generación de la interfaz de usuario en un lenguaje específico, dando así soporte al sistema multimodal y adaptativo propuesto.

Enfocado a la multi-modalidad en (Honkala *et al.*, 06) *XForms* se utiliza en conjunto con hojas de estilo que contienen la información de los elementos pertenecientes a cada modalidad a la que se le da soporte. Con esta estrategia se tiene una interfaz multi-modal.

III.4 Resumen y Conclusiones

El objetivo de un *UIDL* no está limitado, ya que puede servir como un elemento temporal dentro del proceso de generación de la interfaz, o como la entidad final a interpretarse en tiempo de ejecución. Ambos enfoques requerirán el «intérprete» necesario para llevarse a cabo. La idea de utilizar la «interfaz abstracta» como parte del proceso para generar la «interfaz concreta» en un lenguaje específico se utiliza en algunos entornos de desarrollo, e incluso es la motivación del diseño de *UIML*.

En el caso de *XForms*, se tienen «intérpretes» tanto para el lado del cliente como para el servidor. En esta propuesta el interés es el trabajo en el lado del cliente.

Ya sea que *XForms* se utilice para web o para aplicaciones de escritorio de la manera en la que se ha hecho mención, la parte que se relaciona a la manipulación de la información e interfaz de usuario se apegará al enfoque del patrón arquitectónico *MVC*, debido a la propia especificación del lenguaje.

Capítulo IV

Diseño e Implementación de los Componentes de Adaptación

IV.1 Introducción

Este capítulo presenta las consideraciones de diseño e implementación de los elementos de la estrategia de adaptación de la «interfaz de usuario» basada en *XForms*. Los requisitos para cada elemento se establecen a partir de la funcionalidad requerida y la relación que mantiene con los otros elementos. En primera instancia se describen brevemente algunos escenarios donde se requiere cierto tipo de adaptación en la «interfaz de usuario», estos escenarios dan la pauta para conocer el tipo de adaptación a la que se requiere dar soporte.

IV.1.1 Escenarios de Adaptación de la Interfaz de Usuario

El escenario principal se enfoca al apoyo en el hogar para la vida independiente de un adulto mayor que sufre una afectación cognoscitiva que le impide recordar actividades básicas como medicarse, asearse, etc.

Un sistema que apoye a la persona debe recordarle y llevar cuenta de las actividades de su rutina diaria básica, por ejemplo, en la mañana darle a conocer la lista de actividades matutinas y conforme transcurra el día darle a conocer las correspondientes.

Por otro lado, al momento de cocinar el sistema puede proveer asistencia para que la actividad se realice sin contratiempo, por ejemplo, definiendo la secuencia de preparación de los alimentos.

Otro tipo de asistencia se relaciona con el recordatorio de información de distintas índoles, por ejemplo, teléfonos de emergencia, número de seguros social, clave de acceso a la caja fuerte, o información financiera, la cual se puede consultar a través de alguna pantalla digital. Sin embargo, alguna de esta información se puede clasificar como privada. Si en un momento dado la persona se encuentra consultando este tipo de información y en el entorno se encuentran más personas, queda en riesgo dicha privacidad.

Otra cuestión es hacer disponible la información a la persona independientemente del lugar de la casa donde se encuentre, por lo tal es necesario que se pueda presentar en distintos dispositivos.

La ayuda prevista por un «sistema consciente del contexto» en este tipo de escenarios viene dada por la anticipación del sistema para apoyar a la persona en las actividades presentes y futuras, así como recordarle de las ya realizadas. Obviamente se supone la existencia de la tecnología necesaria para poder llevar a cabo los planteamientos mencionados.

Con relación a la disponibilidad de la información a través del dispositivo presente en el lugar en el que se encuentre la persona, hay que considerar que hay dispositivos más adecuados que otros para presentar cierto tipo de información, por ejemplo, recordarle a la persona sobre su medicación puede realizarse a través de una pantalla digital, lo que permite mostrarle información detallada , por otro lado, si el objetivo es hacerlo consciente de la actividad, sin necesidad de entregar tanta información, puede hacerse a través de algún medio que capte la atención de la persona y la haga consciente de dicha actividad. Por ejemplo la Figura 12 muestra un dispositivo que consta de *leds*, una bocina y un sensor, cuando es hora de la medicación, este dispositivo enciende sus luces y a través de mensajes grabados notifica de la dosis a ingerir. Además permite en cierta medida conocer si la persona toma sus medicinas ya que el sensor del dispositivo detecta si el frasco fue retirado de él y vuelto a colocar, lo que permite inferir con una posibilidad de error que la persona ha tomado su medicina.

Figura 12. ***Ejemplo de un dispositivo visual y auditivo que notifica a la persona sobre la medicación correspondiente***[5]**.**

El tipo de interfaz que aquí se presenta, da cuenta de dos tipos de información de salida y un tipo de información de entrada. Las salidas son: la notificación visual y la notificación auditiva. La entrada se da cuando se detecta que el frasco ha sido retirado del dispositivo. Esta misma «interfaz de usuario» presentada como una «interfaz de usuario de *software* » en una *PC* de escritorio podría mapear ambas salidas a una representación textual, la primera notificando sobre la actividad y la segunda sobre los detalles de dicha actividad. Por otro lado, la entrada podría mapearse a un control de la interfaz que permitiera indicar manualmente que dicha actividad ha sido llevada a cabo, por ejemplo un *checkbox*.

La adaptación de la «interfaz de usuario» que se plantea, recae sobre las capacidades de cada dispositivo, sin embargo, otro tipo de adaptación que puede sufrir la interfaz una vez que la actividad principal ha terminado (toma de medicinas), sería actualizar las salidas para realizar notificaciones de la próxima actividad. En esta adaptación se pueden contemplar dos enfoques generales para llevar a cabo las adaptaciones requeridas en cada situación:

[5] http://www.leechvideo.com/key/Context-aware-applications/

1. El primer enfoque contempla que un servidor genere una nueva interfaz adecuada y la envíe al cliente en cuestión.
2. El segundo enfoque contempla que en el cliente se lleve a cabo la adaptación de la «interfaz de usuario», para lo cual es necesario predefinir las adaptaciones que puede realizar.

El segundo enfoque contempla ciertas ventajas como la reducción del número y tamaño de llamadas al servidor, aunque es posible que existan adaptaciones que no puedan realizarse en el cliente ya sea por el procesamiento que involucran o la falta de mecanismos en el lado del cliente para ejecutarlas.

IV.1.2 Suposiciones y Diseño Preliminar

Como se menciona en el Capítulo II, en la especificación de la «interfaz abstracta» no se contempla información explicita sobre el «contexto», ya que se debe considerar durante el proceso de la generación de la «interfaz concreta». Por otro lado, el «contexto» debe ser administrado por otra instancia encargada de las tareas relacionadas como la recolección, interpretación, diseminación, etc. Partiendo de estas premisas, de la propuesta planteada, de la especificación *XForms* y de los escenarios descritos, se contemplan las siguientes suposiciones y consideraciones de diseño preliminar como preámbulo del diseño e implementación de la propuesta.

IV.1.2.1 Suposiciones

Para la definición de la estrategia de adaptación se parte de cuatro supuestos:

1) La aplicación se basa en un enfoque cliente - servidor.
2) Existe una arquitectura que da soporte para la detección y estimación del «contexto».
3) Existe una entidad que hace el papel de «servidor de contexto», encargada de las tareas afines a la administración del «contexto» como son la recolección, interpretación, inferencia, manejo de incertidumbres, diseminación de cambios, etc.
4) Se tiene un cliente con soporte a *XForms*.

IV.1.2.2 Diseño preliminar

1. El estado del «contexto» actual así como las notificaciones de los cambios en éste deben ser comunicados por el «servidor de contexto», de tal manera que los elementos de adaptación puedan utilizar dicha información sin necesidad de conocer cómo fue obtenida, de la misma manera en la que se plantea en la definición de un «sistema consciente del contexto» que se presenta en el Capítulo II. Bajo esta idea se contempla un enfoque basado en eventos. La definición de un «servidor de contexto» escapa del alcance de este trabajo, limitándose a la definición del modelo para representar el «contexto» actual y la información a comunicar dado un cambio en el «contexto».
2. En el servidor se debe generar un «documento *XForms*» que define la «interfaz abstracta», este documento debe ser enviado al cliente. Para efectos de prueba se utiliza como cliente el navegador *Mozilla Firefox 2.0.0.14* para *PC* al que se le agrega el intérprete *Mozilla XForms 0.85*. Por lo tal se utiliza *XHTML* como lenguaje anfitrión.
3. La adaptación de la «interfaz de usuario» se define en tres etapas:
    - La primera se lleva a cabo en el servidor, y contempla el proceso de generación de la «interfaz abstracta» al definir el contenido de ésta, es decir, el lenguaje anfitrión, y elementos *XForms* (modelo, controles, eventos y acciones).
    - En la segunda etapa, en el cliente, el «intérprete *XForms*» es responsable de traducir la «interfaz abstracta» a una manifestación concreta adecuada al cliente, la cual es la «interfaz concreta funcional».
    - La tercera etapa comprende las posibles adaptaciones predefinidas sobre la «interfaz concreta funcional», dichas adaptaciones están descritas a través de «acciones *XForms*» definidas en la «interfaz abstracta», las cuales debe implementar el «intérprete» en cuestión. Cada «acción *XForms*», se asocia a un «evento» personalizado y se ejecutan a partir de la notificación de dicho «evento» el cual representa un cambio en el «contexto» actual.
4. En relación a la «tercera etapa de adaptación», la «interfaz abstracta» generada debe contener en su descripción, no la información contextual explícita, sino las adaptaciones que pueda sufrir en relación a cambios en el «contexto», esto se hace a través de los mecanismos dados por *XForms*, es decir, las «acciones *XForms*» y la especificación

XML Events que permite la definición de eventos genéricos. Estas adaptaciones deben ser definidas durante la fase de diseño de la aplicación correspondiente y son las que la «interfaz concreta funcional» puede llevar a cabo. Por ejemplo, actualizar información presentada, cambiar entre distintas vistas de la información, agregar o eliminar elementos presentados, etc.

5. La generación de la «interfaz abstracta» en la «primera etapa de adaptación» debe hacerse de manera dinámica a partir de una especificación del «contexto» actual al que se denomina «**modelo del contexto**», contenido predefinido denominado «**modelo de elementos**» y «**reglas de adaptación**» que guíen las etapas de adaptación planteadas. La generación de la «interfaz abstracta» queda a cargo de una aplicación denominada «**compositor *XForms***».
6. El cliente debe ser capaz de conocer los cambios en el «contexto» a través de las notificaciones del «servidor del contexto». Esto no implica que el cliente debe ser capaz de sensar su «contexto», ya que toda la información debe ser entregada por el servidor. El enfoque planteado, donde el cliente es el encargado de llevar a cabo la traducción de la «interfaz abstracta» es cuestionado por (Nathanail *et al*., 07) quien indica que al enviarle el «documento *XForms*» al cliente se le da la carga de la conversión. Tal afirmación es correcta, ya que se requiere que en el cliente se realice la interpretación, sin embargo uno de los propósitos de este trabajo es explorar las características que *XForms* presenta como soporte a la adaptación de información en la «interfaz de usuario». Además, *XForms* realiza la mayor parte del trabajo en el lado del cliente, lo que reduce el número y tamaño de llamadas al servidor. Esta consideración es importante para dispositivos móviles, que tienden a trabajar en condiciones poco favorables de ancho de banda (XForms, 08). Por tal se propone el esquema de eventos donde el servidor notifique al cliente de los cambios en el «contexto».
7. La adaptación de la interfaz de usuario está en función del «contexto», por tal es necesario contar con una estructura común que permita la descripción del «contexto» de manera que pueda ser manipulado y comunicado, tanto por el «servidor de contexto» como por los elementos de la estrategia de adaptación. Además de esto, es necesario considerar que existen «parámetros contextuales» cuyo valor cambia en el tiempo, de

aquí que se requiere diferenciar entre estos parámetros «dinámicos» y aquellos «estáticos». Los «parámetros contextuales» hacen referencia a las entidades que conforman el «contexto», estos pueden ser, el tamaño de pantalla de un dispositivo, la ubicación del usuario, alguna preferencia del usuario, etc.

La Figura 13 hace un bosquejo de las tres fases de adaptación, los involucrados y el lugar donde se llevan a cabo. La nomenclatura utilizada se describe a continuación.

- ME, Modelo de Elementos
- MC, Modelo del Contexto
- RA, Reglas de Adaptación
- CXF, Compositor *XForms*
- IUA, Interfaz de Usuario Abstracta
- IXF, Intérprete *XForms*
- IUC, Interfaz de Usuario Concreta
- IUCM, Interfaz de Usuario Concreta Modificada

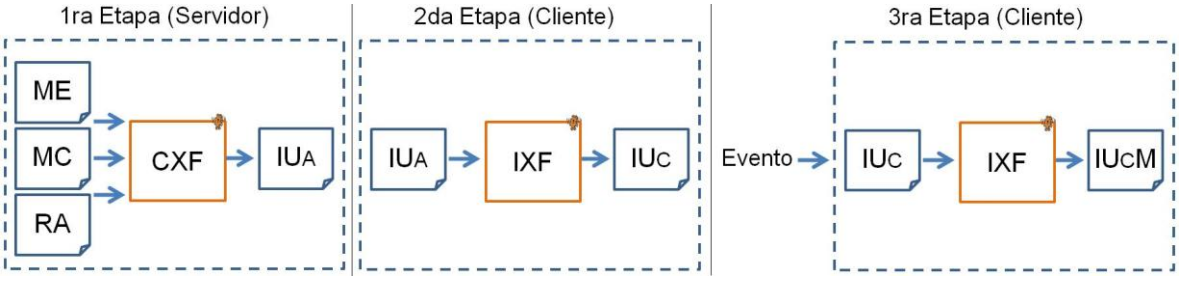

Figura 13. ***Etapas de adaptación por las que pasa la interfaz de usuario.***

La propuesta de este trabajo se centra en la implementación de la «primera etapa de adaptación», al definir el «modelo de elementos», el «modelo de contexto», las «reglas de adaptación» y la aplicación «compositor *XForms*». De este modo se contempla la generación de una «interfaz abstracta» que al estar descrita en *XForms* pueda ser procesada por clientes con un «interprete *XForms*» correspondiente. Como se ha mencionado para efectos de prueba se utiliza *XHTML* como lenguaje anfitrión.

La «segunda etapa de adaptación» se deja a un «intérprete *XForms*» existente, así como la 3ra etapa donde un «servidor de contexto» esté notificando al cliente sobre cambios en el «contexto». Cada evento notificado debe estar asociado a una «accion *XForms*» en la

«interfaz abstracta», de modo que el «intérprete» ejecute dicha acción sobre la «interfaz concreta funcional».

La forma de comunicar la aplicación «compositor *XForms*» con el servidor en cuestión, la comunicación entre el cliente y servidor, así como la interacción entre el cliente y el «intérprete» para notificar un «evento» se dejan a cargo de la arquitectura cliente-servidor anfitriona. En el Capítulo V se describe la implementación de una arquitectura cliente-servidor, utilizando *servlets* del lado del servidor interactuando con el Navegador *Mozilla Firefox* y el intérprete *Mozilla XForms* en el cliente.

IV.1.2.3 Diseño General

En este apartado se da una visión general del diseño de la estrategia de adaptación trabajando dentro de una arquitectura que da soporte a la «consciencia del contexto». En primer lugar se contempla la definición de tres modelos y una aplicación encargada de generar el «documento *XForms*». Se supone la existencia de un servidor, un cliente con soporte para *XForms* y un «servidor de contexto».

El «**modelo de elementos**» define el contenido que se puede incorporar a la interfaz a generar, estos pueden ser campos de texto, imágenes, etc. Cada posible elemento a incorporar se caracteriza por un conjunto de propiedades que permiten al «compositor *XForms*» definir la estructura adecuada para incorporarlo dentro del «documento *XForms*». Dentro de estas propiedades se encuentran las que definen si el elemento es sensible al «contexto» lo que permite saber qué «acción *XForms*», adecuada a la adaptación requerida, incorporar.

El «**modelo del contexto**» define el estado del «contexto» en un momento dado, especificando los «parámetros contextuales» y sus valores actuales, además de definir si es un parámetro «dinámico» o «estático». El «servidor de contexto» debe generar este documento así como monitorear aquellos parámetros que se definan «dinámicos». Al no contemplar la propuesta la definición de un «servidor de contexto», se limita a la definición del modelo para representar el «contexto» y la estructura de información para comunicar los cambios en el «contexto».

Las «**reglas de adaptación**», relacionan los dos modelos anteriores, en primer lugar deben apoyar en la «primera etapa de adaptación», y segundo deben de indicar bajo qué valor específico de un «parámetro contextual» una entidad del «modelo de elementos» definida como sensible al «contexto» lleva a cabo su adaptación.

El «**compositor *XForms***» es el encargado de generar el «documento *XForms*» a partir de los modelos anteriores y las reglas de adaptación en la «primera etapa de adaptación». En dicha etapa para cada entidad del «modelo de elementos» definida como sensible al contexto se genera un registro que contiene un «evento» único relacionado a un valor específico de un «parámetro contextual» que se especifica en las «reglas de adaptación», dicho «evento» se asocia a una «acción *XForms*» definida en la «interfaz abstracta». El conjunto de registros generado debe ser pasado al «servidor de contexto», y durante el tiempo de ejecución, cuando el valor actual de un «parámetro contextual» monitoreado corresponda con el valor definido en los registros, se debe notificar el «evento» correspondiente al cliente.

Después de plantear a grandes rasgos los modelos y la aplicación propuestos para la estrategia de adaptación se describe una secuencia genérica que involucra la propuesta plateada.

El flujo comienza cuando (*i*) un cliente capaz de soportar *XForms* realiza la solicitud de un servicio al servidor, (*ii*) el servidor solicita al «compositor *XForms*» generar la «interfaz abstracta» para el servicio en cuestión. (*iii*) El compositor utiliza el «modelo de elementos» relacionado al servicio, el «modelo de contexto» generado en ese momento por el «servidor de contexto» y las «reglas de adaptación» para generar la «interfaz abstracta». (*iv*) Una vez generada la «interfaz abstracta» se envía al cliente, por otro lado, se entrega al «servidor de contexto» los registros de «eventos» a notificar. (*v*) En el cliente, el «intérprete» se encarga de traducir la «interfaz abstracta» a una «interfaz concreta funcional». (*vi*) Una vez realizada la traducción a una «interfaz concreta funcional», las subsecuentes adaptaciones en el cliente, relacionadas a la «tercera etapa de adaptación», se da en un proceso de tres pasos: (1) reconocimiento de un cambio en el «contexto», (2) recuperación del «evento» relacionado al cambio y notificación al cliente y (3) ejecución de la acción correspondiente al «evento» en el cliente por el «intérprete *XForms*».

IV.2 Diseño de los Componentes de Adaptación

Para el proceso de generación de la interfaz se definen cuatro componentes principales, el «**modelo de contexto»**, el «**modelo de elementos»**, las «**reglas de adaptación»**, y el «**compositor *XForms*»**. Tanto los modelos como las reglas de adaptación se encuentran definidos en documentos separados, los cuales son utilizados por el compositor durante la generación de la «interfaz abstracta» en *XForms*.

IV.2.1 Modelo del Contexto

En base a la definición de «contexto» adoptada en el Capítulo II, así como las consideraciones pertinentes a la restricción del rango de «parámetros contextuales» a considerar y la categorización definida para estos se requiere lo siguiente:

1. El «modelo de contexto» debe reflejar únicamente los «parámetros contextuales» relevantes a la aplicación y al usuario así como los posibles valores de estos.
2. Dado el supuesto de la existencia de un «servidor de contexto» se contempla que la información que entrega tiene una notación legible para representar el «contexto». Por ejemplo, un sistema para calcular la ubicación de la persona dentro de una casa entregará un valor relacionado a un lugar concreto en vez de las posibles coordenadas utilizadas para el cálculo. Esta decisión se basa en que la administración de la información del «contexto» escapa del alcance de la propuesta.
3. Es necesario establecer una diferencia entre los «parámetros contextuales» cuyo valor permanece estático y aquellos cuyo valor puede variar en el tiempo, Para esto se hace una categorización entre parámetros «estáticos» y «dinámicos».
4. La definición de un parámetro como «estático» o «dinámico» dependerá de los requisitos de la aplicación, por ejemplo, la ubicación de una persona tomada como un «parámetro contextual», la cual se infiere a partir de la ubicación del dispositivo para acceder a un servicio, puede definirse como dinámica si utiliza un dispositivo portátil (p. ej. una *PDA*) o estática si utiliza un dispositivo fijo (p. ej. una pantalla pública).

5. La granularidad con la que se describa el «contexto» así como los tipos de relaciones definidas dependerán de igual manera de los requisitos de la aplicación.
6. El «servidor de contexto» debe ser el encargado de generar el documento que contiene el «modelo del contexto». Este documento debe servir únicamente como un repositorio inicial del estado del «contexto», es decir, los valores actuales de los «parámetros contextuales» en un momento dado, además de indicar cuáles son «dinámicos» para así llevar a cabo un monitoreo de estos.
7. El papel del «modelo del contexto» es proveer un marco de referencia para poder capturar la información del «contexto» en un momento dado y comunicarlo a todos los interesados a través de una estructura común.
8. Para facilitar el manejo del «contexto» se hace una división de éste en tres categorías: *contexto de cliente*, *contexto sensado* y *contexto conocido*.
    - *Contexto de cliente*: aquí se contemplan la especificación del dispositivo actual, así como las preferencias y perfil del usuario. Generalmente se trata de información estática.
    - *Contexto sensado o del entorno*: es aquel en el que fuentes externas proveen la información como lo sería un dispositivo de sensado. Ejemplos de este tipo de información es la ubicación física, condiciones ambientales como luz, temperatura, etc., otra información puede ser la actividad o tarea realizada por la persona o la compañía actual. Generalmente es información que varía en el tiempo.
    - *Contexto conocido*: se define como la información de la que es consciente el «servidor de contexto» en cuestión, es decir, información de la que no depende de unidades externas. Información de este tipo puede ser la hora, fecha, ancho de banda disponible, etc.

IV.2.2 Modelo de Elementos

La propuesta contempla la entrega de un «documento *XForms*» al cliente, por lo tal es necesario conocer la posible información que lo compondrá.

Generalmente el enfoque seguido cuando se utiliza un *UIDL* es que a partir de la «interfaz abstracta», se genera la «interfaz concreta». En el caso de la propuesta es necesario generar el «documento *XForms*» que contempla la «interfaz abstracta». Ya que se desean utilizar los mecanismos nativos propuestos en la especificación *XForms* para definir la «interfaz abstracta» y las posibles adaptaciones que se requieran en la «interfaz concreta funcional». La conversión a la «interfaz concreta funcional» y su manejo se dejan al «intérprete» en cuestión.

La idea aquí es definir un modelo que permita generar el «documento *XForms*». El cometido del «modelo de elementos» es justamente dar cierto nivel de abstracción de la especificación *XForms* de manera que capture la información necesaria a fin de que el «compositor *XForms*» sea capaz de generar a partir de él un «documento *XForms*». Este «documento *XForms*» define la «interfaz abstracta», la cual contiene el «modelo *XForms*», los controles de la forma y la declaración de «acciones» y «eventos» relacionados con la adaptación dinámica de la «interfaz concreta funcional».

El modelo contempla tres características básicas por cada elemento a definir:

1. El tipo de control de la forma.
2. La instancia del «modelo *XForms*» a la que se vincula, si es el caso.
3. El tipo de adaptación que sufre en la «interfaz concreta funcional», la cual se traduce a una «acción *XForms*».

Cada «modelo de elementos» se asocia con un archivo *XML*, que contiene la instancia de datos a referenciar en el «modelo *XForms*». Ambos se predefinen y son específicos para un servicio o aplicación en particular.

Todas las decisiones de interpretación del «modelo de elementos» se encuentran plasmadas en el «compositor *XForms*», las propiedades que dicta el «modelo de elementos» permiten definir los elementos *XForms* que constituyen la «interfaz abstracta».

IV.2.3 Reglas de Adaptación

El cometido de las «reglas de adaptación» es relacionar los elementos del «modelo del contexto» con el «modelo de elementos», para tal se pueden distinguir dos tipos de reglas:

1. Las primeras, se utilizan en la «primera etapa de adaptación», definen la inclusión o no de un elemento a la «interfaz abstracta». Dicha decisión se puede determinar por las capacidades del dispositivo, el entorno del usuario, el tipo de elemento a incluir, o la prioridad asignada. El enfoque de este tipo de regla va mas allá de definir una nueva estrategia para la «adaptación de contenidos», en relación a tareas afines a dicha área, únicamente pretende ejemplificar cómo asociar un «parámetro contextual» con una entidad del «modelo de elementos». Por lo mismo escapa también de toda tarea relacionada con la transcodificación de recursos o medios.
2. El segundo tipo de reglas define bajo qué valor de un «parámetro contextual» se dispara una acción que se declara en la «interfaz abstracta», dicha acción se vincula a un evento único el cual debe ser notificado por el «servidor de contexto». Esto es, si el elemento se definió como sensible al contexto.

Las reglas no capturan información sobre cómo definir un elemento *XForms* o la estructura de las acciones, ya que esto se contempla en el «modelo de elementos» y el «compositor *XForms*».

IV.2.4 Compositor XForms

Su principal función es generar el «documento *XForms*» a partir de las propiedades de las instancias del «modelo de elementos», estas propiedades representan indicios para saber el tipo de elemento *XForms* a incorporar. De este modo se contribuye al nivel de abstracción para *XForms* contemplado por el «modelo de elementos». Además estas mismas propiedades se utilizan para decidir la posible inclusión o no del elemento en cuestión.

Durante la construcción del «documento *XForms*», el compositor genera un conjunto de registros vinculando un «evento» único con un valor de un «parámetro contextual». Cada uno de estos «eventos» se asocia a una «acción *XForms*» única en la «interfaz abstracta». Este conjunto es pasado al «servidor de contexto» el cual es responsable de realizar las notificaciones.

La Tabla III muestra la relación lógica definida para vincular los cambios en el «contexto» y las adaptaciones a llevarse en la «interfaz de usuario».

Tabla III. Relación lógica para vincular información del «contexto» y las adaptaciones de la «interfaz de usuario».

Servidor de Contexto	**Interfaz Abstracta**
Valor Parámetro Contextual - Evento*i*	Evento*i* - Accion*i*

IV.3 Implementación de los elementos de adaptación

Bajo el enfoque propuesto, en el cual el servidor realiza una notificación al cliente para desatar una adaptación en la «interfaz de usuario», toda la tarea de monitoreo del «contexto» y la decisión de cuándo debe llevarse a cabo una adaptación, se deja totalmente al servidor.

Una cuestión a observar es que independientemente de todo el proceso que realice un «servidor de contexto» (interpretación, inferencias, manejo de incertidumbre, reconocimiento de comportamientos, correspondencias, etc.) para saber que es necesario adaptar el comportamiento de un servicio, al final todo se reduce a una sola condición. Esta idea es la que conduce la «estrategia de adaptación», ya que en el cliente únicamente se esperará por la notificación de un «evento» el cual indicará la ejecución de una «acción».

Cada posible adaptación que pueda sufrir la interfaz se asocia a una «acción *XFoms*», por tal las adaptaciones contempladas están sujetas a la especificación de la «acción» correspondiente.

IV.3.1 Modelo del Contexto

Para la descripción del «contexto», se utiliza el estándar *Composite Capabilities / Preference Profiles* (CC/PP) (Klyne *et al*., 04). Un «perfil *CC/PP*», describe las características de un dispositivo (p. ej tamaño/resolución de pantalla, tamaño de memoria, etc.) y las preferencias de usuario (p. ej. lenguaje, tipo de navegación, reproducción de sonidos activada, etc.). Se puede concebir como una estructura jerárquica de dos niveles, ya que se conforma de «componentes» y estos a su vez de «atributos. En el enfoque tradicional un «perfil» está vinculado a una *URI* o *URL*. Cuando un dispositivo realiza una solicitud al servidor, se envía dentro del encabezado de dicha solicitud la dirección de la *URI*/*URL* relacionada a su «perfil». De este modo el servidor puede consultarlo y conocer el «contexto de entrega» adecuado al cual adaptar el servicio o aplicación. Otro enfoque

contempla al cliente como la entidad que envía la información utilizando el formato definido por *CC/PP*.

CC/PP es un lenguaje extensible a partir de la definición de nuevos «componentes» y sus «atributos», la definición de estos se conoce como «vocabulario». *CC/PP* está basado en el *Resource Description Framework* (*RDF*) (Beckett, 04), un *framework* basado en *XML* para la descripción de meta-datos. Una de las características que se incluye de RDF es su capacidad de permitir definir relaciones del tipo:

[Recurso] ─ Propiedad → "Valor de Propiedad", donde la interpretación revela el hecho de que la *Propiedad* del *Recurso* posee un determinado *Valor*. Por ejemplo, si se tiene:

[Libro] ─ Autor → "Gabriel G. Márquez"

El ejemplo anterior debe leerse como "El *autor* del *libro* es *Gabriel G. Márquez*". A partir de este tipo de relación, *CC/PP* se ayuda para definir «componentes» y «atributos».

Los «componentes» específicos y «atributos» de un «perfil» no están definidos en la especificación *CC/PP*. La definición de un «vocabulario» específico se lleva a cabo con otros lenguajes estándar. Por tal los «vocabularios» deberían ser definidos usando *RDF Schema* (Brickley *et al*., 04), de este modo es posible asegurar la extensión de *CC/PP*.

A pesar que *CC/PP* está diseñado para describir información acerca del *hardware* y *software* de un dispositivo, puede describir también una amplia variedad de información contextual, tanta como la que pueda ser descrita en términos de «componentes» y «atributos» *CC/PP* (Indulska *et al.*, 03).

Su jerarquía de dos niveles es una desventaja, ya que la estructura de un dispositivo no puede ser descrita estrictamente en dos niveles (Held *et. al*., 02). Para superar la limitación de la jerarquía de dos niveles, en (Indulska *et al.*, 03) se propone que algunos «componentes» puedan poseer atributos de tipo «componentes». Dicho trabajo sirve de base para (Viterbo *et al.*, 06), al retomar las ideas propuestas y proponer que algunos «atributos» pueden presentar un comportamiento «dinámico» al modificar su valor en el tiempo, por tal propone, en base a los objetivos de su trabajo, la definición de cuatro propiedades para identificar este tipo de «atributos»: valor inicial, fuente, objetivo y

protocolo, dando así una descripción completa de cómo puede obtenerse la información de los valores para el «atributo» en cuestión.

La utilidad de la propuesta anterior reside en que el «perfil *CC/PP*» sirve para comunicar el estado del «contexto actual» en un momento dado, definiendo cuáles valores cambiarán en el tiempo y cuáles no, además de incluir información relacionada, por ejemplo la fuente de dónde se obtienen las actualizaciones de dichos valores (p. ej. el identificador del nodo sensor y el protocolo de comunicación).

Los trabajos anteriores permiten ver cómo *CC/PP* puede ser extendido de modo que pueda ser utilizado más allá del enfoque tradicional para el que fue concebido, por ejemplo su uso dentro de un «sistema consciente del contexto».

A modo de ejemplo, para el «perfil» genérico de un dispositivo de cómputo, se pueden definir como «componentes», la plataforma de *hardware* y la plataforma de *software*, «atributos» relacionados a la plataforma de *hardware* pueden ser el tipo de dispositivo, modelo del dispositivo o tamaño de pantalla, por otro lado para la plataforma de *software* se pueden contemplar, el sistema operativo, la versión del sistema operativo, etc., Esta decisión de composición de la información constituye el «*vocabulario*». El «perfil *CC/PP*» incluye los «componentes» específicos en conjunto con sus «atributos», donde cada «atributo» se le especifica un valor que dé información del dispositivo en cuestión. Para mostrar de manera gráfica el concepto de un «perfil *CC/PP*» se adopta la notación propuesta por (Viterbo *et al.*, 06), para describir la estructura planteada del dispositivo de cómputo, como lo podría ser un *Smartphone*, véase la Figura 14.

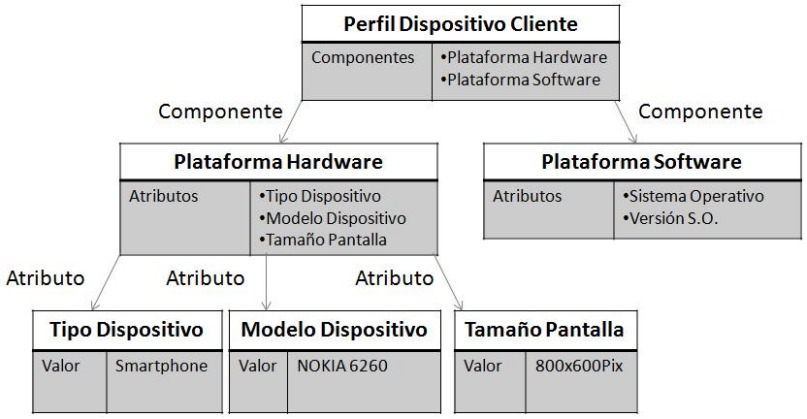

Figura 14. ***Descripción gráfica del «perfil» de un dispositivo de cómputo. El ejemplo muestra la descripción de un Smartphone.***

El «modelo del contexto» está conformado por tres «perfiles *CC/PP*», uno para cada una de las clasificaciones hechas del «contexto»: cliente, entorno y contexto conocido. Para la implementación se define un archivo esquema en *RDF Schema* que rige la composición de un archivo *RDF* que contiene el «perfil *CC/PP*». El archivo esquema define los elementos a incluir dentro del «perfil» y la relación entre «componentes» y «atributos».

Debido a que cada «modelo del contexto» es particular a una aplicación, en base a los «parámetros contextuales» y la granularidad establecida por los requisitos, es necesario definir un «vocabulario» especifico para cada aplicación. Lo que *CC/PP* brinda es un marco común para definir dicho «vocabulario», es decir «componentes» y «atributos», relacionados a los «parámetros contextuales» deseados. Para la cuestión relacionada con la clasificación de los «parámetros contextuales» en «dinámicos» y «estáticos», a cada «atributo» que sea «dinámico» se le definen tres de los cuatro tipos de información propuestos en (Viterbo *et al.*, 06), fuente, valor inicial y protocolo. Esto se implementa definiendo una clase denominada *Dinámico* en el documento esquema la cual contiene tres «atributos» relacionados a la *fuente*, *valor* y *protocolo*. Ya en el perfil, a los «atributos» que son dinámicos se definen del tipo *Dinámico* y de tipo descriptores utilizando el atributo `rdf:parseType="Resource"`, de manera que adoptan los tres «atributos» mencionados, lo que permite conocer su naturaleza dinámica e información relevante.

Para ejemplificar, se presenta el archivo esquema y el «perfil» generado para un *Smartphone*, que desea dar a conocer el modelo de batería que utiliza (estático) y la carga de batería que le resta (dinámico). Para esto se define un solo «componente» con dos «atributos» relacionados. La Figura 15 muestra el esquema *DRF Schema*. Se realiza este ejemplo debido a que no hay un esquema y «perfil» generales, si no que son específicos a cada aplicación.

```
1  <?xml version="1.0"?>
2  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-
3  schema#">
4    <!-- P e r f i l -->
5    <rdf:Description rdf:ID="SmartPhoneCliente">
6      <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
7      <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
8      <rdfs:comment>Clase que incluye los diferentes componentes para el Perfil</rdfs:comment>
9    </rdf:Description>
10   <!-- C o m p o n e n t e s  d e l  P e r f i l -->
11   <rdf:Description rdf:ID="Bateria">
12     <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
13     <rdfs:subClassOf rdf:resource="#SmartPhoneCliente"/>
14     <rdfs:comment>Describe la Bateria del Dispositivo</rdfs:comment>
15   </rdf:Description>
16   <!-- A t r i b u t o s  d e l  C o m p o n e n t e  B a t e r i a -->
17   <rdf:Description rdf:ID="Modelo">
18     <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
19     <rdfs:domain rdf:resource="#Bateria"/>
20     <rdfs:comment>Modelo de la Bateria</rdfs:comment>
21   </rdf:Description>
22   <rdf:Description rdf:ID="CargaActual">
23     <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
24     <rdf:type rdf:resource="#Dinamico"/>
25     <rdfs:domain rdf:resource="#Bateria"/>
26     <rdfs:comment>Carga actual restante de la Bateria</rdfs:comment>
27   </rdf:Description>
28   <!-- C l a s e  p a r a  A t r i b u t o s  D i n á m i c o s -->
29   <rdf:Description rdf:ID="Dinamico">
30     <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
31     <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
32     <rdfs:comment>Clase para definir atributos dinámicos</rdfs:comment>
33   </rdf:Description>
34   <rdf:Description rdf:ID="fuente">
35     <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
36     <rdfs:domain rdf:resource="#Dinamico"/>
37     <rdfs:comment>Describe la fuente de información de los datos</rdfs:comment>
38   </rdf:Description>
39   <rdf:Description rdf:ID="valor">
40     <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
41     <rdfs:domain rdf:resource="#Dinamico"/>
42     <rdfs:comment>Describe el valor actual del atributo</rdfs:comment>
43   </rdf:Description>
44   <rdf:Description rdf:ID="protocolo">
45     <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
46     <rdfs:domain rdf:resource="#Dinamico"/>
47     <rdfs:comment>Describe al protocolo de comunicación con la fuente</rdfs:comment>
48   </rdf:Description>
49 </rdf:RDF>
```

Declaración de la Clase principal

Declaración del componente Bateria

Declaración del «atributo» Modelo

Declaración del «atributo» CargaActual, se define de tipo Dinámico

Declaración de la Clase Dinamico con sus tres propiedades: fuente, valor y protocolo

Figura 15. ***Archivo esquema en RDF Schema, para crear un perfil de un Smartphone que desea dar a conocer datos sobre su Batería.***

La Figura 16 muestra el archivo *RDF* relacionado con el «perfil *CC/PP*», el cual contiene los valores del dispositivo. La Figura 17 muestra el perfil gráficamente.

```
1  <?xml version="1.0"?>
2  <rdf:RDF xmlns:ccpp="http://www.w3.org/2000/07/04-ccpp#"
3  xmlns:esq="file:///C:/.../EsquemaSmartPhone.rdfs#"
4  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
5    <rdf:Description rdf:ID="SmartPhone">
6      <ccpp:component>
7        <rdf:Description rdf:ID="Bateria">
8          <rdf:type rdf:resource="file:///C:/.../EsquemaSmartPhone.rdfs#Bateria"/>
9          <esq:Modelo>Mod-BL-C5</esq:Modelo>
10         <esq:CargaActual rdf:parseType="Resource">
11           <rdf:type rdf:resource="file:///C:/.../EsquemaSmartPhone.rdfs#Dinamico"/>
12           <esq:fuente>Interno</esq:fuente>
13           <esq:valor>85%</esq:valor>
14           <esq:protocolo>N/A</esq:protocolo>
15         </esq:CargaActual>
16       </rdf:Description>
17     </ccpp:component>
18   </rdf:Description>
19 </rdf:RDF>
```

Definición del Perfil

Definición del «atributo» estático Modelo

Definición del «atributo» dinámico CargaActual

Definición del «componente» Bateria

Figura 16. ***«Perfil» CC/PP de las características del Smartphone.***

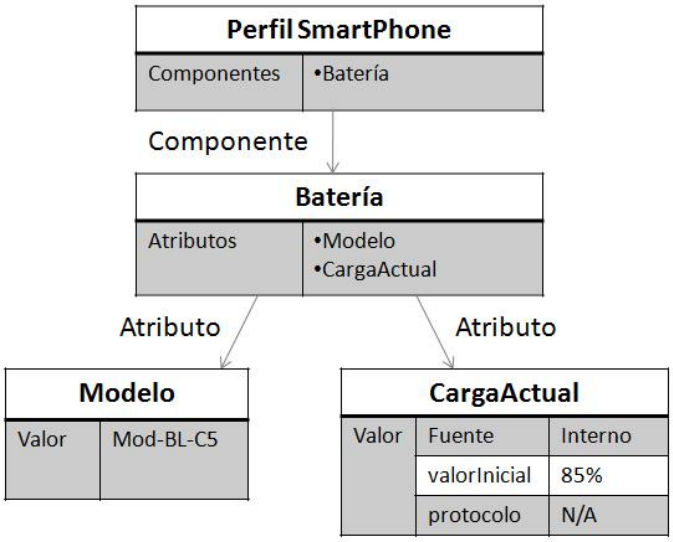

Figura 17. ***«Perfil» gráfico de las características del Smartphone.***

Todas las consideraciones hechas sobre el manejo de atributos especiales y formas de relacionar componentes que no están contempladas dentro de la especificación *CC/PP* deben ser implementadas por el «intérprete de perfiles» correspondiente.

IV.3.2 Modelo de Elementos

Como se ha mencionado la idea detrás de este elemento es definir un modelo que permita generar el «documento *XForms*», que capture la información necesaria a fin de que el «compositor XForms» sea capaz de generarlo.

La idea detrás del desarrollo de este modelo se basa en el artículo presentado en (McCreary, 07), donde la idea es automatizar el desarrollo de formularios web en *XForms*. Para tal efecto se basa en una notación estandarizada de los elementos que contendrá. En su propuesta, se parte de dos documentos, uno contiene una estructura *XML* que define los campos que contendrá el formulario y el otro es el documento *XML Esquema* que rige la composición del primero. El primero define elementos como `NombrePersona`, y el segundo define su tipo, el número de veces que se puede repetir en el formulario, etc. Para la generación del formulario se utiliza la sintaxis del elemento, por ejemplo en `NombrePersona` el prefijo `Nombre` indica la declaración de un «campo de texto», restringido por las propiedades dadas en el «documento esquema». Dicha propuesta contempla únicamente formularios para captura de información. En el caso de la estrategia

de adaptación es necesario definir controles, eventos, acciones y los elementos necesarios que trabajan en conjunto con ellos.

Partiendo de un concepto similar se propone que el «modelo de elementos» describa las «entidades de información» a presentar en la «interfaz de usuario» de manera predefinida, a través de características relacionadas a su presentación y el tipo de adaptación que debe realizar en respuesta a cambios en el «contexto». En concreto, todas las características contenidas en dicho modelo son analizadas sintácticamente y mapeadas a «elementos» o «acciones» de *XForms*. Como se ha hecho, se referirá a «entidades» o «entidades de información» a las instancias que conforman el «modelo de elementos», éstas pueden representar salidas como texto o imágenes, o entradas como campos de texto o listas de selección, etc.

Dado el propósito del «modelo de elementos» es necesario realizar una revisión a la forma de declarar los elementos que conforman un «documento *XForms*», y así tomar las decisiones que permitan definir un marco común que capture la información necesaria para generar el «documento *XForms*». Por ejemplo, la forma de vincular un «control de la forma» con un elemento en los «datos de instancia» puede hacerse de más de una manera: a través del elemento `bind`, o haciendo referencia al dato desde el control. El primer caso toma prioridad sobre el segundo cuando es necesario definir restricciones al dato, de otra forma el efecto de ambos caminos para hacer la vinculación es el mismo. Aunque hay que considerar que el elemento `bind` permite visualizar de manera más clara la separación entre datos y presentación.

IV.3.2.1 Representación de los «controles de la forma»

En primer lugar, en relación a los «controles de la forma», cada uno requiere un conjunto de parámetros de información para ser definido. En este momento el «modelo de elementos» se enfoca a seis controles específicos: `input`, `output`, `select`, `select1`, `secret` y `textarea`. Esta decisión se basa en el hecho que *XForms* puede ser utilizado más que para formas web, por lo tal el cometido inicial de *XForms* en este trabajo sigue la misma idea. Los elementos que no se contemplan son: el elemento `upload`, cuya función es

permitir especificar un archivo para enviar a un servidor y el elemento `range`, que es un control que permite especificar un valor dentro de un rango limitado.

La referencia a un dato de instancia utilizando *XPath* puede hacerse a través de cuatro atributos:

- `ref`, si se requiere referenciar a un nodo particular. En caso que exista un conjunto de nodos bajo la misma etiqueta, se aplica la regla de primer nodo.
- `nodeset`, si el objetivo del vinculo es seleccionar un conjunto de nodos de cualquier tamaño.
- `model`, en caso de tratarse de un documento complejo, podrían presentarse más de un «modelo *XForms*», por lo que este atributo se utiliza para referenciar al modelo deseado a través de un identificador (IDREF).
- `bind`, este atributo hace referencia al identificador de un elemento `bind` asociado a un dato de instancia. Este atributo toma precedencia sobre cualquiera de los otros tres, si es que aparecen como atributos también.

A modo de ejemplo la Figura 18 muestra una «instancia de datos» sencilla y las posibles formas de hacer referencia al nodo `nombre` de dicha instancia utilizando el atributo `ref`.

```
1 <xf:instance id="idInstancia" xmlns="">
2  <data>
3    <nombre>Gabriel</nombre>
4    <apellido>García</apellido>
5  </data>
6 </xf:instance>
```

```
ref=//nombre
ref=/data/nombre
ref=instance('idInstancia')/nombre
```

Figura 18. ***Tres formas de referenciar al nodo nombre utilizando el atributo ref.***

La Tabla IV hace una síntesis de la declaración de los seis controles contemplados.

Tabla IV. Síntesis de la declaración de controles XForms.

Control	Parámetros	Referencia	Elemento referencia
`input`		`ref`=exp. *XPath* | `bind`=IDREF | `nodeset`=exp. *XPath*	Dato de instancia

	Ejemplo:
	`<xf:input ref="instance('idInstancia')/nodo"/>`
	ó
	`<xf:input ref="instance('idInstancia')/nodo">`
	`<xf:label>` Etiqueta: `</xf:label>`
	`</xf:input>`		

`output`		`ref`=exp. *XPath* | `bind`=IDREF | `nodeset`=exp. *XPath*	Dato de instancia
	Ejemplo: `<xf:output ref="instance('idInstancia')/nodo"/>`		

secret		ref=exp. *XPath* | bind=IDREF | nodeset=exp. *XPath*	Dato de instancia
	Ejemplo: <xf:secret ref="instance('idInstancia')/nodo"/>		
textarea		ref=exp. *XPath* | bind=IDREF | nodeset=exp. *XPath*	Dato de instancia
	Ejemplo: <xf:textarea ref="instance('idInstancia')/nodo"/>		
select		ref=exp. *XPath* | bind=IDREF | nodeset=exp. *XPath*	Dato de instancia
	Ejemplo: <xf:select> <xf:itemset nodeset="instance('idInstancia')/nodo"> <xf:label ref="etiqueta"/> <xf:value ref="valor"/> </xf:itemset> </xf:select>		
select1		ref=exp. *XPath* | bind=IDREF | nodeset=exp. *XPath*	Dato de instancia
	Ejemplo: <xf:select1> <xf:itemset nodeset="instance('idInstancia')/nodo"> <xf:label ref="etiqueta"/> <xf:value ref="valor"/> </xf:itemset> </xf:select1>		

Nota: La expresión «exp. *XPath*» se refiere a una expresión *XPath*, «IDREF» se utiliza para nombrar a un identificador. Los ejemplos suponen que el *Namespace* de *XForms* se ha definido como `xf`.

Todos los controles pueden incluir el atributo `id` como parámetro para definir su identificador. Además pueden contener al elemento `label`, como se muestra en el ejemplo del *input*.

IV.3.2.2 Representación de las «acciones *XForms*»

En este apartado se revisa la estructura para la declaración de «acciones *XForms*». Como en el caso de los «controles», se requiere un conjunto de parámetros para definir una «acción *XForm*», además algunas de ellas requieren la definición de otros elementos con los cuales trabajan en conjunto.

La Tabla V hace una síntesis de la declaración de las «acciones *XForms*».

Tabla V. Síntesis de la declaración de acciones XForms.

Acción	Parámetros	Referencia a objetivo	Elemento objetivo
message	level = modal | ephemeral | modeless		
	Ejemplo:		

	<xf:message level="modal">Texto de Mensaje</xf:message> ó <xf:message level="modal" ref="//mensaje"/>		
setvalue	value = exp. *XPath* | valor alfanumérico	ref = exp. *XPath* | bind = IDREF	Dato de instancia
	Ejemplo: <xf:setvalue value"6461223344" ref="//Persona/Numero"/> ó <xf:bind id="idBind" ref="//Persona/Numero"/> ... <xf:setvalue value"6461223344" bind="idBind" />		
setfocus		control = IDREF	control *XForms*
	Ejemplo: <xf:setfocus control="idControl"/> ... <xf:input id="idControl" ref="//Nombre" />		
send		submission = IDREF	submission
	Ejemplo: <xf:model> ... <xf:submission id="idSubmit" replace="instance" action="status.xml" method="post"/> </xf:model> ... <xf:send submission="idSubmit"/>		
reset		model = IDREF	model
	Ejemplo: <xf:reset modelo="idModel"/>		
load	resource = URI | ref = exp. *XPath* show = replace | new		
	Ejemplo: <xf:load resource="http://www.google.com/" show="replace"/> ó <xf:load ref="//url" show="new"/>		
toggle		case = IDREF	switch / case
	Ejemplo: <xf:toggle case="case0"/> ... <xf:switch> <xf:case id="case0"> <xf:input ref="instance('instancia')/nodo0"/> </xf:case> <xf:case id="case1"> <xf:input ref="instance('instancia')/nodo1"/> </xf:case> </xf:switch>		
insert	at = 1 | last() | index('IDREF') position = after | before	nodeset = exp. *XPath*	repeat
	Ejemplo: <xf:repeat id="idRepeat" nodeset="/Lista/Persona"> <xf:input ref="Nombre"/> <xf:input ref="Telefono"/> </xf:repeat> ... <xf:action ev:event="Event1"> <xf:insert nodeset="/Lista/Persona" at="last()" position="after"/> <xf:setvalue ref="/Lista/Persona[last()]/Nombre">Gabriel</xf:setvalue> <xf:setvalue ref="/Lista/Persona[last()]/Telefono">044646</xf:setvalue> </xf:action>		
delete	at = 1 | last() | index('IDREF')	nodeset = exp. *XPath*	repeat
	Ejemplo: <xf:delete nodeset="/Lista/Persona" at="last()"/>		

setindex	index = número entero	repeat = IDREF	repeat
	Ejemplo: <xf:setindex repeat="idRepeat" index="1"/>		
dispatch	name = evento *XForms* bubbles = true | false cancelable = true | false	target = IDREF	acción *XForms* control *XForms*
	Ejemplo: <dispatch name="xforms-refresh" target="idModelo"/>		

Nota: Los ejemplos suponen que el *namespace* de *XForms* se define como `xf`.

Cuando las acciones se declaran de manera independiente requieren especificar el parámetro `ev:event` relacionado a un «Evento *XForms*», a un «evento» personalizado, o perteneciente a otro *Namespace*. En otro caso si la acción es declarada dentro de la acción `action` de *XForms*, que actúa como un contenedor de acciones, este contenedor es el que especifica el parámetro del «evento». Todas las acciones declaradas dentro de la acción `action` son desatadas bajo dicho «evento». Otro parámetro que puede llevar una «acción» es el `id` el cual define su identificador.

IV.3.2.3 Representación de los «eventos *XForms*»

Como se define en la sección III.3.3, *XForms* se basa en *XML Events* para la definición de «eventos», por lo tal para la declaración de «eventos», se adopta la tercera forma definida por *XML Events*, en la cual al «manejador» se le adjunta el «evento» y «elemento objetivo». Dado el enfoque de la estrategia, durante la generación del «documento *XForms*» se definen «eventos» genéricos, los cuales únicamente se utilizan para hacer la conexión lógica entre un cambio en el «contexto» y la «acción *XForms*».

IV.3.2.4 Estructura del «modelo de elementos»

En base a la revisión de la declaración de elementos *XForms* se define un conjunto de propiedades que debe incluir una entidad del «modelo de elementos» para poder generar el elemento *XForms* correspondiente. Por otro lado en relación a las adaptaciones que puede tener una entidad, se definen cinco tipos obtenidos del escenario planteado al inicio de este capítulo.

La Figura 19 muestra el modelado en *UML* del documento esquema definido para describir un «modelo de elementos», en este diagrama se muestra la jerarquía y la estructura pertinentes. En el Apéndice A se encuentra el documento esquema definido.

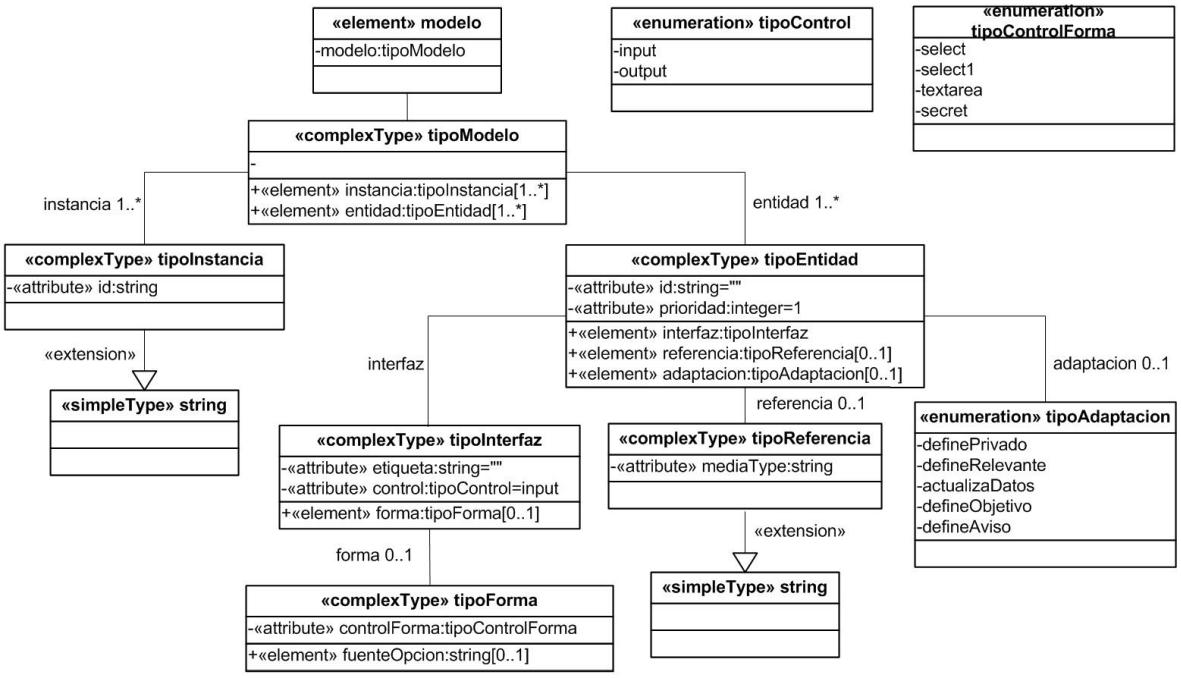

Figura 19. ***Modelado UML del «modelo de elementos».***

La Figura 20 muestra el ejemplo de la definición de una entidad del «modelo de elementos» en este caso se trata de una imagen, la cual contiene información considerada como privada y por lo tal se utiliza la adaptación `definePrivado`.

```
1 <entidad prioridad="1">
2   <interfaz control="output"/>
3   <referencia mediaType="image/*">instance('datos')/NIP</referencia>
4   <adaptacion>definePrivado</adaptacion>
5 </entidad>
```

Figura 20. ***Ejemplo de la definición de una entidad del «modelo de elementos».***

Las propiedades que describe el modelo de elementos se describen en la Tabla VI.

Tabla VI. Propiedades del «modelo de elementos».

Propiedad	Descripción
`modelo`	Elemento que define la raíz del documento.
`instancia`	Elemento que contiene la URI del archivo *XML* que define la instancia de datos a la que harán referencia las entidades del modelo.
`entidad`	Elemento que define cada una de las entidades del modelo.
`prioridad`	Atributo que especifica la prioridad de la entidad, utilizado en la «primera etapa de adaptación». La prioridad más alta es 1, inferiores a ésta son los números enteros mayores a 1.
`id`	Atributo que especifica un identificador único para la entidad, se utiliza

	cuando se requiere definir una «regla de adaptación» que afecte sólo a una entidad determinada.
`interfaz`	Elemento que indica el tipo de control a generar.
`etiqueta`	Atributo que especifica la incorporación de un elemento `label` y la leyenda que contendrá.
`control`	Atributo que define el control especifico a generar: `input` o `output`. Si no se especifica se define alguno de los otros controles contemplados.
`forma`	Elemento utilizado para definir un control tipo `secret`, `textarea`, `select` y `select1`.
`fuenteopcion`	Elemento que indica el archivo *XML* del cual obtienen sus valores los elementos `select` y `select1`.
`referencia`	Elemento que indica el nodo de la «instancia de datos» al cual hace referencia un control.
`mediaType`	Atributo que indica el tipo de información al que hace referencia el nodo de la instancia de datos, éste puede ser texto (`text/plain`) o una imagen (`image/*`).
`adaptación`	Este elemento define uno de los cinco tipos de adaptaciones posibles que un elemento dentro de la «interfaz de usuario» concreta puede tener.

La Tabla VII describe cada uno de las cinco tipos de adaptaciones posibles.

Tabla VII. Descripción de los tipos de adaptaciones.

Adaptación	**Descripción**
`definePrivado`	Este tipo de adaptación se utiliza cuando se desea ocultar una parte de la interfaz de usuario. Se implementa a través de la acción `toggle` y los elementos `switch` y `case`.
`defineRelevante`	Este tipo de adaptación se utiliza para condicionar la aparición de información en la interfaz de usuario. Se implementa a través de la propiedad `relevant` la cual se añade al dato de instancia relacionado.
`actualizaDatos`	Este tipo de adaptación se relaciona con la actualización de las instancias de datos en la interfaz de usuario. Se implementa a través de la acción `send` y el elemento `submission`.
`defineObjetivo`	Este tipo de adaptación se utiliza para establecer el objetivo de atención en un control específico. Se implementa a través de la acción `setfocus`.
`defineAviso`	Este tipo de adaptación define una notificación en la interfaz de usuario. Se implementa a través de la acción `message`.

Las decisiónes sobre el tipo de «acción *XForm*» a utilizar están contenidas en la implementación del «compositor *XForms*».

Posiblemente el nombre de la adaptación `definePrivado` hace referencia a una cuestión muy específica, sin embargo puede ser utilizado en otro tipo de escenario que no necesariamente requiera definir información de tipo privada. Por ejemplo, para establecer campos de información apropiados a «distintos tipos de usuario», por ejemplo, a un paciente consultando información de su condición actual en una pantalla se le pueden ocultar campos con información muy técnica, que a un médico en cambio sí se le mostrarían. Aquí el «parámetro contextual» utilizado puede ser el «tipo de usuario» que consulta la información, la actualización del valor en dicho parámetro se da cuando el «tipo de usuario» que consulta la pantalla cambia, en este caso un «sistema consciente del contexto», notificaría al cliente en cuestión del evento relacionado a este cambio contextual lo que activaría la acción `toggle` que ocultaría o no los campos afectados. La aplicación de esta adaptación se enfoca a ocultar alguna información de la interfaz, como en el caso de la persona que consulta información privada en su casa.

La privacidad mencionada en el modelo se relaciona al contenido de la interfaz, diferenciando si alguna información puede ser desplegada o no en base a la situación actual. La privacidad relacionada al manejo de datos y transacciones no se contempla ya que *XForms* puede utilizar el estándar de «Plataforma para Preferencias de Privacidad» (*P3P*, por sus siglas en ingles) de la *W3C* para definir la privacidad relacionada a la recolección y transferencia. Dicho estándar especifica un método por el cual los recursos y los consumidores de recursos en internet pueden comunicar a otros qué información privada se permite intercambiar y cual se espera o necesita intercambiar para llevar a cabo una transacción. Además, ofrece a los usuarios la flexibilidad de decidir qué información personal comparten y a quién. Esto permite a agentes de usuario acceder a dicha información y advertir al usuario sobre su utilización.

Posiblemente el efecto visual que presentan la acción `toggle` y la propiedad `relevant`, sean iguales, sin embargo cada uno se enfoca a un aspecto diferente. En el caso de `toggle`, trabaja a nivel de la estructura de la interfaz, en conjunto con el elemento `switch` el cual permite agrupar conjuntos de elementos de la interfaz y habilitarlos para ser desplegados al

usuario de manera selectiva. Cada grupo se encapsula en un elemento `case`, los cuales se rigen por un atributo booleano que determina cuál está activo, y es el que se despliega al usuario. Cada elemento `case` se distingue por un identificador. Cada acción `toggle` se asocia a un elemento `case` a activar. En el caso de la propiedad `relevant`, ésta se asocia a un nodo de la «instancia de datos» a través de un elemento `bind`. Los controles relacionados a este nodo serán habilitados o se harán visibles cuando la expresión *XPath*, definida a la propiedad `relevant` sea verdadera. De aquí se observa que `toggle` trabaja a nivel de los «controles de la forma» y la propiedad `relevant` a nivel del «modelo *XForms*». El uso de `defineRelevante` puede darse para desplegar información detallada de las actividades a realizar.

La adaptación `actualizaDatos` define la actualización de la «instancia de datos», esto es posible gracias a los parámetros que se pueden definir al elemento `submission`. Desafortunadamente la especificación actual de *XForms* sólo contempla la actualización de la instancia completa, no se permite realizar actualizaciones sobre nodos particulares, por lo tal la ejecución de la acción actualiza toda la instancia y con esto todos los controles relacionados. Una extensión que se pudiera realizar a *XForms* sería permitir realizar actualizaciones sobre nodos específicos. La aplicación de este tipo de adaptación es para actualizar la información actual que está contenida en la instancia de datos y dada la situación actual es necesaria realizar la actualización.

La adaptación `defineObjetivo` permite situar el foco en un control específico. En el caso de su aplicación, en una página web podría guiar sobre qué control observar, por ejemplo, para guiar la secuencia de la preparación de alimentos. En un dispositivo de audio, en el que los controles de salida se traducen a salidas auditivas, permitiría definir qué control presentar asociado a una salida.

Por último, la adaptación `defineAviso` muestra un aviso con un determinado nivel de intrusión, desde un mensaje efímero hasta un mensaje que requiere ser atendido. En un dispositivo de audio el efímero podría traducirse a una alerta de unos segundos y el otro en una alerta que se mantiene mientras no sea atendida.

Además del manejo simple que se hace de las características, pueden definirse relaciones entre ellas, de manera que la presencia de ciertos valores denote algún significado, por

ejemplo una entrada establecida como privada puede mapearse al elemento `secret` el cual define un elemento de entrada que despliega asteriscos para cada carácter introducido, manteniendo así la privacidad sobre la información mostrada en pantalla.

El manejo de solicitudes del cliente y la forma de servir las páginas se deja al sistema correspondiente, por tal la definición del «modelo de elementos» está relacionada a cada interfaz posible dentro de la aplicación.

IV.3.3 Reglas de Adaptación

Al igual que el «modelo de elementos», la estructura de las «reglas de adaptación» se rige por un documento esquema definido en *XML Schema*, donde se describe la estructura y restricciones. Las reglas se basan en las presentadas en [Hanumansetty, 04], a las cuales se les realizan las modificaciones pertinentes para encajar en el enfoque de la propuesta de este trabajo. Éstas se componen de una sección de condición y otra sección de acciones. La sección de condición se adopta casi en su totalidad, eliminando elementos específicos, y añadiendo elementos relacionados a la referencia a entidades del «modelo de elementos», la sección de acciones se replantea totalmente debido a la diferencia en los objetivos buscados.

Las reglas se involucran en la primera y «tercera etapa de adaptación». Como se ha mencionado se definen dos tipos de reglas: las que se enfocan a la inclusión de elementos y las que definen las condiciones contextuales para ejecutar una adaptación en la «interfaz de usuario». Para definir el valor de los parámetros contextuales bajo el cual se debe notificar el «evento» relacionado, las reglas sólo reflejan valores puntuales. Esto se basa en la decisión de que el «servidor de contexto» es el responsable de decidir cuándo se debe llevar a cabo una adaptación.

Con *XForms* se podría definir a través de expresiones *XPath* las condiciones bajo las cuales se dispara una acción, esto sería manteniendo una «instancia de datos» la cual refleje una versión reducida de algunos «parámetros contextuales» y sus valores actuales. Esta estrategia requeriría que el servidor le envíe la información correspondiente al cliente y de algún modo realizar actualizaciones a los nodos de la «instancia de datos».

La especificación de *XForms*, contempla que si alguna entidad s*cript* modifica el estado de una «instancia de datos» o un control, la actualización de las instancias relacionadas a los elementos afectados debe ser llevada por la misma entidad *script*. Por ejemplo si mediante una entidad *script* se actualiza el valor de un nodo de la «instancia de datos», dicho *script* es el responsable de actualizar el estado del control relacionado y con esto la vista actual de la interfaz de usuario. Esta práctica puede alterar la integridad del estado de la interfaz si no se llevan a cabo las actualizaciones pertinentes.

La Figura 21 muestra el modelado en *UML* del documento esquema definido para describir las «reglas de adaptación, en este diagrama se muestra la jerarquía y la estructura pertinentes. En el Apéndice B se encuentra el documento esquema definido.

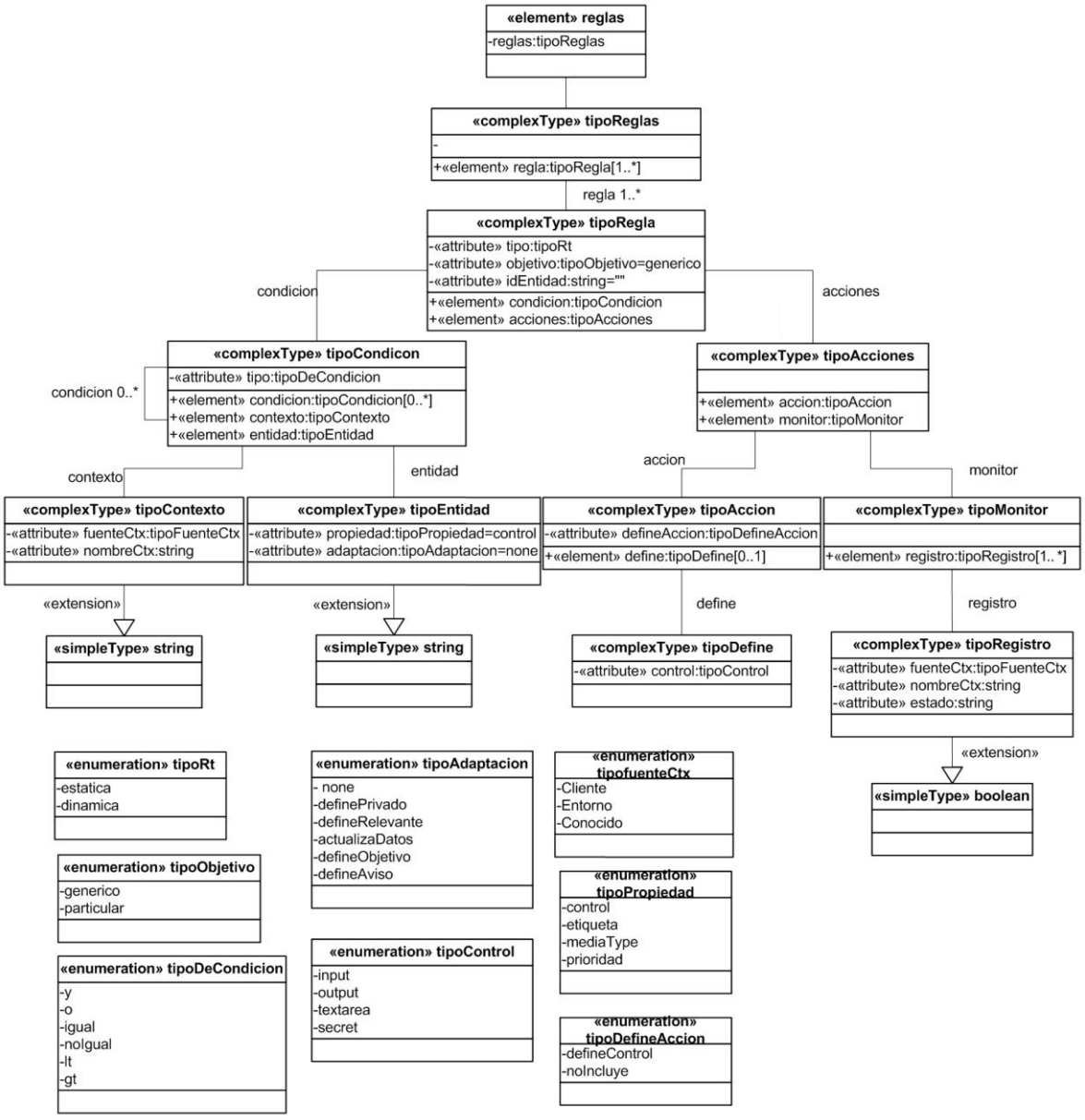

Figura 21. ***Modelado UML de las «reglas de adaptación ».***

La Figura 22 muestra un ejemplo de la definición de dos «reglas de adaptación», en la primera se define que si la especificación del dispositivo indica una falta de soporte a imágenes y la entidad del «modelo de elementos» en cuestión es de tipo imagen, entonces dicho elemento no se debe incluir. La regla en cuestión es de tipo estática lo que indica que se utiliza durante la «primera etapa de adaptación». La segunda regla indica que es de tipo dinámica, por lo tal durante la «primera etapa de adaptación» define los valores de los

«parámetros contextuales» que afectan a todas las entidades que incluyen el tipo de adaptación que la regla define. Es posible definir reglas específicas a entidades particulares, esto se lleva a cabo utilizando los atributos: `objetivo` con el valor '`particular`' e indicando el identificador de la entidad correspondiente en el campo `idEntidad`.

```
<regla tipo="estatica" objetivo="generico">
  <condicion tipo="y">
    <condicion tipo="igual">
      <contexto fuenteCtx="Cliente" nombreCtx="Plataforma/SoporteImagenes">falso</contexto>
    </condicion>
    <condicion tipo="igual">
      <entidad propiedad="mediaType">image/*</entidad>
    </condicion>
  </condicion>
  <acciones>
    <accion defineAccion="noIncluye"/>
  </acciones>
</regla>
```

```
<regla tipo="dinamica" objetivo="particular" idEntidad="idControl4">
  <condicion tipo="igual">
    <entidad adaptacion="definePrivado"/>
  </condicion>
  <acciones>
    <monitor>
      <registro fuenteCtx="Entorno" nombreCtx="Compania/Estado" estado="noSolo">true</registro>
      <registro fuenteCtx="Entorno" nombreCtx="Compania/Estado" estado="solo">false</registro>
    </monitor>
  </acciones>
</regla>
```

Figura 22. ***Ejemplo de la definición de dos «reglas de adaptación».***

La descripción de los elementos para componer las «reglas de adaptación» se muestran en el Tabla VIII.

Tabla VIII. Propiedades de las «reglas de adaptación».

Propiedad	Descripción
`reglas`	Elemento que define la raíz del documento.
`regla`	Elemento que se utiliza para definir cada una de las reglas.
`tipoRt`	Atributo para definir si la regla es estática o dinámica.
`Objetivo`	Atributo para definir si la regla es general o particular a una entidad del «modelo de elementos»
`identidad`	Atributo para indicar el identificador de la entidad del «modelo de elementos» si la regla es particular.
`condicion`	Elemento utilizado para indicar el comienzo una sección de condiciones de la regla. Las condiciones compuestas se definen anidando elementos de tipo condicion.
`tipoDeCondicion`	Indica el tipo de condición lógica que se aplica.
`contexto`	Elemento para definir los parámetros contextuales a condicionar.
`fuenteCtx`	Atributo para indicar el tipo de contexto al cual hace referencia la regla.
`nombreCtx`	Atributo para indicar el componente y atributo contextual al cual hace referencia la regla.
`entidad`	Elemento para definir las propiedades de las entidades del «modelo de

	elementos» a condicionar.
`control`	Atributo para indicar la propiedad de la entidad del «modelo de elementos» a condicionar.
`adaptacion`	Elemento para especificar el tipo de adaptación al cual se le definirá el valor de los «parámetros contextuales» a monitorear.
`acciones`	Elemento utilizado para indicar el comienzo de la sección de acciones de la regla.
`accion`	Elemento para indicar la acción a llevar a cabo dado el cumplimiento de la condición.
`defineAccion`	Atributo que incluye el tipo de acción específica a realizar.
`define`	Elemento utilizado en caso que una acción requiera la definición de un campo específico dentro de la «interfaz abstracta»
`monitor`	Elemento para indicar los valores de los «parámetros contextuales» a monitorear.
`tipoRegistro`	Atributo para describir el «parámetro contextual».
`estado`	Atributo para definir el valor del «parámetro contextual».

El elemento `registro` es de tipo booleano, además de indicar la descripción del «parámetro contextual», requiere que se le especifique un valor `true` o `false`. El valor `true` indica las condiciones especificas bajo las cuales se lleva a cabo la adaptación, en caso de tener el valor `false` indica que se lleve a cabo el proceso inverso de la adaptación. Esta condición del proceso inverso no aplica a todas las adaptaciones. En el caso de `definePrivado` sí es posible dado que puede ocultarse la sección de la interfaz y más adelante volver a desplegarla, en el caso de `actualizaDatos` es una adaptación que no puede revertirse.

IV.3.4 Compositor XForms

Como se ha descrito la tarea del compositor es generar el «documento *XForms*», a partir de los tres modelos definidos en las secciones anteriores. Su implementación consiste en el parseo de los documentos conociendo su estructura, sintaxis y relación.

El primer documento a analizar es el «modelo del contexto», generando registros temporales simplificados del perfil *CC/PP* para acceder y utilizar la información de manera más sencilla. El segundo documento es el «modelo de elementos», a la par se analizan las «reglas de adaptación». En el «modelo de elementos» para cada entidad se analizan una a una sus propiedades, en este paso cada entidad es comparada con las «reglas de

adaptación» y se llevan a cabo las acciones de inclusión o definición de controles, de igual manera en el caso de las adaptaciones se implementa las estructuras de la «acción *XForms*» que se plantean en la Tabla VII de la sección IV.3.2.4.

El compositor genera identificadores únicos de «eventos» y «acciones» asociados a los valores de los «parámetros contextuales» definidos en las «reglas de adaptación» de tipo dinámicas. A partir de aquí genera los registros los cuales siguen la relación lógica definida en la Tabla III. La estrategia utilizada en la «tercera etapa de adaptación» es que el servidor realiza las validaciones y correspondencias entre registros y el estado del «contexto» actual, notificando al cliente del evento correspondiente de manera que se ejecute la «acción *XForms*» relacionada.

La estructura utilizada para los registros consta del tipo de contexto (cliente, entorno o conocido), su componente y parámetro, especificados de la manera en la que *XPath* referencia nodos anidados, el valor deseado y el identificador único, por ejemplo: `Entorno compañía/Estado solo 1`.

En el capítulo V se implementa un «compositor *XForms*» utilizando el lenguaje de programación Java, en conjunto con un arquitectura que simula un «servidor de contexto».

IV.4 Resumen y Conclusiones

Los elementos definidos en este capítulo se apegan a la propuesta de estrategia definida. Inicialmente se consideran cinco tipos de adaptación para la «tercera etapa de adaptación» contemplada, las cuales se obtuvieron de la especificación de los escenarios de aplicación considerados.

Con el enfoque propuesto, la adaptación de la interfaz en base a los recursos del dispositivo se deja al intérprete *XForms* pertinente, a partir de aquí la propuesta se centra en la adaptación de la «interfaz concreta funcional». Las adaptaciones diseñadas se encuentran descritas de manera independiente del dispositivo al utilizar *XForms* y *XML Events*, de aquí que sea el mismo interprete el encargado de llevarlas a cabo.

Debido a la manera en la que se definen los elementos se consideran dos tipos de extensión posibles:

1. La primea está dada por el uso de los elementos de adaptación en otros escenarios posibles. Esto involucra definir cada uno de los elementos correspondientes en base a los requisitos de la aplicación y guiándose por las estructuras planteadas.
2. El segundo tipo de extensión involucra definir nuevos tipos de adaptación, lo que conlleva la extensión de los elementos propuestos, los modelos, las reglas y el compositor. Una cuestión a considerar en este tipo de extensión es que es necesario conocer *XForms* y definir bajo que escenario podría aplicarse la extensión propuesta.

Debido a la propia especificación de *XForms* las aplicaciones desarrolladas se apegan al patrón arquitectónico *MVC*. De hecho, la misma estructura de *XForms* da la pauta para establecer la estrategia de adaptación. Además, al no dar soporte a modelos declarativos como el de tareas, únicamente se contempla la estructura de la interfaz y la «instancia de datos» como los objetivos de adaptación en la «tercera etapa de adaptación».

Capítulo V

Escenarios de Aplicación

V.1 Introducción

Para utilizar y validar la estrategia de adaptación presentada en el capítulo anterior, se han diseñado e implementado dos escenarios de aplicación que reflejan un conjunto de requisitos de adaptación de información, los escenarios se sitúan en ambientes dotados de una arquitectura que da soporte para la detección y estimación del «contexto»». El cumplimiento de los requisitos de adaptación presentados en dichos escenarios dará la pauta inicial para la validación de la estrategia de adaptación propuesta. Además es necesario hacer una revisión de la factibilidad de la extensión de los elementos de adaptación en caso necesario para cumplir los requisitos planteados.

V.1.1 Consideraciones preliminares

Para implementar y utilizar los elementos de adaptación de la «interfaz de usuario» dentro de un «sistema consciente del contexto», es necesario que se realicen las siguientes tareas:

1. A partir de los requisitos de la aplicación, definir los documentos esquema y perfiles para el «modelo del contexto», identificando los «parámetros contextuales» relevantes, el posible rango de valores que manejan, e identificar la naturaleza dinámica ó estática de dichos parámetros.

2. Especificar las «reglas de adaptación» definiendo las condiciones contextuales, las propiedades de las entidades del «modelo de elementos» y las acciones a ser tomadas.
3. Definir los «modelos de elementos» relacionados a cada «interfaz de usuario» que requiere cada servicio ofrecido.

En un «sistema consciente del contexto», tanto el «modelo de elementos» como el «modelo de contexto» deberían ser generados al momento de la solicitud del servicio, ya que:

1. El «modelo del contexto» debe reflejar el estado del «contexto» actual, de manera que las decisiones sobre la generación de la interfaz sean acordes a la situación presente.
2. El «modelo de elementos», puede pertenecer a algún servicio que maneja información dinámica en el tiempo. Por ejemplo, un servicio de notificación de actividades a realizar, aquí la información presentada debe ser adecuada a la hora en la que se realiza la solicitud del servicio.

Por otro lado, las «reglas de adaptación» deben ser específicas a la aplicación, ya que ahí se plasman las condiciones de adaptación impuestas a dicha aplicación. En el caso de los escenarios, tanto los modelos como las reglas se predefinen, esto con el propósito de ilustrar cómo dichos elementos trabajan en conjunto. Sin embargo, se detalla la manera de definir cada modelo.

V.2 Escenario 1. Apoyo en el Hogar a un Adulto Mayor

Un área de estudio en donde tecnologías como los «sistemas conscientes del contexto» han tomado partido, dada el área de oportunidad, es el hogar. En dicha área un tópico de interés es el apoyo a la vida independiente de adultos mayores, con el fin de que puedan permanecer en su hogar el mayor tiempo posible antes que requieran ser trasladados a centros de ayuda y cuidado. Esto con el fin de brindarles una mejor calidad de vida. El apoyo dado a estas personas en su hogar puede realizarse a través de sistemas de monitoreo y alarmas (Haigh *et al*., 02) o aplicaciones que les recuerden sobre actividades a realizar, como por ejemplo la toma de medicinas (Mynatt *et al*., 00).

Descripción: en este escenario se presentan con más detalle dos posibles situaciones de las contempladas en el Capítulo IV relacionadas con el apoyo en el hogar a un adulto mayor. La primera contempla la presentación de información detallada sobre las actividades futuras una vez que se han completado las actuales. La segunda situación describe la adaptación de la interfaz cuando se encuentra desplegando información catalogada como privada y se presenta una situación que compromete la información a pantalla, por ejemplo, la presencia de otras personas en el entorno.

Objetivo del escenario: corroborar cómo la propuesta encaja en los requisitos de adaptación contemplados en el escenario. De esta manera se verifica cómo *XForms* provee conceptos de manera nativa que pueden ser utilizados para el desarrollo de un «sistema consciente del contexto». Obviamente bajo la premisa que *XForms* puede ser utilizado más allá que sólo que para implementar formas web y los elementos que presenta son totalmente independientes de plataforma, lo que permite definir una interpretación adecuada en otro tipo de aplicaciones.

Escenario:

Don Fernando en un adulto mayor que vive solo, su hogar está dotado de una infraestructura tecnológica que es capaz de dar soporte a un «sistema consciente del contexto», ya que se conforma de dispositivos de sensado, cámaras, monitores, etc., lo cual permite obtener información del ambiente, además de realizar inferencias y tareas afines para conocer el estado actual del entorno. El sistema forma una red de información la cual está a la disponibilidad de don Fernando. Dicho sistema apoya a don Fernando en sus actividades básicas de cada día, ya que sufre una afectación cognoscitiva la cual le impide llevar una vida normal al olvidar pasajes de su día e incluso llevar a cabo actividades elementales como asearse o alimentarse. Imagine un día común:

Don Fernando ha terminado de desayunar y se encuentra leyendo el periódico. En la mesa se encuentra un dispositivo que permite saber si Don Fernando ha tomado su medicina, lo cual realiza en base al sensor que trae incorporado. En la mesa también se encuentra una pantalla digital. Siendo las 8:55 de la mañana el sistema toma consciencia de la próxima actividad a realizar, la cual es tomar la medicina de las 9:00 am. Justo a la hora indicada don Fernando sigue leyendo el periódico aún en el desayunador, el sistema en cuestión

ubica a don Fernando en este lugar, así como a la pantalla digital que se encuentra cerca de él. La pantalla es encendida y se le envía la información a desplegar. Cuando la pantalla se enciende llama la atención de don Fernando el cual se hace consciente de la información que le están presentando, en este caso la toma de medicina correspondiente. Don Fernando toma el frasco de medicinas e ingiere su pastilla. El dispositivo que contiene al frasco detecta que ha sido retirado, con lo cual infiere que don Fernando ha tomado su pastilla, lo cual notifica al «sistema consciente del contexto», que a su vez envía una notificación al cliente, en este caso la pantalla digital, para que despliegue la información detallada de la próxima actividad, es decir, la consulta médica de las 10:00 am. Esto es únicamente con el objetivo de recordarle sobre la actividad en cuestión. Más adelante se le recordará una vez más de la consulta médica. Cada información detallada se asocia a la realización de la actividad inmediata anterior. En este caso la información detallada sobre la consulta médica se asocia a la toma de la medicina de las 9:00 am. La Figura 23 resume gráficamente el escenario planteado.

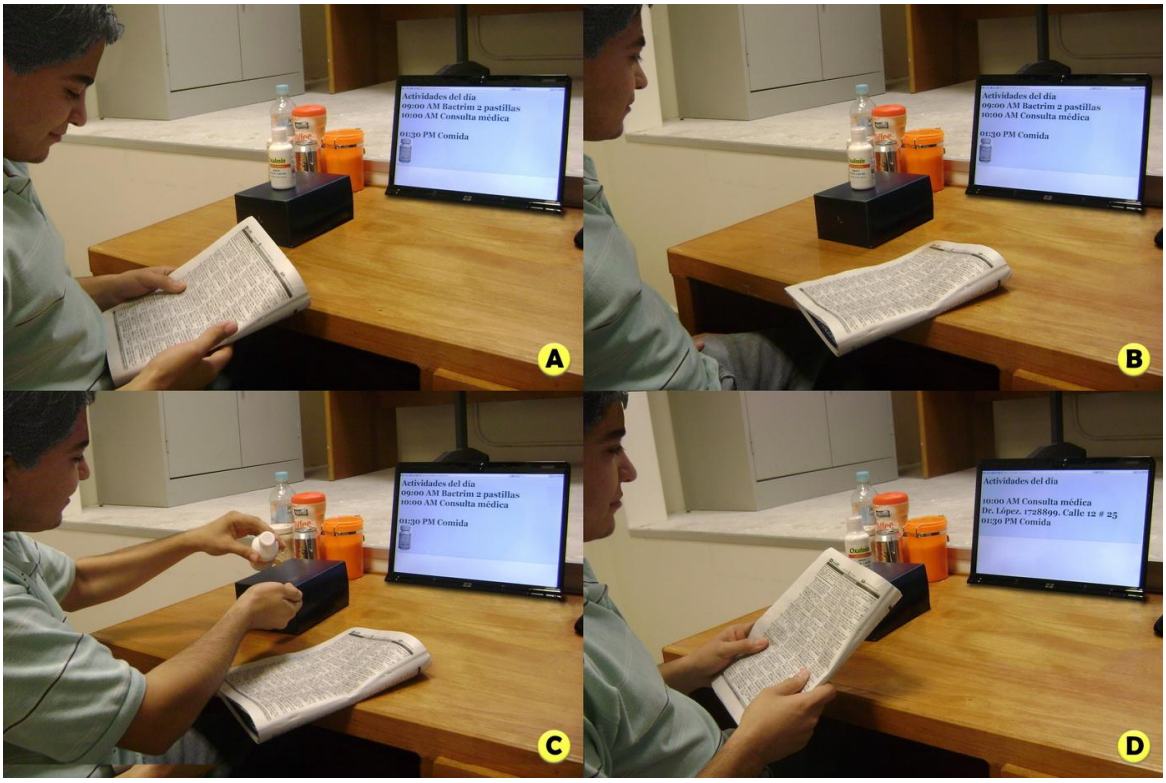

Figura 23. ***Notificación de actividades del día donde se muestra información relevante.***

Por la tarde don Fernando necesita retirar dinero para pagar la despensa semanal que llegará mañana, por lo cual necesita consultar información personal sobre su cuenta de banco. Para ello se acerca a una pantalla en la sala de su casa, ésta dispone de un sensor que le permite saber que alguien se encuentra frente a ella, en base a esto se muestra un menú con las opciones disponibles, éstas se relacionan a la consulta de información, por ejemplo actividades pendientes y realizadas, números de emergencia e información personal. Don Fernando selecciona la opción de -información bancaria-, el «sistema consciente del contexto» detecta que la persona que solicita la información es don Fernando, ya que porta en su ropa un pequeño identificador electrónico, por lo tal, se envía y despliega en el cliente la información bancaria la cual está catalogada como privada. En el momento en el que don Fernando se encuentra consultando la información, llega su vecino solicitando el periódico del día el cual no pudo conseguir, un dispositivo de monitoreo detecta la nueva presencia, la cual comunica al «sistema consciente del contexto», que a su vez notifica dicha información al cliente, que en ese momento se encuentra desplegando la información y como respuesta a esto oculta la información catalogada como privada sobre la cuenta, esto se lleva a cabo sin la interacción explicita del usuario. La Figura 24 resume gráficamente el escenario planteado.

Figura 24. ***Consulta de información catalogada como de tipo privada.***

V.2.1 Implementación

V.2.1.1 Diseño de Arquitectura

La arquitectura propuesta para la implementación de ambos episodios del escenario se muestra en la Figura 25 en la cual se ilustra el «diagrama de emplazamiento». Este

diagrama muestra la relación física que guardan los componentes de *software* con los componentes de *hardware*. En el lado del «cliente» se requiere de un «navegador» que permita acceder a los servicios ofrecidos, por lo tal, debe contar con un «intérprete *XForms*», así como con una aplicación que sea intermediaria entre el cliente y las notificaciones publicadas por el «servidor de contexto», dicha aplicación intermediaria se denomina «consumidor de contexto». En el lado del «servidor», se cuenta con un «servidor web» que se encarga de gestionar las solicitudes del cliente. El «servidor de aplicaciones» es el encargado de generar el «modelo de elementos» relacionado al servicio solicitado. El «servidor de contexto» brinda la información sobre el dispositivo desde el que se solicita el servicio, así como del estado actual del «contexto». El «compositor *XForms*» genera la «interfaz de usuario abstracta» que se entrega al cliente. El «servidor de contexto» utiliza el «publicador del contexto» para notificar, por medio de mensajes, los cambios que ocurren en el ambiente. Dicha notificación es consumida por la parte correspondiente en el cliente, es decir, el «consumidor de contexto». Esto se lleva a cabo en tiempo de ejecución del servicio.

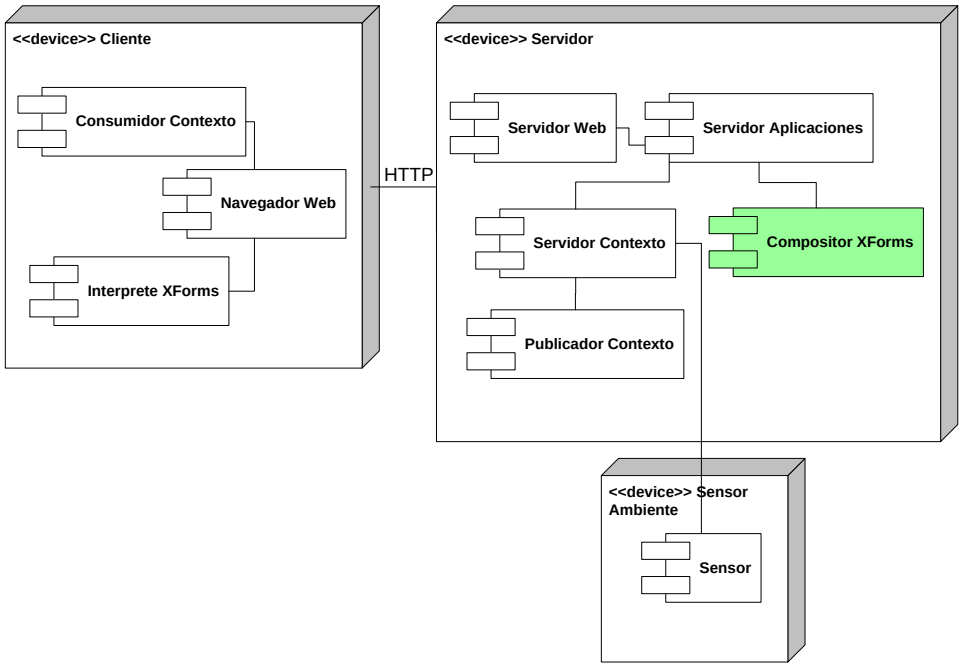

Figura 25. ***Diagrama de emplazamiento propuesto para el escenario 1.***

La Figura 26 define el diagrama de secuencia, que muestra las llamadas realizadas entre los componentes de *software*, su dependencia y el flujo seguido en el proceso de generación y entrega de la «interfaz de usuario», definida en *XForms*, relacionada al servicio solicitado. Una vez que se ha recibido la petición desde el cliente, el «servidor de aplicaciones» solicita la información contextual actual al «servidor de contexto», el cual la entrega en dos «perfiles *CC/PP*», uno relacionado al cliente y otro al entorno, ambos forman el «modelo del contexto». Por su parte el «servidor de aplicaciones» selecciona la información relacionada al servicio solicitado en conjunto con las características que debe reflejar, lo cual plasma en el «modelo de elementos». Una vez que se tienen ambos modelos, se solicita al «compositor *XForms*» generar la interfaz a entregar al cliente. La «reglas de adaptación» se encuentran predefinidas para todos los servicios ofrecidos. La secuencia anterior corresponde a la «primera etapa de adaptación» de la «interfaz de usuario» relacionada a la generación del «documento *XForms*».

La tarea de seleccionar qué contenido mostrar, en base al servicio solicitado, es responsabilidad del «servidor de aplicaciones». Por ejemplo, en base a la hora se selecciona la información relevante. La tarea del «compositor *XForms*» reside en mapear la información definida por el «servidor de aplicaciones» a un «documento *XForms*».

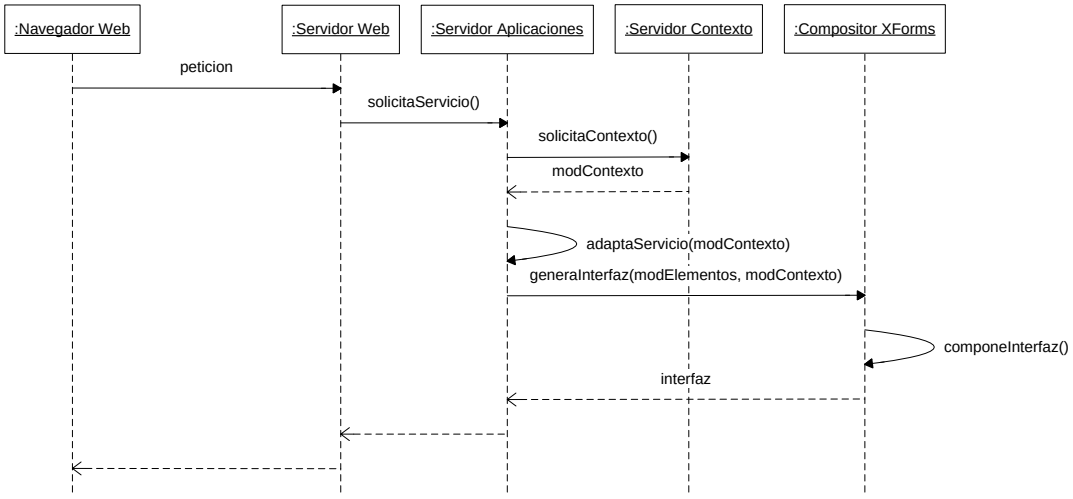

Figura 26. ***Flujo de la generación de la «interfaz de usuario» en XForms.***

La Figura 27 ilustra a través de un diagrama de secuencias el flujo seguido para notificar los cambios ocurridos en el ambiente en tiempo de ejecución del servicio, que corresponde a la «tercera etapa de adaptación» de la «interfaz de usuario». Cuando ocurre un cambio en el ambiente el «servidor de contexto», a través del «publicador del contexto», publica un mensaje que contiene el evento y la acción relacionados a dicho cambio en el ambiente. En el cliente la contraparte consumidora de estos mensajes en respuesta, realiza la vinculación a la «acción *XForms*» correspondiente.

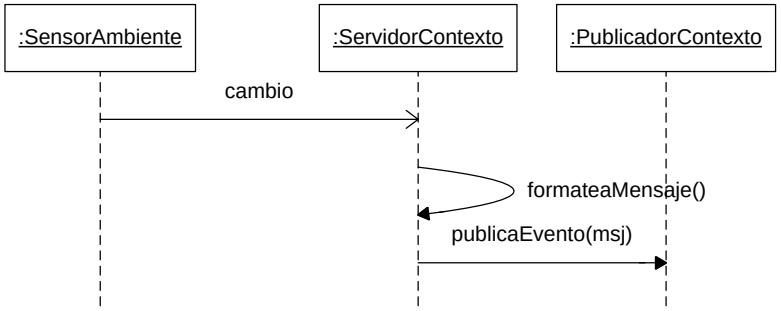

Figura 27. ***Flujo para la notificación de un cambio en el contexto en tiempo de ejecución.***

Como se menciona en el Capítulo IV, la «segunda etapa de adaptación» de la «interfaz de usuario» se relaciona con la traducción del «documento *XForms*» a una «interfaz concreta funcional», tarea que se lleva a cabo por el «intérprete» correspondiente en el cliente.

Una cuestión que se establece en el diseño, es la separación de la funcionalidad del sistema, de las tareas relacionadas a la administración del «contexto». Como se menciona al final de la sección IV.3.2.4, relacionada a la implementación del «modelo de elementos», el manejo de solicitudes del cliente al servidor se deja al sistema anfitrión correspondiente, así como la navegación entre interfaces que pudiera existir, la cual se preestablece y debe ser manejada también por el sistema anfitrión.

Dado que la recolección de datos del ambiente, así como las tareas afines a la manipulación y toma de decisiones propios de un «sistema consciente del contexto» se simulan, los diagramas presentados no entran en detalle a describir dichas tareas.

V.2.1.2 Implementación de la Arquitectura

Como «servidor web» se utiliza Apache Tomcat 6.0.0.14, de manera que cada componente de *software* dentro del servidor se desarrolla como una clase en Java. Tanto el «servidor de aplicaciones» como el «servidor de contexto» se implementan como *servlets*. Por parte del cliente se utiliza una *PC* con el navegador *Mozilla Firefox* 2.0.0.14 y el intérprete *Mozilla XForms* 0.85 para dicho navegador.

El consumidor y el publicador de contexto se desarrollan utilizando las bibliotecas *dojo* 1.0.2 y *Jetty* 6.1.7 que implementan la especificación *Comet*. La biblioteca *dojo* está implementada en *javascript*, por su parte *Jetty* está implementada en Java. El objetivo de *Comet* es intercambiar datos entre el cliente y el servidor. En el enfoque que presenta, el cliente no necesita realizar una solicitud explícita para que el servidor le entregue datos, ya que se mantiene una conexión abierta entre estos y el servidor en cualquier momento puede enviar información al cliente, lo que permite tener una aplicación web basada en eventos (el enfoque también es conocido como *Push Server*).

Para el primer caso del escenario, relacionado a la notificación de la toma de medicinas, el «sistema consciente del contexto» realiza la solicitud al servidor para iniciar el servicio correspondiente, en el segundo caso, relacionado a la consulta de información bancaria, la solicitud se realiza desde el «navegador». Tanto el «modelo de elementos» como el «modelo del contexto» se predefinen, de modo que el «servidor de aplicaciones» solamente entrega el documento ya establecido. El «servidor de contexto» por su parte entrega los «perfiles» ya establecidos, además, lee los documentos correspondientes a dichos «perfiles» en busca de «atributos dinámicos» de modo que mantenga una lista de este tipo de «atributos». Por otro lado, realiza la tarea de monitoreo de los registros que genera el «compositor *XForms*» de eventos a notificar. La actualización de los valores de los «parámetros contextuales» se lleva a cabo a través de la lectura de datos en archivos de texto que hacen el papel de los sensores del ambiente.

A continuación, en las secciones V.2.2, V.2.3 yV.2.4 se describen los dos modelos y las reglas de adaptación utilizados en la «primera etapa de adaptación» de la interfaz la cual contempla la construcción del «documento *XForms*» y la generación de registros de eventos a monitorear en la «tercera etapa de adaptación».

V.2.2 Modelo del Contexto

Se utiliza un mismo «modelo del contexto» para ambos episodios del escenario, se define un «perfil *CC/PP*» para el cliente (dispositivo, usuario) y un «perfil *CC/PP*» para el entorno (ubicación, tarea, etc.). Los documentos esquema y los «perfiles» se pueden consultar en los Apéndices C, D, E y F.

Para el primer caso el «parámetro contextual» de interés es la actividad realizada, por lo tal, en el «perfil» del entorno, el elemento «actividades» se define como un «componente», al cual se le definen «atributos» relacionados a cada una de las actividades de la persona, por ejemplo, desayuno, baño matutino, medicina matutina, etc. Cada una de estas actividades individuales se definen de tipo «dinámico», el valor dinámico de interés se relaciona a conocer si dichas actividades han sido realizadas o no, dicho valor cambiará en el tiempo conforme la persona lleve a cabo cada actividad.

Para el segundo caso el «parámetro contextual» de interés es la compañía, por lo tal en el «perfil» del entorno se considera como un «componente» el elemento «compañía», al cual se le define un único «atributo» de tipo dinámico, dicho «atributo» denominado «estado» indica si la persona se encuentra sola o acompañada.

En relación al «perfil» del cliente, es de interés el «tipo de dispositivo» y su «soporte a imágenes», lo cual se refleja en el documento correspondiente.

La Tabla IX resume los «parámetros contextuales» que son vinculados a entidades del «modelo de elementos» en el escenario 1.

Tabla IX. Resumen de los «parámetros contextuales» vinculados a entidades del «modelo de elementos».

Tipo de Contexto	**Componente**	**Atributo**	**Rango de valores**	**Tipo Atributo**
Entorno	Actividades	MedicinaMatutina	verdadero/falso	Dinámico
Entorno	Compania	Estado	solo/noSolo	Dinámico
Cliente	Plataforma	TipoDispositivo	PC, PDA, etc.	Estático
Cliente	Plataforma	SoporteImagenes	verdadero/falso	Estatico

La decisión de utilizar una baja granularidad en los «perfiles», así como el uso de «parámetros contextuales» muy específicos y con un rango de valores simple, se hace con la idea de ejemplificar la idea de manera sencilla. Como se ha mencionado la granularidad

dependerá de los requisitos de la aplicación, así como el posible rango de valores contemplados.

Es necesario distinguir entre la información contextual que el «sistema consciente del contexto» y el «compositor» utilizan, la cual depende de las tareas que tienen a cargo. El «sistema consciente del contexto» es responsable de seleccionar el lugar de la casa en el cual le será entregada la información a don Fernando, por lo tal, la ubicación es de interés para el sistema. Por otro lado, el «compositor» utiliza la información contextual para generar la «interfaz abstracta», la cual está definida en las «reglas de adaptación», si alguna de ellas requiere que se tome en cuenta la ubicación entonces dicho «parámetro contextual» será de interés para el «compositor», por ejemplo, si la ubicación de la casa desde la que don Fernando accede el servicio no es apropiada para mostrar cierta entidad de información, ésta puede ser omitida en la «interfaz abstracta».

V.2.3 Modelo de Elementos

Los «modelos de elementos», definidos para el escenario 1 se pueden consultar en los apéndices G y H, para cada uno de los casos respectivamente.

Para el primer caso, el «modelo de elementos» está formado de cinco entidades las cuales se describen en la Tabla X. La entidad de interés es aquella relacionada con la información de los «detalles de la consulta médica», el documento *XML* que contiene la «instancia de datos» se llama actividades.xml, y su identificador es `actividades`.

Tabla X. Entidades del «modelo de elementos» para el primer caso del escenario 1.

Entidad	Tipo	Referencia	Adaptación
Servicio	texto	instance('actividades')/servicio	N/A
Medicina	texto	instance('actividades')/medicina	N/A
Consulta	texto	instance('actividades')/consulta	N/A
Detalles Consulta	texto	instance('actividades')/consultad	defineRelevante
Comida	texto	instance('actividades')/comida	N/A

Para el segundo caso, el «modelo de elementos» está formado de cuatro entidades las cuales se describen en la Tabla XI. Las entidades de interés son aquellas relacionadas con la información de la «cuenta de banco» y el «número de tarjeta». El documento *XML* que contiene la «instancia de datos» se llama cuenta.xml, y su identificador es `cuenta`.

Tabla XI. Entidades del «modelo de elementos» para el segundo caso del escenario 1.

Entidad	Tipo	Referencia	Adaptación
Servicio	texto	instance('cuenta')/servicio	N/A
Cliente	texto	instance('cuenta')/cliente	N/A
Cuenta	texto	instance('cuenta')/cuenta	definePrivado
Tarjeta	texto	instance('cuenta')/tarjeta	definePrivado

Las características descritas en las dos Tablas anteriores se plasman en cada «modelo de elementos» siguiendo la estructura definida en el documento esquema.

V.2.4 Reglas de Adaptación

La relación definida entre las entidades del «modelo de elementos» y los «parámetros contextuales» del «modelo del contexto» se refleja en las «reglas de adaptación», las cuales se pueden consultar en el Apéndice I.

Para el primer caso, la «regla de adaptación» de interés se define como de tipo particular, la cual define que la entidad del «modelo de elementos» con el identificador `idConsultaDetalles` que defina el tipo de adaptación `defineRelevante`, se verá afectada cuando el «parámetro contextual» «`Entorno/Actividades/MedicinaMatutina`» tenga el valor «`verdadero`». La notación utilizada para el «parámetro contextual» simplifica la jerarquía dada por *CC/PP*, donde se tiene: «`Tipo de Contexto/Componente/Atributo`».

La entidad de los «detalles de la consulta médica» se declara de tipo `defineRelevante` y con el identificador `idConsultaDetalles`.

Para el segundo caso, la «regla de adaptación» de interés se define de tipo genérica, la cual declara que todas las entidades del «modelo de elementos» que definan el tipo de adaptación `definePrivado`, se verán afectadas cuando el «parámetro contextual» «`Entorno/Compania/Estado`» tenga el valor «`noSolo`».

Las entidades de la «cuenta bancaria» y «número de tarjeta» se definen de tipo `definePrivado`.

Las dos reglas anteriores son de tipo «dinámicas» por lo tal su efecto se ve reflejado en la «tercera etapa de adaptación». Para la «primera etapa de adaptación» se definen dos reglas, la primera dicta la exclusión de imágenes si el dispositivo no tiene soporte para dicho

medio, la segunda dicta la exclusión si la prioridad de la entidad es mayor o igual a tres y el dispositivo en cuestión es una *PDA.*

La Tabla XII resume las cuatro «reglas de adaptación» descritas.

Tabla XII. Descripción de las «reglas de adaptación» para el escenario 1.

Tipo	**Condiciones**	**Acción**
Dinámica	Entidad: `Adaptación = defineRelevante`	Monitorear: `Entorno/Actividades /MedicinaMatutina = verdadero`
Dinámica	Entidad: `Adaptación = definePrivado`	Monitorear: `Entorno/Compania /Estado = noSolo`
Estática	Contexto: `Cliente/Plataforma/SoporteImagenes = falso` y Entidad: `tipo = imagen`	No Incluir
Estática	Contexto: `Cliente/Plataforma/TipoDispositivo = PDA` y Entidad: `prioridad >=3`	No incluir

V.2.5 Proceso de Generación

La Figura 28 muestra a modo de ejemplo las correspondencias entre los tres elementos de adaptación para el primer caso del escenario. Como puede observarse es necesario que exista una consistencia sintáctica al momento de implementar cada modelo así como las reglas. Dichas correspondencias son implementadas por el «compositor *XForms*». En el ejemplo de la Figura en particular, el compositor define en el «documento *XForms*» la «acción *XForms*» correspondiente y genera un registro para monitorear el valor del «parámetro contextual» especificado.

Figura 28. ***Correspondencias entre los elementos de adaptación.***

Al término de la «primera etapa de adaptación» el compositor genera los registros de «eventos» únicos a notificar relacionados a los valores de los «parámetros contextuales» definidos en las reglas de adaptación. En el «documento *XForms*» se definen los «eventos» relacionados con sus «acciones» correspondientes. En la «tercera etapa de adaptación» el «servidor de contexto» se encarga de monitorear los «parámetros contextuales» correspondientes. Cuando un «parámetro contextual» cambia su valor y corresponde con uno de los registros que está monitoreando, envía un mensaje al cliente, el cual contiene el identificador del «evento» y «acción» correspondientes. En el cliente se reciben dichos identificadores con los cuales se ejecuta la «acción». En el apéndice J se puede consultar la rutina utilizada en el lado del servidor y la rutina en el lado del cliente para comunicar y recibir el mensaje mencionado. Cada rutina corresponde al «publicador de contexto» y «consumidor de contexto» respectivamente.

Durante la «segunda etapa de adaptación» se envía el «documento *XForms*» al cliente, dicho documento es traducido por el «intérprete» utilizado.

V.2.6 Resultados

El compositor únicamente se encarga de definir los elementos *XForms* necesarios, la presentación (a través de hojas de estilo) se deja a un lado quedando a consideración del desarrollador. Por otro lado, los elementos que ofrece el lenguaje anfitrión (*XHTML*) se utilizan para definir la estructura de distribución de los objetos (los «controles *XForms*») en la «interfaz concreta funcional».

Por cuestión estética, conociendo el «intérprete» a utilizar y el dispositivo objetivo (*PC*), se incorpora una «hoja de estilos» para presentar las «interfaces de usuario» una vez que han sido entregadas al cliente e interpretadas. La incorporación de dicho estilo no interfiere en la lógica del compositor, ni requiere alguna consideración especial.

La Figura 29 muestra la «interfaz concreta funcional» obtenida para el caso uno. A la izquierda se muestra al momento de terminar de ser cargada por el cliente, a la derecha se muestra la modificación que sufre cuando el valor del «parámetro contextual» «`Entorno/Actividades/MedicinaMatutina`» obtiene el valor «`verdadero`».

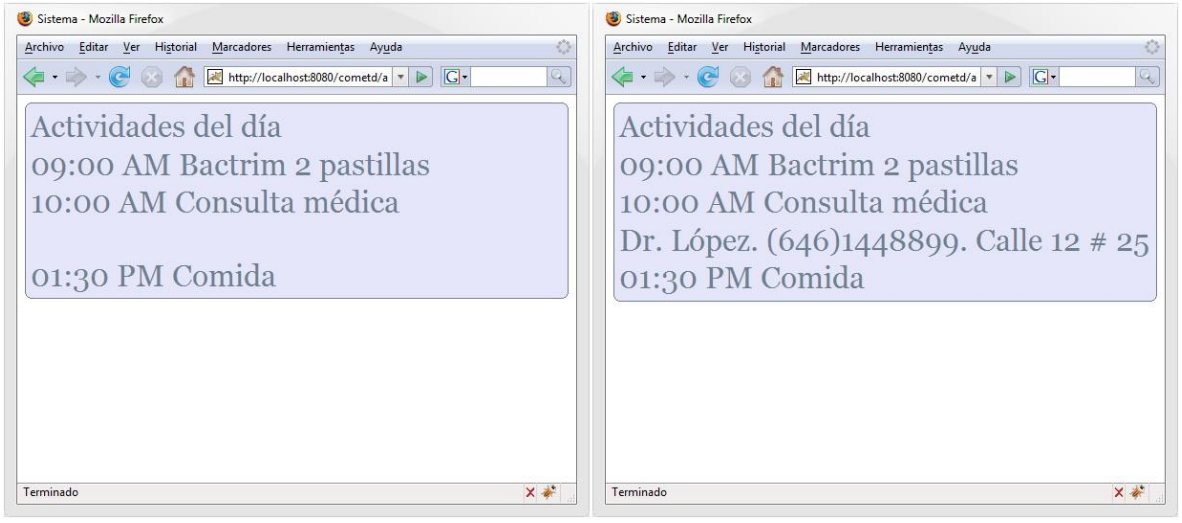

Figura 29. ***«Interfaz concreta funcional» para el caso uno del escenario.***

La Figura 30 muestra la «interfaz concreta funcional» obtenida para el caso dos. A la izquierda se muestra al momento de terminar de ser cargada por el cliente, a la derecha se

muestra la modificación que sufre cuando el valor del «parámetro contextual» «`Entorno/Compania/Estado`» obtiene el valor «`noSolo`».

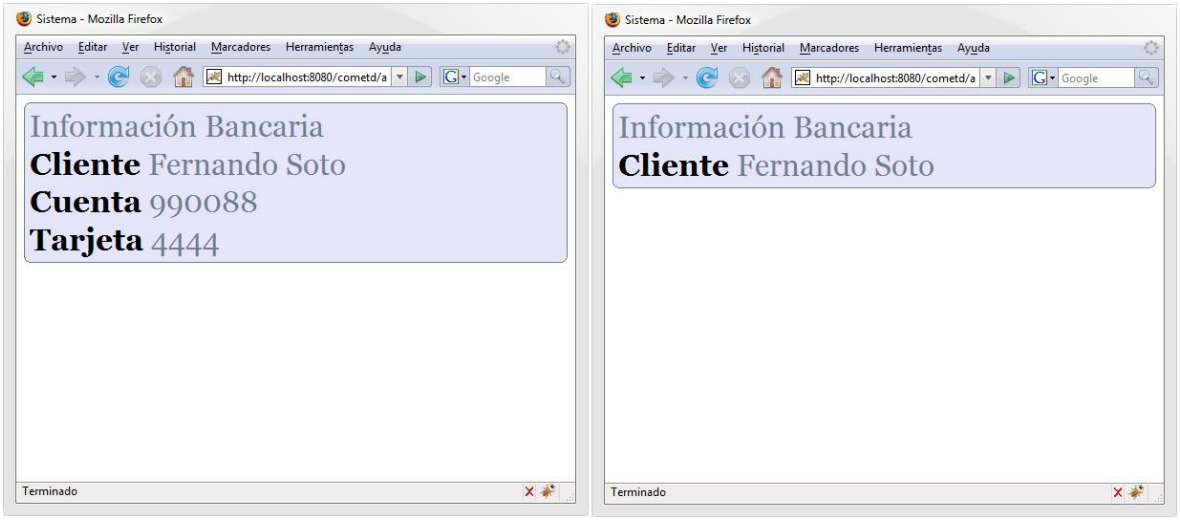

Figura 30. ***«Interfaz concreta funcional» para el caso dos del escenario.***

Para el caso uno del escenario se realiza una modificación al «modelo de elementos». Se agrega una entidad relacionada con la imagen de un frasco de medicinas. Además, esta nueva entidad y la entidad de «medicina matutina» definen una adaptación de tipo `defineRelevante`, para lo cual se define una nueva «regla de adaptación» genérica la cual indica que dejan de ser relevantes cuando el valor del «parámetro contextual» «`Entorno/Actividades/MedicinaMatutina`» sea «`verdadero`». El «modelo de elementos» está formado de seis entidades las cuales se describen en la Tabla XIII. Las entidades de interés son: la información de la «medicina», los «detalles de la consulta médica» y la «imagen del frasco de medicinas», el documento *XML* que contiene la «instancia de datos» se llama actividades.xml, y su identificador es `actividades`. La diferencia en la forma de reaccionar de cada uno de los elementos cuyo tipo de adaptación es `defineRelevante` se define en las «reglas de adaptación».

Tabla XIII. Entidades del «modelo de elementos» para la modificación del primer caso del escenario 1.

Entidad	**Tipo**	**Referencia**	**Adaptación**
Servicio	texto	instance('actividades')/servicio	N/A
Medicina	texto	instance('actividades')/medicina	defineRelevante
Consulta	texto	instance('actividades')/consulta	N/A
Detalles Consulta	texto	instance('actividades')/consultad	defineRelevante
Comida	texto	instance('actividades')/comida	N/A

Imagen medicina	imagen	instance('actividades')/medicinaimg	defineRelevante

Las extensiones están en el Apéndice K. La Figura 31 muestra la interfaz al terminar de ser cargada (izquierda) y cuando el «parámetro contextual» obtiene el valor «`verdadero`» (derecha). El objetivo es que la información de la «medicina matutina» y la imagen se oculten para que sólo quede la nueva información de los «detalles de la consulta médica» y aquella aún vigente.

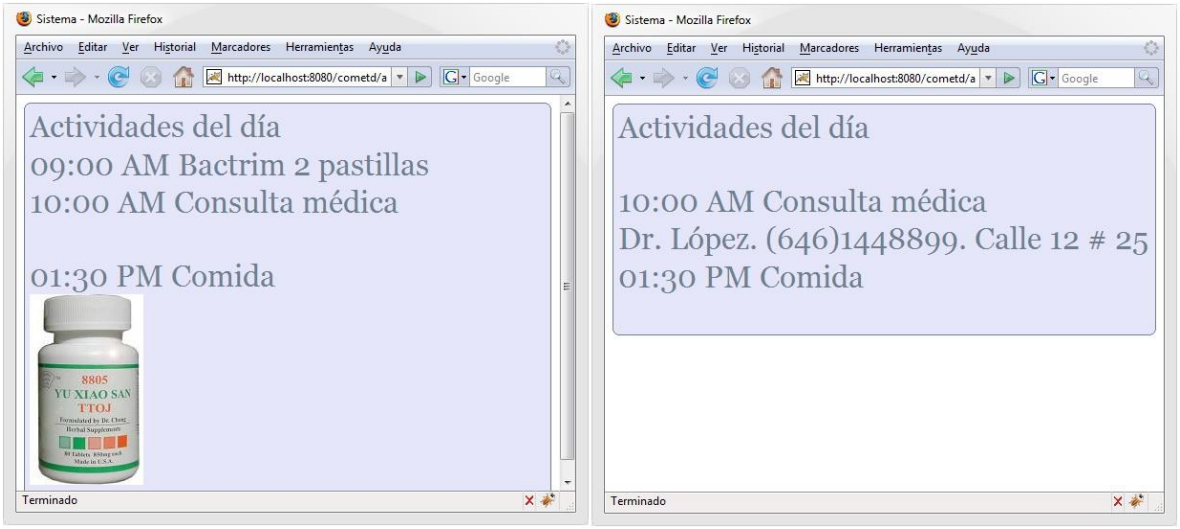

Figura 31. ***«Interfaz concreta funcional» para el caso uno extendido del escenario.***

V.2.7 Discusión

En el primer caso del escenario la propiedad `relevant` actúa a nivel del modelo de datos, de manera que define una restricción al «dato de instancia» al que se asocia, por lo tal, hay que definir una expresión *XPath* de la cual depende que dicho dato se considere o no relevante a la forma.

En la implementación del intérprete utilizado para el escenario, cuando el resultado de la evaluación de la expresión *XPath* de la propiedad `relevant` es `verdadero`, todos los controles vinculados al «dato de instancia» correspondiente se muestran, en otro caso, cuando es `falso` se ocultan. En el caso de una forma web, aquellos datos a los que se les agrega la propiedad `relevant` y la evaluación de su expresión es `falso`, no son enviados cuando se realiza el envió de datos al servidor.

En el segundo caso el uso de los elementos `toggle/switch/case` proveen un mecanismo que permite presentar secciones de la interfaz selectivamente, los cuales actúan a nivel de la estructura de la interfaz. Como se menciona en el Capítulo IV, la privacidad contemplada es únicamente si cierta información puede ser desplegada o no en la interfaz en función de un «contexto» cambiante.

Como se plantea en el Capítulo IV, en la estrategia de adaptación se omite enviar datos al cliente para realizar cambios a la «instancia de datos», para no entrar en conflicto con el procesamiento de *XForms* sobre los datos y los controles. La actualización a los datos de la instancia se maneja con el tipo de adaptación `actualizaDatos`, la cual recarga la «instancia de datos». Se contempla que la «instancia de datos» sea externa, de esta manera un servidor puede estar actualizando la instancia y en un momento dado el cliente puede recargar dicha instancia y actualizarse con los nuevos valores. De este modo la solución dada queda totalmente en términos de *XForms*, por lo tal cualquier «intérprete *XForms*» puede utilizar el documento generado tomando en cuenta que habrá que definir un lenguaje anfitrión correspondiente, así como utilizar el intérprete adecuado.

Otra consideración de la estrategia es que se opta por enviar eventos para disparar las acciones, sin embargo, no existe una acción para disparar el elemento `relevant`, ya que este elemento es una propiedad que se aplica a algún dato en la «instancia de datos». La estrategia seguida es definir una «instancia de datos» local que contiene solo un dato, cuyo valor se utiliza como la condición booleana al que hace referencia la expresión *XPath* de la propiedad `relevant`. Para modificar el valor de la instancia se elige definir una acción `setvalue` a la cual sí es posible enlazar directamente a través de un evento. Dicha acción únicamente cambia el valor del dato de la instancia local de falso a verdadero o viceversa.

La generación del «documento *XForms*» define instancias, eventos y acciones relacionados de manera dinámica conforme se requiera según el tipo de adaptación deseado.

Debido al tipo de adaptación que presenta la interfaz, el enfoque de «adaptación en el lado del cliente», resulta más adecuado en términos de: el tiempo de respuesta y la cantidad de datos que serían enviados de nuevo al cliente dada la generación de una nueva interfaz. Esto es, contemplando que el «servidor del contexto» es el responsable de realizar toda la tarea pertinente a la administración del «contexto».

V.3 Escenario 2

Descripción: se presenta un escenario en el cual se desea dar a conocer información del estado de un paciente en múltiples «pantallas sensibles», las cuales son dispositivos conscientes de su entorno y su objetivo es proveer información constante sobre el «contexto» de manera sutil, periférica y expresiva que no interfiera con la actividad primaria realizada (Favela *et al*., 09). En base a la información contextual: (1) se selecciona el dispositivo en donde se mostrará la información ambiental, (2) se genera la «interfaz concreta funcional» para el dispositivo seleccionado y, (3) finalmente se adapta la información que se muestra en el dispositivo objetivo. La «interfaz concreta funcional» se genera a partir de una «interfaz abstracta» y genérica descrita en *XForms*.

Objetivo del escenario: el objetivo de este escenario es implementar a cierto nivel el «modelo de procesamiento *XForms*» el cual especifica cómo unir cada una de las partes que conforman *XForms* («modelo *XForms*», «controles de la forma», etc.). Además, verificar si *XForms* permite definir toda la información necesaria para que un «intérprete» pueda traducir «la interfaz abstracta» a su plataforma específica, o qué elementos provee que puedan ser interpretados para dicha tarea. Esto con el propósito de ilustrar el tipo de interpretación realizada a los elementos *XForms* para un dispositivo de cómputo no convencional.

Escenario:

El escenario se sitúa dentro de un ambiente hospitalario, en específico en apoyo a la labor de las enfermeras. Debido a las distintas actividades que realizan las enfermeras que atienden a los pacientes, con frecuencia desconocen de eventos relevantes como puede ser que el suero se les ha terminado o la bolsa de orina se les ha llenado. En este ambiente la introducción de un sistema capaz de entregar información constante sobre el estado de los pacientes, ayudaría a las enfermeras a estar conscientes del estado de sus pacientes sin la necesidad de ir personalmente a revisarlos. Un requisito es que dicha información les sea entregada sin necesidad que ellas realicen una consulta explicita o estén conscientes de tener que realizar dicha consulta. Aquí las «pantallas ambientales» resultan una alternativa para cumplir dicho requisito. El *FlowerBlink* (Segura *et al*., 08) es un dispositivo de este

tipo, el cual es un pequeño florero artificial cuyas flores tienen en el centro *leds* de colores, que proveen información relevante. El *FlowerBlink* puede ser situado en algún lugar estratégico que se encuentre a la vista pero que no sea intrusivo, la acción de encender sus luces indicará un suceso relacionado al ambiente, como por ejemplo que a un paciente se le ha llenado la bolsa de orina. El objetivo es realizar una notificación de manera sutil indicando que algo ha ocurrido, esto hará que una enfermera tome consciencia de que algo está pasando con alguno de sus pacientes, ya sea que se le ha acabado el suero o la bolsa de orina se ha llenado. Aquí la información es entregada sin necesidad que la enfermera realize una consulta consciente y explicita. Si se desea información más detallada sobre qué paciente es o el cuarto correspondiente, se tendría que hacer uso de otro dispositivo de manera explícita.

Suponga que el *FlowerBlink* es utilizado en un hospital para indicar a las enfermeras que la bolsa de orina de un paciente está a punto de llenarse, *Rita es una enfermera que utiliza el FlowerBlink para conocer el estado de la bolsa de orina de Pedro. Un sensor en la bolsa de orina detecta la presencia de una nueva evacuación, con lo cual el liquido ha rebasado cierto límite considerado como crítico. Justo en ese momento Rita se acerca al vestíbulo de las enfermeras. Por lo tal, un «sistema consciente del contexto», decide notificarle de este evento a través del FlowerBlink localizado en esa área. La descripción abstracta de la información es enviada a un agente que actúa como proxy para dicho dispositivo. El agente utiliza un intérprete adecuado para transformar la «interfaz abstracta» en una «interfaz concreta» de la cual se obtiene el estado del FlowerBlink, es decir, si encenderá sus luces o no. Rita nota que el florero se encuentra parpadeando y se da cuenta que la notificación se relaciona con uno de sus pacientes. De modo que decide consultar su teléfono celular, para conocer la información detallada. Por medio de éste, se entera que Pedro ha orinado por tercera vez esta mañana y por lo tal su bolsa de orina pronto tendrá que ser remplazada. La interfaz en el FlowerBlink apaga las luces cuando su agente proxy se hace consciente de que Rita ha consultado la información en su teléfono celular. La Figura 32 resume gráficamente el escenario planteado.*

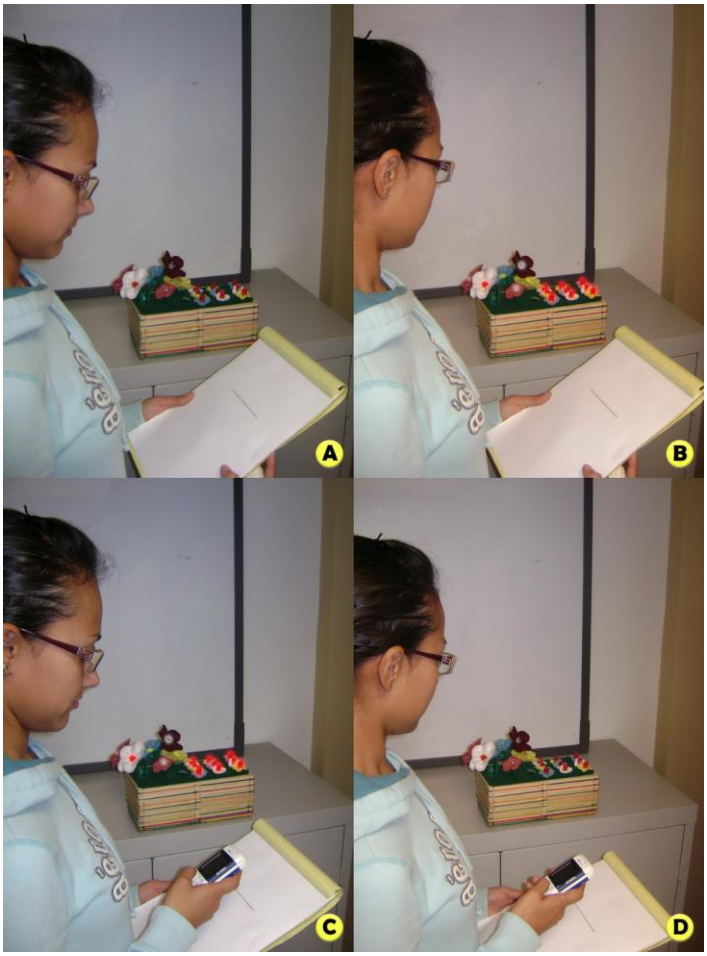

Figura 32. ***Presentación de información del paciente en distintos dispositivos.***

Aquí la información contextual se utiliza en el sistema en dos etapas. Primero, la información de la ubicación se utiliza para decidir qué «dispositivo sensible» utilizar. Ya que Rita se encuentra en el vestíbulo de las enfermeras, se elige el *FlowerBlink* para hacer la notificación. Una vez que la «interfaz concreta» es ejecutada en el dispositivo, se adaptada cuando se informa al «agente *proxy*» que el usuario se ha hecho consciente de la información, En el ejemplo cuando el «agente *proxy*» es notificado que Rita está consciente del estado del paciente, adapta su interfaz apagando las luces del florero. La Figura 33 da ejemplo del *FlowerBlink* situado en el vestíbulo de enfermeras de una manera no intrusiva. Originalmente consta de dos tipos de flores unas con tallo (Flores de alerta) y otras sin tallo (Flores de información). Las primeras se utilizan para llamar la atención de la enfermera cuando parpadean, las segundas están dispersas en una matriz de 3 x 3 con lo cual se puede

hacer un tipo de relación entre una flor sin tallo y un paciente, de manera, que si una de estas flores está encendida, indique el paciente para el cual se notifica la alerta. En el caso del escenario se utiliza todo el conjunto de flores para realizar la alerta.

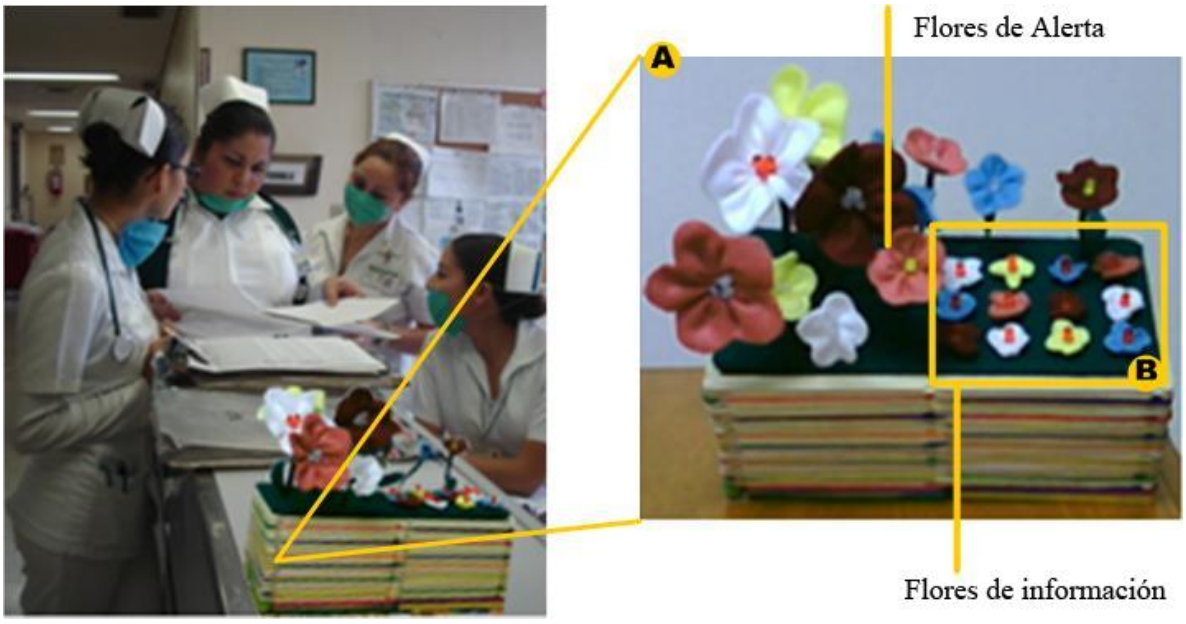

Figura 33. ***El FlowerBlink ubicado en el vestíbulo de las enfermeras.***

V.3.1 Implementación

V.3.1.1 Diseño de Arquitectura

Las Figuras 34 y 35 muestran el diseño a través de diagramas de secuencia que ilustran el flujo de llamadas entre los componentes que conforman la arquitectura. En los diagramas se definen los principales componentes de la arquitectura, la cual se describe a continuación. En la arquitectura, un «servidor» comunica a los distintos «dispositivos sensibles», por medio de sus «agentes *proxy*», los cambios en el ambiente notificados por el «gestor de sensado». El «gestor de sensado» se encuentra conectado a un conjunto de sensores que actualizan la información sobre el estado de las bolsas de orina de los pacientes. Cuando una notificación se va a enviar, el «servidor» solicita al compositor generar la «interfaz abstracta» la cual define la información del estado del paciente (paciente, enfermedad, hora, nivel de la bolsa de orina, etc.). Dicha interfaz es traducida para cada dispositivo dependiendo de sus recursos. El «servidor» decide notificar a la enfermera utilizando el

FlowerBlink debido que se encuentra en el vestíbulo de las enfermeras, de tal manera que la «interfaz abstracta» es enviada el «agente *proxy*» del *FlowerBlink*. Este agente actualiza el estado del *FlowerBlink* en respuesta a la traducción hecha por su «intérprete». En este caso las luces se encienden, para indicar que la bolsa de orina está casi llena.

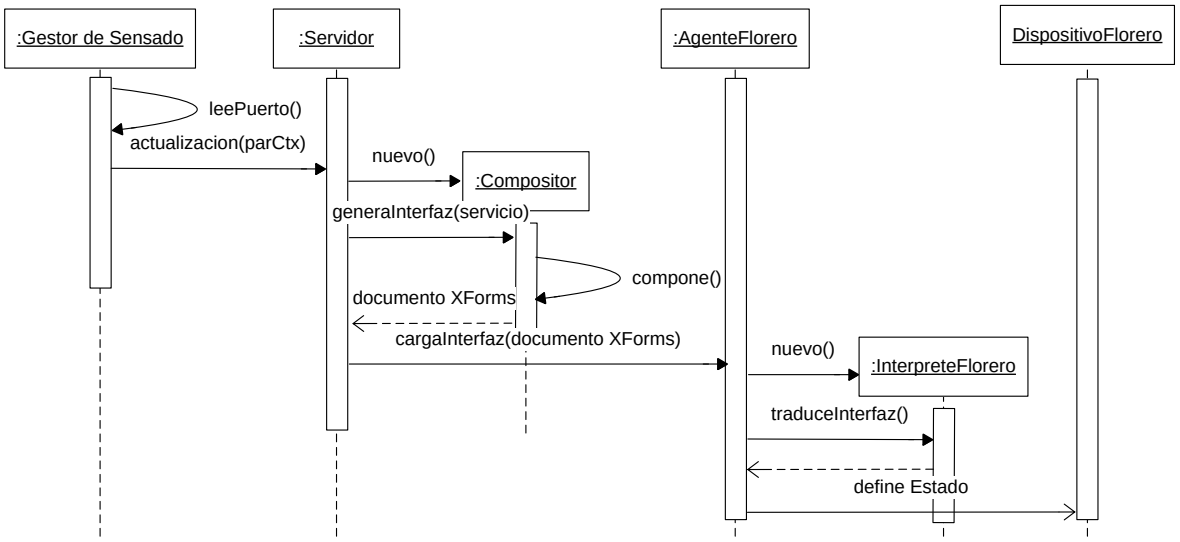

Figura 34. ***Diagrama de secuencia que muestra la notificación realizada a través del FlowerBlink.***

Cuando la enfermera nota que el *FlowerBlink* está parpadeando, decide consultar su teléfono celular para conocer información detallada. El «agente proxy» del teléfono celular solicita la información al «servidor», la cual le es entregada utilizando la misma «interfaz abstracta» utilizada para comunicar la información al *FlowerBlink*. El «agente proxy» del teléfono celular invoca a su propio «intérprete» para traducirla a sus capacidades. De esta manera el agente puede presentar la información a través de texto e imágenes. Dicha consulta actúa como un indicador de que Rita ha tomado consciencia del estado del paciente, de manera que el *FlowerBlink* puede ser apagado en respuesta. Dicha notificación es responsabilidad del «servidor», la cual es recibida por el «agente proxy» y traducida por su «intérprete», el cual sabe qué acción realizar, en este caso apagar las luces del florero.

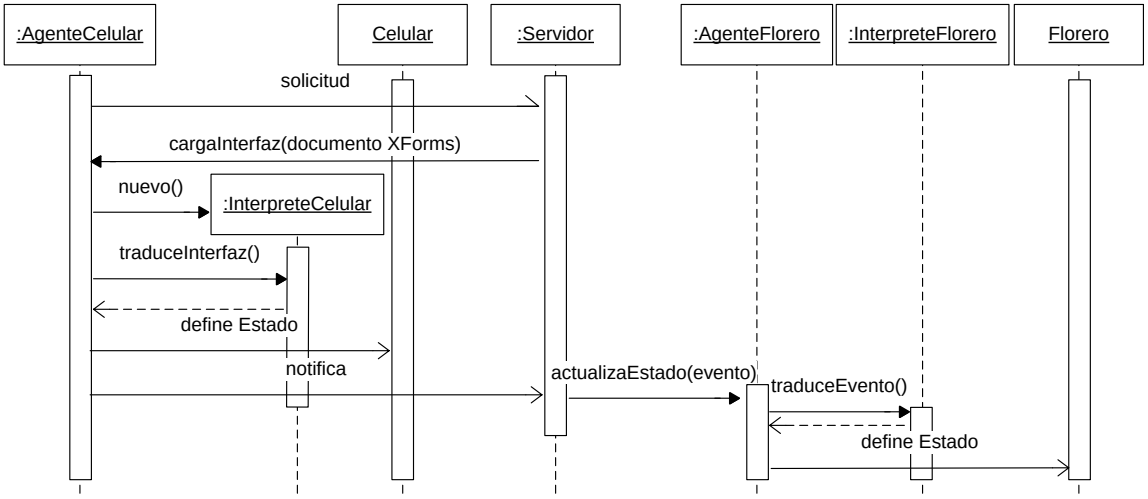

Figura 35. ***Diagrama de secuencia que muestra cómo el FlowerBlink es apagado una vez que la enfermera ha tomado consciencia del estado del paciente.***

Como se ha mencionado en los objetivos del escenario la idea es implementar el «modelo de procesamiento *XForms*» de modo que se procese un «documento *XForms*» y se interprete para un dispositivo no convencional como lo sería el *FlowerBlink*. Para esto se define un «*agente proxy*» cuya tarea es implementar la «interfase de programación» para comunicarse con el *FlowerBlink*, además se implementa un «intérprete» específico a los recursos del *FlowerBlink* el cual es el encargado de implementar el «modelo de procesamiento *XForms*». Las acciones pertinentes dictadas por dicho «intérprete» son enviadas al «agente proxy» el cual se comunica de manera real con el *FlowerBlink*.

En este escenario el proceso de generación del «documento *XForms*» no es el objetivo primario sin embargo se bosquejan las consideraciones necesarias que se requieren realizar a los elementos de adaptación para cumplir con los requisitos de este escenario. No obstante, el uso e interpretación del «documento *XForms*» como medio para transportar la información de manera genérica y abstracta a distintos dispositivos es muy relevante.

No se realiza la implementación real ni la comunicación con un *teléfono celular*. Para esta parte se define un «agente proxy» con su «intérprete» correspondiente, suponiendo que el dispositivo para el cual se utilizarían no presenta restricciones como las que presenta el

FlowerBlink. Esto es únicamente para implementar el «modelo de procesamiento *XForms*» y las consideraciones que se hacen sobre la «interfaz abstracta».

V.3.1.2 Implementación de Arquitectura

El *FlowerBlink* es un dispositivo real construido a partir de un *toolkit* de *phigets*[6] (Greenberg *et al*., 01). Para dicho *toolkit* existen *librerías*[7] en distintos lenguajes de programación que implementan una «interfase de programación» que permiten comunicar a una aplicación de *software* con el *phidget* correspondiente. La Figura 36 muestra un conjunto de *phidgets* a modo de ejemplo. El *PhidgetInterfaceKit* 8/8/8 contiene 8 entradas digitales, 8 entradas analógicas y 8 salidas digitales. Estas 8 salidas digitales pueden ser utilizadas para controlar *leds*. El PhidgetInterfaceKit 0/0/4 permite realizar una interfase con dispositivos como motores. La última ilustración a la derecha muestra un servo motor y el *PhidgetServo Kit 1-Motor* el cual permite controlarlo.

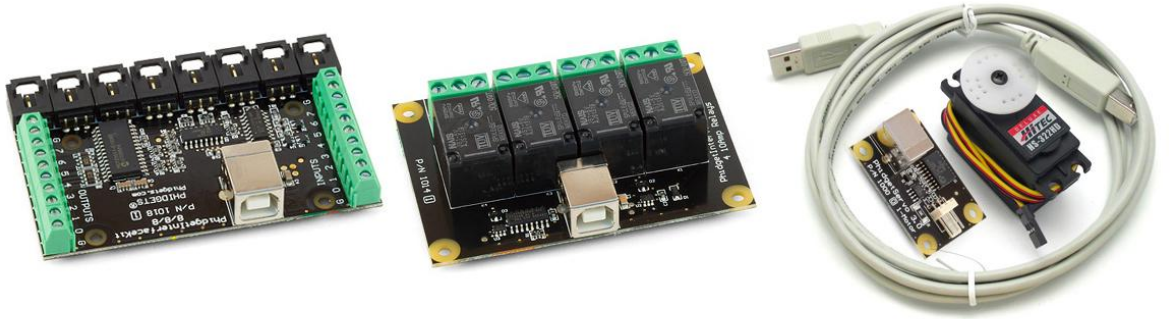

Figura 36. ***Ejemplo de phidgets.***

En este escenario se utiliza el *FlowerBlink* al cual se le implementa un «agente proxy» que se encarga de la comunicación con éste. La implementación se realiza en el lenguaje de programación Java y el conjunto de librerías correspondiente a dicho lenguaje. Cada uno de los «agentes *proxy*» se implementa como un «escuchador de eventos» los cuales se suscriben al «servidor de contexto», que se implementa como un «notificador de eventos». Para dicha tarea, el «servidor de contexto» simula los cambios en el «contexto», estos

[6] http://www.phidgets.com/index.php

[7] http://www.phidgets.com/downloads.php?os_id=1

cambios son notificados en forma de evento a los escuchadores («agentes *proxy*») los cuales pasan a sus «intérpretes» correspondientes el evento notificado, de este modo cada intérprete sabe qué «acción» ejecutar.

Cada evento notificado está compuesto de un identificador, el «parámetro contextual», y el nuevo valor de dicho parámetro. Sólo el identificador es utilizado para disparar la «acción» correspondiente definida en la «interfaz abstracta», siguiendo la misma estrategia establecida por los elementos de adaptación.

V.3.1.3 Consideraciones de Diseño e Implementación

Una consideración de diseño impuesta para este escenario es utilizar la misma «interfaz abstracta» en cada dispositivo para entregar la información del paciente (paciente, enfermedad, hora, número de veces que ha orinado y el nivel de la bolsa de orina). En el teléfono celular, debido a sus capacidades, es posible consultar toda esta información de manera explícita, por otro lado, en el *FlowerBlink* debido a su naturaleza y propósito, ser una «pantalla ambiental», denotará a través del parpadeo de las luces que el nivel de la bolsa de orina ha rebasado cierto límite.

Sin contemplar el requisito de una única «interfaz abstracta» para todos los dispositivos, la estrategia para este escenario se basaría en la especificación de una «interfaz abstracta» para cada dispositivo, que contenga sólo la información que utilizaría el dispositivo. Para esto, en el lado del servidor un nuevo tipo de adaptación incluiría la selección del «modelo de datos» y el conjunto de «controles» relacionados a dicho modelo. Por ejemplo, para el *FlowerBlink*, su «instancia de datos» únicamente incluiría el dato del nivel de la bolsa de orina. Este tipo de decisiones se definirían en las «reglas de adaptación». Sin embargo, al contemplar la generación de distintas «interfaces abstractas» se pierde el objetivo de utilizar un *UIDL*, que permite describir una interfaz de manera abstracta y genérica que debería poder ser interpretada por cualquier dispositivo.

Entonces, el requisito contemplado involucra la especificación de información adicional a la «interfaz abstracta» que permita al «intérprete *XForms*» tomar las decisiones en base al «documento *XForms*», para realizar la interpretación adecuada. El «intérprete» es consciente de su plataforma de cómputo, por lo tal conoce el tipo de «interfaz concreta

funcional» que debe generar así como las interpretaciones que debe realizar. Trewin indica que un *UIDL* idealmente debería proveer toda la información necesaria de un servicio específico para construir la «interfaz de usuario» (Trewin *et al.*, 03). La información adicional no debe suponer capacidades del dispositivo, ya que el *UIDL* escapa de toda suposición que respecte a la plataforma de cómputo. *XForms* no contempla elementos específicos debido a su soporte a la independencia de dispositivos. El objetivo es analizar qué elementos y propiedades provee *XForms* de modo que puedan interpretarse para seleccionar la información necesaria.

Aquí la decisión es utilizar la propiedad *navindex*, la cual en una página web define el orden de navegación sobre los controles. Este elemento se define como un atributo del control, su valor se obtendrá de la prioridad especificada en el «modelo de elementos». En la estrategia de adaptación dicha prioridad se utiliza para la selección de elementos a incluir, si así lo específica alguna «regla de adaptación». En este escenario no se realiza tal selección por el requisito dado. Utilizar la prioridad para hacer una selección, en cierta medida indica la importancia del elemento dentro de la interfaz, por lo tal, indicar el atributo *navindex* infiere únicamente la prioridad que toman los controles en la «interfaz concreta funcional». La interpretación que se realiza sobre dicho atributo es elegir el «control de la forma», relacionado al dato de instancia de interés, con el que trabajará el «intérprete» del *FlowerBlink*. En lo que respecta a la extensión requerida, es necesario codificar este tipo de decisión en el «compositor *XForms*».

Otra consideración se relaciona a la interpretación de la información realizada por el «intérprete» del *FlowerBlink*, una vez que se decide qué control utilizar en base a la propiedad *navindex*, en este caso el nivel de la bolsa de orina. Hay que tener en cuenta que el *FlowerBlink* únicamente parpadeará cuando el nivel de orina haya rebasado por ejemplo los 600 ml. Para este caso se establecen tres propuestas:

1) El «intérprete» puede ser implementado de modo que, conociendo el objetivo y alcance del *FlowerBlink* respecto a qué tipo de información provee (información básica en forma de alerta), realice una comparación, predefinida a nivel de código, del dato del nivel de la bolsa de orina y dado que rebase el límite definido indique a su «agente proxy» el estado a definir al *FlowerBlink* (encender o apagar). Esta estrategia no hace

uso de las especificaciones del «documento *XForms*» únicamente toma la información de la «instancia de datos», además es una solución muy específica con poca flexibilidad para dar mantenimiento, por ejemplo, en el caso que se desee restablecer el límite definido para alertar sobre la bolsa de orina, suponiendo que en el hospital se instalen nuevos contenedores con mayor o menor capacidad.

2) El segundo caso contempla utilizar la estrategia de la propiedad `relevant` afectando al dato de la bolsa de orina, tal como se utiliza en el escenario uno, en el cual la expresión *XPath* de la propiedad `relevant` se asocia al valor de un dato de instancia que se modifica a través de la acción `setvalue`. En el caso de este escenario, si la expresión *XPath* se hace verdadera puede indicar que el *FlowerBlink* empiece a parpadear. Aquí el servidor monitorearía la cantidad de orina y dado que rebase la cantidad especificada se enviaría al cliente el evento para disparar la acción `setvalue`. Esta estrategia deja todo el trabajo al servidor, y en el cliente solo se haría la evaluación para saber si el dato de instancia ya es relevante. Sin embargo, tomando en cuenta que el *FlowerBlink* sólo tiene un tipo de salida, y que ésta es a través del parpadeo de las luces, el intérprete no hace uso de la información de la instancia de datos, sino que únicamente espera a que la expresión *XPath* se haga verdadera para indicar que el *FlowerBlink* empiece a parpadear. En el escenario uno, este tipo de estrategia si resulta conveniente, ya que más de un dato de instancia se puede declarar como relevante y cada uno trae consigo información diferente la cual es mostrada o no, pero en el caso del *FlowerBlink* la única salida que se tiene es el parpadeo de luces. Lo que hace cuestionarse sobre la verdadera necesidad de un «agente proxy» y un intérprete para realizar esto.
3) El tercer caso contempla que a partir de las reglas de adaptación se construyan expresiones en *XPath* relacionadas al valor del nivel de orina en la «instancia de datos», es decir, que en el cliente se realice la comparación del nivel actual de la bolsa de orina y el límite establecido. Por ejemplo, si en el registro a monitorear de la «regla de adaptación» dinámica, indica que se notifique el evento relacionado cuando el «parámetro contextual» del nivel de bolsa de orina sea mayor a 600 ml, esta condición contextual se puede traducir a la siguiente condición en *XPath*:
   “`instance(paciente)/nivelOrina[.>600]`”

En esta situación el intérprete utiliza la información que la instancia de datos provee, utiliza los demás elementos del «documento *XForrms*» y obtiene las condiciones a partir de las reglas de adaptación lo que lo hace flexible de mantener.

La expresión *XPath* debe estar asociada a alguna «propiedad *XForms*». Si se utiliza la propiedad `relevant`, al consultar en el teléfono celular el estado del paciente, si la condición no se cumple, el dato del nivel de orina podría ser omitido, esto depende de la implementación del intérprete. La expresión «el dato de instancia es relevante», se utiliza para indicar que la expresión condicional *XPath* asignada a ese dato se cumple. Se elige utilizar la propiedad `required` la cual en una forma web se usa para indicar que un «control de entrada de datos» debe ser llenado para que la forma sea válida, en dicho caso la expresión *XPath* condicionaría que el valor no debe ser vacío. En el caso del teléfono celular no afecta ya que son controles de salida y el control relacionado al dato de instancia siempre es mostrado aunque no cumpla con la condición. En el caso del *FlowerBlink* sólo se tienen salidas, por lo tal en el «intérprete» se implementa de modo que cuando la condición se cumpla el *FlowerBlink* empiece a parpadear, indicando que el nivel de orina a rebasado el nivel crítico. Aquí también es necesario indicar que el servidor debe notificar que existen cambios en el ambiente y es necesario actualizar la instancia de datos.

Este caso, sin embargo, como se discute en el capítulo IV, requiere que se actualice la instancia de datos del cliente, y una vez actualizada el «intérprete» debe actualizar todas las entidades relacionadas que se vean afectadas. La actualización de la instancia de datos se puede realizar a través de dos formas:

a. La primera contempla a una entidad ajena a *XForms* que manipule los «datos de instancia», en este caso es necesario también que realice las actualizaciones pertinentes en los elementos afectados, lo que puede ocasionar inconsistencias en el estado de la interfaz.
b. La segunda forma implica el uso de la acción `send` en conjunto con el elemento `submission`. En el ambiente web, estos indican el envío de datos al servidor, sin embargo *XForms* provee los mecanismo para indicar qué acción realizar después de enviar los datos, en las formas web tradicionales queda a

consideración del servidor pertinente, en cambio *XForms* permite especificar qué hacer, un camino es justamente actualizar la «instancia de datos». En base a la plataforma de trabajo del *FlowerBlink*, el cual es un dispositivo que se conecta directamente a través de un cable *USB* a un equipo de cómputo, el trabajo es totalmente local, la interpretación realizada es únicamente sobre la recarga de datos, el envío de datos se omite ya que no se utiliza en un ambiente web. En base a la plataforma de cómputo es válido realizar la interpretación a conveniencia de ésta.

Para la acción de apagar el florero, se hace la suposición que una vez que la enfermera se ha hecho consciente, cambia la bolsa. Por lo tal, el dato de instancia se puede poner a cero, indicando que la bolsa está vacía. Sin embargo, si la bolsa no se cambia, cuando haya una nueva presencia de líquidos, el servidor notificará al cliente con la información real el cual parpadeará una vez más.

Esta tercera estrategia, también requiere que el servidor esté actualizando la instancia externa la cual se utilizará para actualizar los datos de la instancia del cliente.

De los tres casos contemplados se selecciona el tercero debido a las consideraciones mencionadas. La cual requiere extensiones en el «modelo de elementos», las «reglas de adaptación» y el «compositor *XForms*».

A continuación, en las secciones V.3.2, V.3.3 y V.3.4 se describen los dos modelos y las reglas de adaptación utilizados en la «primera etapa de adaptación» de la interfaz la cual contempla la construcción del «documento *XForms*» y la generación de registros de eventos a monitorear en la «tercera etapa de adaptación». Además se describen las consideraciones de extensión realizadas a los modelos y reglas.

V.3.2 Modelo del Contexto

El «modelo de contexto» definido en este escenario contempla dos «parámetros contextuales» de interés, el primero es la ubicación de la enfermera, ya que permite seleccionar un dispositivo para notificarla del estado del paciente y el segundo, es el hecho de que la enfermera ha tomado consciencia del estado del paciente. Este último «parámetro contextual» es necesario traducirlo a un nivel técnico que permita plasmarlo en la

descripción del contexto. Hay dos posibilidades para manejarlo, ya sea como una actividad de la enfermera o como un estado del teléfono celular que indica si realiza la consulta de información detallada. Independientemente de cómo sea descrito en el «modelo del contexto», cuando el «agente proxy» del teléfono celular solicite la información al «servidor», se utilizará como el evento que indica que la enfermera ha tomado consciencia del estado del paciente. Esta situación no es única en el día, ya que se puede presentar varias veces, aquí el «servidor» debe manejar el evento en conjunto con la hora en que éste se genera y un rango de tiempo considerable o utilizar el número de veces que ha orinado, para distinguir entre el mismo evento en distintos momentos del día. El «perfil» de interés es el del entorno, su documento esquema se puede consultar en el Apéndice L y el «perfil» en el Apéndice M. Aquí la ubicación, actividades y cada paciente a su cargo se definen como componentes.

La Tabla XIV resume los «parámetros contextuales» para el escenario dos.

Tabla XIV. Resumen de los «parámetros contextuales» vinculados a entidades del «modelo de elementos».

Tipo de Contexto	**Componente**	**Atributo**	**Rango de valores**	**Tipo Atributo**
Entorno	Ubicación	Física	Lugares del hospital	Dinámico
Entorno	Actividades	AlertaBasicaAtendida	verdadero/falso	Dinámico
Entorno	Actividades	ActividadActual	Actividades varias.	Dinámico
Entorno	Paciente	IdPaciente	Valor alfanumérico	Estático
Entorno	Paciente	Enfermedad	Valor alfanumérico	Estático
Entorno	Paciente	NivelOrina	Valores reales	Dinámico
Entorno	Paciente	VecesEvacuacion	0,1,2,3…	Dinámico
Entorno	Paciente	NuevaEvacuacion	verdadero/falso	Dinámico

V.3.3 Modelo de Elementos

A partir de la estrategia definida la extensión del «modelo de elementos» contempla únicamente la definición de un nuevo tipo de adaptación denominada `defineCondicion`, la cual se agrega a la lista de adaptaciones ya contempladas. El compositor requiere codificar las nuevas consideraciones que dicha adaptación conlleva. Las extensiones y el «modelo de elementos» definidos pueden consultarse en el Apéndice N.

Dicha adaptación involucra el uso del elemento `required` al cual se le define una condición *XPath* que se obtiene de las reglas de adaptación. En conjunto se define la acción `send` con el elemento `submission` indicando los parámetros para que se lleve a cabo una

actualización de la «instancia de datos». Para el escenario, el «modelo de elementos» está formado de seis entidades las cuales se describen en la Tabla XV. La entidad de interés es aquella relacionada con la información del «nivel de orina», el documento *XML* que contiene la «instancia de datos» se llama paciente.xml, y su identificador es `paciente`.

Tabla XV. Entidades del «modelo de elementos» para el escenario 2.

Entidad	Tipo	Referencia	Adaptación	Prioridad
Paciente	texto	instance('paciente')/paciente	N/A	2
Enfermedad	texto	instance('paciente')/enfermedad	N/A	2
Hora aviso	texto	instance('paciente')/hora	N/A	2
Nivel orina	texto	instance('paciente')/cantidad	defineCondicion	1
Número de evacuaciones	texto	instance('paciente')/veces	N/A	2
Imagen	imagen	instance('paciente')/imagen	N/A	2

V.3.4 Reglas de Adaptación

La parte de las «reglas de adaptación» que requiere una extensión es la parte de acciones, ya que las reglas de tipo dinámicas contemplan únicamente la definición de eventos a monitorear. Por lo tal se define un elemento que captura la información condicional para construir la expresión *XPath* (p. ej. que el nivel de orina haya rebasado los 600 ml). Las extensiones y las reglas específicas pueden consultarse en el apéndice O.

La Tabla XVI resume la «regla de adaptación» de interés.

Tabla XVI. Descripción de las «reglas de adaptación» para el escenario 2.

Tipo	Condiciones	Acción
Dinámica	Entidad: `Adaptación = defineCondicion`	Construir: `Entorno/Paciente/NivelOrina > 600` Monitorear: `Entorno/Paciente/NuevaEvacuacion = verdadero`

V.3.5 Proceso de Generación

La Figura 37 muestra a modo de ejemplo las correspondencias entre los tres elementos de adaptación para el escenario. Aquí se utilizan las extensiones definidas. Es necesario que exista una consistencia sintáctica al momento de implementar cada modelo así como las reglas. Las nuevas consideraciones son implementadas por el «compositor *XForms*». En el

ejemplo de la Figura en particular, el compositor define en el «documento *XForms*» una expresión *XPath* relacionada al nivel de orina, así como la acción `send` y el elemento `submission` necesarios, además genera un registro para monitorear el valor del «parámetro contextual» especificado.

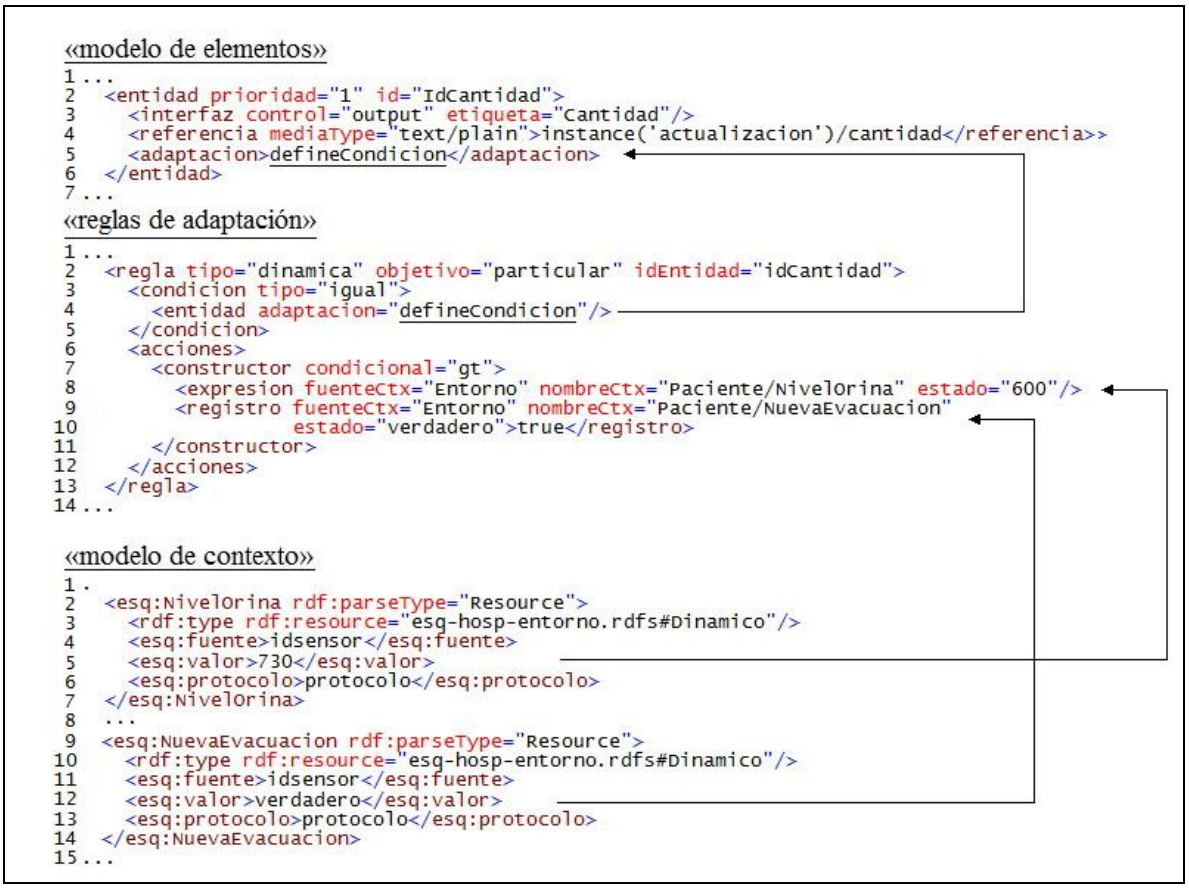

Figura 37. ***Correspondencias entre los elementos de adaptación.***

El documento *XForms* generado se envía al «agente *proxy*» del florero cuando es necesario notificar sobre una nueva evacuación, si el «dato de instancia» indica que se han rebasado los 600 ml, debe parpadear a modo de alerta. Cuando el «agente *proxy*» del teléfono celular solicita la información se le envía el mismo «documento *XForms*». Cada uno de los «agentes *proxy*» utilizan la información que tienen de sus plataformas para mostrar la información. En el caso del *FlowerBlink* ya se han descrito las interpretaciones que realiza sobre el «documento *XForms*», en el caso del teléfono celular, a pesar de no haberse implementado para el dispositivo real, se implementa una versión prototipo de un intérprete que no tiene impedimento en mostrar texto o imágenes, por lo tal, hace uso de la «interfase

de programación» dada por el lenguaje *Java* para generar este tipo de salidas. Ambos «agentes» implementan, al nivel de interés del escenario, el «modelo de procesamiento *XForms*» en el cual se especifica la relación entre los datos de instancia, los controles de la forma, y demás elementos que afectan a estos como acciones, eventos, etc.

V.3.6 Resultados

En este escenario como en el anterior el «servidor de contexto» se simula así como la infraestructura que da soporte al sensado e interpretación del «contexto».

La Figura 38 muestra el prototipo intérprete que lee el «documento *XForms*» que se le entrega, del cual obtiene la información para definir una salida o entrada, así como su información relacionada a un dato de instancia. La acción de desplegar la información detallada del paciente actúa como un disparador de un evento que notifica que la enfermera ha tomado consciencia de la información. Esta acción, como se ha comentado, infiere que el nivel de orina se pone a cero (se cambia la bolsa), por lo tal se notifica al «agente proxy» del *FlowerBlink* para que actualice su instancia de datos actual, con lo que deja de parpadear debido a que la condición de los 600 ml, ya no se cumple. El «servidor» mantiene una «instancia de datos» local que actualiza y es la que el «agente *proxy*» del *FlowerBlink* recarga.

Un escenario alternativo podría contemplar que la «interfaz abstracta» entrega información sobre el nivel de la bolsa de orina, el nivel de la bolsa de suero y el ritmo cardiaco. En el *FlowerBlink* se requerirá que se asocie un determinado patrón de parpadeado o utilizar selectivamente algunas flores para indicar qué información se está entregando a la enfermera. Sin embargo, realizar este tipo de planteamiento infiere darle a la persona cierta carga cognoscitiva al tener que realizar asociaciones e interpretaciones, de lo que el patrón de parpadeo quiere dar a conocer. En este caso utilizar un dispositivo como el teléfono celular es más adecuado ya que permite presentar información concreta, y por otro lado, el objetivo del *FlowerBlink* como «pantalla ambiental» se perdería.

Figura 38. ***Prototipo intérprete que puede manejar texto e imágenes.***

V.3.7 Discusión

Las extensiones definidas en este escenario, requieren que se actualice el «modelo de elementos», las «reglas de adaptación» y el «compositor *XForms*». En relación al «modelo del contexto» se requiere plasmar la información contextual relevante en forma de «componentes» y «atributos», es decir, la definición de un nuevo vocabulario, siguiendo la especificación de *CC/PP*.

En el caso de las extensiones es necesario conocer los elementos que la especificación *XForms* provee y cómo estos pueden ser utilizados para cumplir los requisitos impuestos por el escenario. Por lo mismo, esta decisión recae en el desarrollador, ya que puede utilizar otro elemento que considere más adecuado para las necesidades que pretende satisfacer.

Debido a que en este escenario se sigue con la estrategia de utilizar únicamente elementos *XForms* para definir la «interfaz abstracta», es necesario realizar las extensiones mencionadas. Esto es con la idea de que la solución siga generando un «documento *XForms*» apegado a la especificación *XForms*. Por lo mismo, cada elemento *XForms* a utilizar requiere ciertas consideraciones tanto en el cliente como en el servidor. Utilizar una instancia de los datos del «contexto» en el cliente requiere que el servidor notifique de actualizaciones, además involucra modelar las relaciones y condiciones utilizando *XPath.*

V.4 Conclusiones

Como puede observarse, la interpretación de un documento *XForms*, puede hacerse más allá que sólo para formas web, ya que sintetiza el patrón arquitectónico MVC. La especificación de *XForms* se puede definir en cuatro secciones con un bajo acoplamiento, el «modelo *XForms*», la «interfaz de usuario *XForms*» (controles de la forma), el «protocolo de envío *XForms*» y el «modelo de procesamiento *XForms*». Cada una de estas secciones puede utilizarse de manera independiente o en combinación con otras tecnologías (B'Far, 05). El «modelo de procesamiento *XForms*» indica cómo vincular cada uno de los elementos de manera significativa. Un «contenedor *XForms*», aquel que maneja un «documento *XForms*», debe implementar dicho «modelo de procesamiento».

Algo que no se puede negar es que las ideas que han dado nacimiento a *XForms* han surgido del ambiente web, en primera instancia al proveer una especificación que viniera a sobrellevar muchas de las deficiencias de las formas web que se utilizan, además de la dependencia que tienen del uso de lenguajes *script* para realizar tareas como validaciones. De aquí que cuando *XForms* se diseñó, un objetivo fue reunir muchas de las practicas comunes de programación que se hacían con *scripts* y plasmarlas de manera nativa en la especificación. *XForms* no define un único protocolo de uso, aunque a pesar de su propio origen está muy vinculado al protocolo *HTTP*, *XForms* puede utilizarse con otros. Una de las consideraciones de *XForms* es que no realiza suposición alguna en el lado del servidor, de hecho el único momento en el que *XForms* tiene contacto con la parte del servidor es a través del elemento para realizar el envío de datos, si es que es necesario.

Capítulo VI

Conclusiones, Aportaciones y Trabajo Futuro

VI.1.1 Conclusiones

Los «lenguajes declarativos de la interfaz de usuario» se utilizan como medio para representar la interfaz de usuario de manera genérica y abstracta, su traducción para un dispositivo de cómputo específico se basa en la información sobre las características de éste y posiblemente en las preferencias del usuario. Sin embargo, otra información contextual se puede utilizar para llevar a cabo distintas adaptaciones más allá de la adaptación para que sea adecuada al dispositivo particular. La propuesta de este trabajo contempla la incorporación de otro tipo de información contextual en distintas etapas relacionadas a su generación, interpretación y entrega e interacción con el usuario final, denominadas primera, segunda y tercera etapas de adaptación respectivamente. Para dicha tarea se selecciona *XForms* como el lenguaje base para definir la «interfaz abstracta» que se genera y traduce para entregar al cliente. Uno de los propósitos es verificar en qué medida dicho lenguaje da soporte de manera nativa o qué consideraciones se necesitan para involucrar información contextual en las distintas adaptaciones por las que pasa la «interfaz de usuario». Aunado a esto se imponen ciertos requerimientos de diseño los cuales se definen en base al análisis de la literatura. Por ejemplo, que la información contextual no se especifica dentro de la descripción abstracta y que debe ser manejada por una entidad externa que haga el papel de «servidor de contexto», otra consideración impuesta es, que en base al lenguaje seleccionado se provea una solución apegada a su especificación.

Los tipos de adaptaciones a los cuales se les da soporte en este momento se obtienen del bosquejo de posibles escenarios de aplicación. Una vez que se diseñan e implementan los elementos de la estrategia de adaptación que guían la adaptación de la interfaz («modelo de elementos», «modelo de contexto», «reglas de adaptación» y el «compositor *XForms*») se validan a través del desarrollo de escenarios de aplicación específicos, inspirados en los escenarios contemplados inicialmente. Aquí es posible ver cómo *XForms* proporciona el marco para poder definir una «interfaz abstracta» que puede ser interpretada para dispositivos como *PC*'s así como dispositivos de cómputo no convencionales (p. ej. *FlowerBlink*), en ambos casos es necesario que el «intérprete» correspondiente conozca su plataforma y en base a esto, interprete los elementos definidos en la «interfaz abstracta». En el caso del *FlowerBlink* es necesario hacer uso de otros elementos *XForms* y establecer consideraciones para dar a conocer la información de la «interfaz abstracta» que es relevante a éste, sin que se haga especulación sobre las características del dispositivo, para así conservar su propiedad genérica y abstracta. Sin embargo, a grandes rasgos, el tipo de interfaces generadas en este momento no presentan una gran complejidad ya que se limitan a definir elementos abstractos, vincularlos a datos de instancia y relacionarlos, si es el caso, a acciones *XForms* que provean un tipo de adaptación deseado para los elementos.

XForms no sólo provee la definición abstracta para generar la «interfaz concreta», sino que además permite definir la instancia de datos con la cual se interactúa. Lo que permite realizar adaptaciones tanto en la estructura de la interfaz como en los datos de información al utilizar las acciones definidas en la especificación *XForms*. Este tipo de adaptaciones corresponden a la «tercera etapa de adaptación» y deben ocurrir dado un cambio en el «contexto» cuando el usuario se encuentra interactuando con la «interfaz concreta». Este tipo de relación lógica (cambio en el contexto - adaptación en interfaz concreta) se realiza por medio de la definición de eventos genéricos, los cuales se contemplan en la especificación de *XML Events*. De este modo se logra la separación de las tareas de sensado e interpretación del «contexto» y el uso de dicha información.

Los elementos definidos para guiar la adaptación se apegan a las consideraciones de diseño impuestas por la propuesta de adaptación. El «modelo de elementos» permite capturar la información suficiente para poder seleccionar y definir elementos *XForms* dentro de la

«interfaz abstracta», la estructura que presenta es genérica para cualquier elemento, por lo cual la definición de los elementos únicamente contempla su definición, su relación a un dato de instancia y una posible adaptación.

En relación al modelado de la información contextual se utiliza una extensión a *CC/PP* de manera que permite reflejar las consideraciones hechas sobre el manejo del contexto, por ejemplo que sirva como un repositorio inicial para informar del estado del contexto en un momento dado, además de indicar «parámetros contextuales» cuyo valor puede variar en el tiempo. Este tipo de requisitos que van más allá del enfoque tradicional de *CC/PP* son implementados como parte de las consideraciones de diseño del «compositor *XForms*». De aquí tenemos que sí es posible capturar este tipo de información utilizando *CC/PP*, sin embargo debido a su especificación es necesario contemplar la información en términos de «componentes» y «atributos» lo que obliga al desarrollador a pensar y estructurar su información siguiendo este lineamiento, que posiblemente en otro escenario haga difícil la lectura y comprensión de la información. Otra consideración, que surge de la suposición de la existencia de un «servidor de contexto», es que la información que éste entrega y utiliza en el «modelo de contexto» es puntual y está en un formato legible para el desarrollador. Como se menciona en la parte del desarrollo de los escenarios de aplicación, en estos, los elementos contextuales definidos se limitan únicamente a aquellos de interés para la generación de la «interfaz abstracta», el sistema anfitrión puede utilizar otra información, pero ésta no se especifica con el objetivo de que los perfiles reflejen la información relevante para la generación de la «interfaz abstracta».

Las «reglas de adaptación», permiten definir condiciones sobre las características de las entidades del «modelo de elementos» y sobre los «parámetros contextuales» del «modelo del contexto», por lo tal, es necesario mantener una consistencia sintáctica entre los elementos de adaptación, ya que el análisis que realiza el «compositor *XForms*» se basa en los elementos sintácticos definidos en estos. Además las reglas sirven para definir los valores de los «parámetros contextuales» que se relacionan a las adaptaciones definidas en la «interfaz abstracta».

Por último el «compositor *XForms*» desarrollado para los escenarios de aplicación implementa las consideraciones necesarias para generar el «documento *XForms*» adecuado a la especificación dada por los modelos y las «reglas de adaptación».

La principal postura de la propuesta se centra en el trabajo en el lado del cliente, tanto la traducción de la interfaz, como las adaptaciones de la «tercera etapa de adaptación». Esto se decide dada la capacidad de *XForms* de proveer mecanismos para el trabajo en el lado del cliente y tomando en cuenta las ventajas que trae la adaptación realizada en el cliente, como por ejemplo, la reducción de llamadas al servidor. Sin embargo, una limitante del trabajo en el lado del cliente es la existencia de intérpretes, aunque esta limitante se ve en toda estrategia que utilice un lenguaje declarativo. Otra limitante es que hay clientes con una capacidad de procesamiento inferior a otros que no permitirá llevar a cabo ciertas adaptaciones.

La estrategia propuesta contempla la «consciencia del contexto» de la forma en la que se ha descrito, aunque para tal es necesario imponer consideraciones al sistema anfitrión en el cual se utilizan los elementos de adaptación.

Si bien las ideas que dan surgimiento a *XForms* provienen de la web, así como de proveer cierta robustez a las formas web tradicionales, el uso de *XForms* en otros ambientes depende de la interpretación que se realiza sobre los componentes que *XForms* provee, los cuales pueden ser desacoplados y utilizados de manera independiente, además es necesario apegarse al patrón *MVC*.

VI.1.2 Aportaciones

La propuesta se basa totalmente en el estándar *XForms*, y las tecnologías en las que se apoya para definir su especificación, y aunque para propósitos prácticos en los escenarios de aplicación se utiliza *XHTML* como lenguaje anfitrión, el uso de *XForms* provee el soporte a la independencia de dispositivos. En la propuesta el proceso de composición da como resultado un «documento *XForms*».

Los elementos de adaptación propuestos se basan en documentos editables y extensibles, lo que les permite ser aplicados en otros escenarios a través de la creación de nuevos documentos, sin la necesidad de realizar codificación alguna. Los «documentos esquema»

definidos dan apoyo a la consistencia estructural para la definición de nuevos modelos. El primer tipo de extensión posible se basa en la generación de documentos para ser utilizados en otro tipo de escenarios. El segundo tipo de extensión el cual requiere codificación, se presentará cuando se desee incorporar nuevos tipos de adaptaciones o definir otro tipo de estrategias y relaciones entre los modelos.

La manera en la que se especifican las relaciones al «contexto» es muy genérica, ya que se limita a utilizar la información que el modelo define. Por lo tal, la factibilidad de utilizar otro tipo de tecnología para definir el «contexto» es viable bajo la encomienda de utilizar únicamente la información que el modelo captura e implementar un intérprete para dicho modelo.

El nivel de abstracción provisto se refleja en el «modelo de elementos» y la implementación del «compositor *XForms*». Por lo tal, se provee un nivel de abstracción al desarrollo de interfaces *XForms* que incorporan una estructura que refleja adaptaciones guiadas por el «contexto», a través de eventos genéricos.

VI.1.3 Trabajo Futuro

El trabajo futuro en torno a los elementos de adaptación desarrollados se relaciona a la definición y extensión de nuevos tipos de adaptaciones conforme se presenten otro tipo de escenarios.

Uno de los principales retos del trabajo fue la definición del «modelo de elementos» ya que como se menciona el enfoque tradicional indica que a partir de la «interfaz abstracta» se obtiene la «interfaz concreta», por lo tal, en este momento sólo describe características que permiten seleccionar un elemento *XForms* adecuado para incorporar en la «interfaz abstracta», por lo tal se desea incorporar otro tipo de información para tomar en cuenta durante la generación de la «interfaz abstracta». Debido a que *XForms* no provee soporte a algún modelo declarativo (tareas, dialogo, presentación, etc.), la propuesta es incorporar un «modelo de tareas», en conjunto con el «modelo de elementos» y el «modelo del contexto», dentro de la «primera etapa de adaptación». Dicho modelo permitiría contemplar nuevos criterios para generar la «interfaz abstracta», por ejemplo, en vez de excluir un elemento dentro de la interfaz debido a las capacidades del dispositivo o información del entorno, se

podrían modelar alternativas en forma de tareas. Suponga que un elemento de la «interfaz abstracta» dicta que el usuario capture información, si dicha interfaz se interpretara para un dispositivo como el *FlowerBlink,* esto no es posible, aquí la propuesta es definir una tarea alternativa que indique que se infiera dicha información o se obtenga de algún repositorio.

Referencias

Alamán, X., Cabello, R., Gómez-Arriba, F., Haya, P., Martínez, A., Martínez, J. y Montoro, G. 2003. "*Using context information to generate dynamic user interfaces*". En: Proceedings of the10th International Conference on Human-Computer Interaction, HCI International. Creta, Greece, 22-23 Junio.

Álvarez, D. 2006. "*CC/PP: Composite Capabilities and Preference Profiles*". Reporte Técnico. http://di002.edv.uniovi.es/~cueva/asignaturas/doctorado/2006/trabajos/ccpp.pdf (Consultado en Enero del 2008)

Beckett, D. 2004. "*RDF/XML Syntax Specification*". http://www.w3.org/TR/rdf-syntax-grammar. (Consultado en Febrero del 2008)

Berhe, G., Brunie, L., y Pierson, J. 2004. "*Modeling service-based multimedia content adaptation in pervasive computing*". En: Proceedings of the 1st Conference on Computing Frontiers. Ischia, Italia, 14-16 Abril. pp 60-69.

B'Far, R. 2005. "*Mobile Computing Principles*". Cambridge University Press. ISBN-13 978-0-511-26576-1

Bomsdorf, B., y Szwillus, G. 1998. "*From Task to Dialogue: Task-Based User Interface Design*". En: Proceedings of CHI '98 Workshop, ACM Press, pp 18-23.

Boyer, J. 2007. "*XForms 1.0 (Third Edition)*". http://www.w3.org/TR/2007/REC-xforms-20071029/ (Consultado en Julio del 2008)

Braun, E., Austaller, G. y Kangasharju, J., Mühlhäuser, M. 2004a. "*Accessing Web Applications with Multiple Context-Aware Devices*". En: Proceedings of the ICWE Workshops, Singapur, 10-12 Septiembre. pp. 353-366

Braun, E. y Mühlhäuser, M. 2004b. "*Extending XML UIDLs for MultiDevice Scenarios*" En: Proceedings of Workshop on Developing User Interfaces with XML: Advances on User Interface Description Languages, Advanced Visual Interfaces.

Brickley, D. y Guha, R. 2004. "*RDF Vocabulary Description Language 1.0: RDF Schema*". http://www.w3.org/TR/rdf-schema/

Butter, T., Aleksy, M., Bostan, P. y Schader, M. 2007. "*Context-aware User Interface Framework for Mobile Applications*". En: Proceedings of the 27th International Conference on Distributed Computing Systems Workshops, Washington, DC, 22-29 Junio. pp. 39.

Cagle, K. 2005. "*The Secret Life of XForms*".

http://www.devx.com/xml/Article/17714/1954 (Consultado en Octubre del 2007)
Clark, J. y DeRose, S. 1999. "*XML Path Language (XPath) Version 1.0*". http://www.w3.org/TR/xpath (Consultado en Febrero del 2008)

Clerckx, T. y Coninx, K. 2005. "*Towards an Integrated Development Environment for Context-Aware User Interfaces*". En: Proceedings of Seminar Mobile Computing and Ambient Intelligence: The Challenge of Multimedia.

Cogliati, A. 2006. "*Personal Content Applications' User Interface Adaptation*". En: Proceedings from Research Seminar on Multimedia.

Dey, A. y Abowd, G. 2000. "*Towards a Better Understanding of Context and Context-awareness*". En: Proceedings of the Workshop on the What, Who, Where, When and How of Context-Awareness, affiliated with the CHI 2000 Conference on Human Factors in Computer Systems, Karlsruhe, Alemania, 27-29 Septiembre. pp. 304-307.

Dey, A. y Salber, D. Abowd, G. 2001. "*A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications*". pp. 97-166.

Ding, Y. y Litz, H. 2006. "*Creating Multiplatform User Interfaces by Annotation and Adaptation*". En: Proceedings of the 11th international conference on Intelligent user interfaces, Sydney, Australia, 29 Enero – 1 Febrero. pp. 270 - 272.

Dourish, P. 2004. "*What We Talk About When We Talk About Context*". Personal Ubiquitous Computing. 8 (1)

Dubinko, M. 2003. "*XForms Essentials*". O'Reilly. ISBN: 0-596-00369-2

Elliot, K., Watson, M., Neustaedter, C. y Greenberg, S. 2007. "*Location-dependent information appliances for the home*". En: Proceedings of Graphics Interface, Montreal, Canada, 28-30 Mayo. pp. 151 - 158.

Favela, J., Tentori, M., Segura, D., y Berzunza, G. 2009. "*Adaptive Awareness of Hospital Patient Information through Multiple Sentient Displays*". En: Proceedings of International Journal of Ambient Computing and Intelligence. 1(1), pp 27-38.

Feuerstack, S., Blumendorf, M., Schwartze, V. y Albayrak, S. 2008. "*Model-based layout generation*". En: Proceedings of the Working Conference on Advanced Visual interfaces, Napoli, Italia, 28-30 Mayo.

Forbrig, P., Dittmar, A. y Muller, A. 2004. "*Adaptive task modelling: From formal methods to XML representations*". En: Multiple User Interfaces, A. Seffah and H. Javahery, Eds. John Wiley & Sons Inc., pp.171-192.

Göschka, K. y Smeikal, R. 2001. "*Interaction Markup Language, An Open Interface for*

Device Independent Interaction with E-Commerce Applications". En: Proceedings of the 34th Hawaii international Conference on System Sciences, Washington, DC, 3-6 Enero.
Greenberg, S. y Fitchett, C. 2001. "*Phidgets: Easy development of physical interfaces through physical widgets*". Paper presented at the 14th annual ACM symposium on User interface *software* and technology. Pp. 209-218.

Grundy, J. y Zou, W. 2004. "*AUIT: Adaptable User Interface Technology, with Extended Java Server Pages*". En: Multiple User Interfaces, A. Seffah and H. Javahery, Eds. John Wiley & Sons Inc., pp 149-167

Haigh, K., Phelps, J. y Geib, C. 2002. "*An open agent architecture for assisting elder independence*". En: Proceedings of the first international joint conference on Autonomous agents and multiagent systems, Bolonia, Italia, 15-19 Julio. pp 578-586.

Hanumansetty, R. 2004. "*Model based approach for context aware and adaptive user interface generation*". Master thesis. Faculty of the Virginia Polytechnic Institute.

Held, A., Buchholz, S. y Schill, A. 2002. "*Modeling of Context Information for Pervasive Computing Applications*". En: Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics.

Honkala, M. y Pohja, M. 2006. "*Multimodal interaction with XForms*". En: Proceedings of the 6th international conference on Web engineering, Palo Alto California, 11-14 Julio. pp. 201-208

Hurtado, N., González, J. y Torres, J. 2004. "*Revisión de Lenguajes Declarativos para la Descripción de Interfaces de Usuario Independientes del Dispositivo*". En: Proceedings of the V Congreso Interacción Persona-Ordenador.

Indulska, J., Robinson, R., Rakotonirainy, A. y Henricksen, K. 2003. "*Experiences in Using CC/PP in Context-Aware Systems*". En: Proceedings of the 4th International Conference on Mobile Data Management, Londres, 21-24 Junio. pp. 247-261.

Kavaldjian, S. 2007. "*A model-driven approach to generating user interfaces*". En: Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Dubrovnik, Croacia, 3-7 Septiembre. pp 303-306.

Klyne, G., Reynolds, F., Woodrow, C., Ohto, H., Hjelm, J., Butler, M. y Tran, L. 2004. "*Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0*". http://www.w3.org/TR/CCPP-struct-vocab/ (Consultado en Febrero del 2008)

Lamadon, J., Herrero, C. y Cogliati, A. 2004. "*Device independent user interfaces - a television programme guide case study*". En: Proceedings from Research Seminar on Digital Media.

Lei, H. 2005. "*Context Awareness: a Practitioner's Perspective*". En: Proceedings of the International Workshop on Ubiquitous Data Management, Washington, DC, 4-4 Abril. pp. 43-52.
Lemlouma, T. y Layaïda, N. 2004. "*Context-Aware Adaptation for Mobile Devices*". En: Proceedings of the IEEE International conference on Mobile Data Management, San Martín Francia. pp. 106-107.

Lewis, R., Merrick, R. y Froumentin, M. 2007. "*Content Selection for Device Independence (DISelect) 1.0*".
http://www.w3.org/TR/cselection/ (Consultado en Abril del 2008)

Liotta, A., Yew, A., Bohoris, C. y Pavlou, G. 2002. "*Supporting Adaptation-aware Services through the Virtual Home Environment Middleware*". En: Proceedings of the 9th Workshop of the HP OpenView University Association.

Luyten, K. 2004. "*Dynamic User Interface Generation for Mobile and Embedded Systems with Model-Based User Interface Development*". PhD Thesis. Hasselt University.

McCreary, D. 2007. "*Generating XForms applications using the National Information Exchange Model (NIEM)*".
http://www.ibm.com/developerworks/library/x-xformsniem/ (Consultado en Marzo del 2008)

Menkhaus, G. y Pree, W. 2002. "*User Interface Tailoring for Multi-Platform Service Access*". En: Proceedings of the 7th international conference on Intelligent user interfaces, San Francisco, California, pp. 208 - 209. 13-16 Enero.

Mohomed, I., Cai, J., Siva, C. y de Lara, E. 2006. "*Context-aware interactive content adaptation*". En: Proceedings of the 4th international Conference on Mobile Systems, Applications and Services, Uppsala, Suecia, 19-22 Junio.

Mynatt, E., Essa, I. y Rogers, W. 2000. "*Increasing the opportunities for aging in place*". En: Proceedings on the 2000 conference on Universal Usability, Arlington, Virginia, Estados Unidos. pp. 65-71. 16-17 Noviembre.

Nathanail, S., Tsetsos, V. y Hadjiefthymiades, S. 2007. "*Sensor-Driven Adaptation of Web Document Presentation*". En: Proceedings of the 12th International Conference on Human Computer Interaction, pp 406-415.

Nylander, S., Bylund, M. y Waern; A. 2005. "*Ubiquitous service access through adapted user interfaces on multiple devices*". Archive: Personal and Ubiquitous Computing, 9(3), pp 123 – 133, Mayo.

Paterno, F., Mancini, C. y Meniconi, S. 1997. "*ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models*". En: Proceedings of the IFIP TC13 International

Conference on Human-Computer Interaction, Londres Reino Unido. pp. 362 - 369.

Phanouriou, C. 2000. "*UIML: A Device-Independent User Interface Markup Language*". PhD thesis, Virginia Polytechnic Institute and State University.

Pohja, M., Honkala, M., Penttinen, M., Vuorimaa, P. y Ervamaa, P. 2006. "*Web User Interaction - Comparison of Declarative Approaches*". En: Proceedings of the 2nd International Conference on Web Information Systems and Technologies, Abril, pp. 295-302.

Puerta, A. 1997. "*A Model-Based Interface Development Environment*". En: IEEE Software IEEE Software, 14 (4), pp.40-47.

Puerta, A. y Eisenstein, J. 2001. "*XIML: A Universal Language for User Interfaces*". Reporte técnico. http://www.ximl.org/documents/XIMLBasicPaperES.pdf

Puerta, A. y Eisenstein, J. 2002. "*XIML: A common representation for interaction data*". En: Proceedings of the 7th international conference on Intelligent user interfaces, San Francisco, California, 13 - 16 Enero. pp. 214 - 215.

Raman, T. 2003. "*XForms: XML Powered Web Forms*". O'Reilly. ISBN: 0-321-15499-1

Repo, P. y Riekki, J. 2004. "*Middleware Support for Implementing Context-Aware Multimodal User Interfaces*". En: Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia, College Park, Maryland, p27-29 octubre. pp. 221 - 227.

Rivera, J. y Taing, L. 2002. "*Get ready for XForms*". http://www.ibm.com/developerworks/xml/library/x-xforms/ (Consultado en Octubre del 2007)

Salber, D., Dey, A. y Abowd, G. 1999. "*The Context Toolkit: Aiding the Development of Context-Enabled Applications*". En: Proceedings of the 1999 Conference on Human Factors in Computing Systems, Pittsburgh, Pennsylvania, Estados Unidos, 15-20 Mayo. pp. 434-441.

Seffah, A. y Javahery, H. 2004a. "*Multiple User Interfaces: Cross-Platform Applications and Context-Aware Interfaces*". En: Multiple User Interfaces, A. Seffah and H. Javahery, Eds. John Wiley & Sons Inc., pp.11-26.

Seffah, A., Forbrig, P. y Javahery, H. 2004b. "*Multi-devices "Multiple" user interfaces: development models and research opportunities*". En: Proceedings of the Journal of Systems and Software, Octubre, 73(2), pp. 287-300.

Segura, D., Favela, J. y Tentori, M. 2008. "*Sentient displays in support of hospital work*".

Paper presented at the UCAMI.

Shankar, A., Louis, J., Dascalu, S., Hayes, L. y Houmanfar, R. 2007. "*User-Context for Adaptative User Interfaces*". En: Proceedings of the 12th international conference on Intelligent user interfaces, Honolulu, Hawaii. pp. 321 - 324.

Sinng, D., Forbrig, P. y Seffah, A. 2004. "*Patterns in Model-Based Development*". En: Proceedings of the First International Workshop of MBUI, Palo Alto, California. 10-14 Julio.

Smith, K. 2007. "*Device Independent Authoring Language (DIAL)*". http://www.w3.org/TR/dial (Consultado en Abril del 2008)

Souchon, N. y Vanderdonckt, J. 2003. "*A Review of XML-Compliant User Interface Description Languages*". En: Proceedings of 10th International Conference on Design, Specification, and Verification of Interactive Systems. Springer-Verlag, Berlin, 2003, pp. 377-391.

Strang, T. y Linnhoff-Popien, C. 2004. "*A Context Modeling Survey*". En: Proceedings of Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp. Nottingham / Inglaterra, Septiembre.

Thevenin, D. y Coutaz, J. 1999. "*Plasticity of User Interfaces: Framework and Research Agenda*". En: Proceedings of Conference on Human Computer Interaction, Edimburgo. pp 110-117.

Thevenin, D., Coutaz, J. y Calvary, G. 2004. "*A Reference Framework for the Development of Plastic User Interfaces*". En: Multiple User Interfaces, A. Seffah and H. Javahery, Eds. John Wiley & Sons Inc., pp.29-51.

Trewin, S., Zimmermann, G. y Vanderheiden, G. 2003. "*Abstract user interface representations: how well do they support universal access?*" En: Proceedings of 2003 conference on Universal usability, Vancouver, Canada, pp. 77-84.

Trewin, S., Zimmermann, G. y Vanderheiden, G. 2004. "*Abstract representations as a basis for usable user interfaces*". Interacting with Computers, 64(5), pp. 477-506.

Van den Bergh, J. y Coninx, K. 2004. "*Model-based design of context-sensitive interactive applications: a discussion of notations*". En: Proceedings of the 3rd Annual Conference on Task Models and Diagrams. Praga, República Checa, pp 43-50

Viterbo, J., Casanova, M., Rubinsztejn, H. y Endler, M. 2006. "*An Ontology Based on the CC/PP Framework to Support Content Adaptation in Context-aware Systems*". En: Proceedings of the 1st Workshop on Ontologies and Metamodels in Software and Data Engineering. Florianópolis, pp. 1-10.

Weiser, M. 1993. "*Some computer science issues in ubiquitous computing*". Communications of the ACM, 36 (7), pp 74 - 84.

Weiser, M. y Brown, J. 1996. "*The coming age of calm technology*". Book: Beyond calculation: the next fifty years, pp. 75-85.

Witt, H. 2005. "*A toolkit for Context-aware User Interface Development for wearable Computers*". En: Doctoral Colloquium at the 9th International Symposium on Wearable Computers. Osaka, Japón, 18-21 Octubre.

"*XForms Tutorial and Cookbook*". 2008
http://en.wikibooks.org/wiki/XForms/ (Consultado en Febrero del 2008)

Apéndices

Apéndice A. Documento esquema para el modelo de elementos.

Documento esquema en *XML Schema* utilizado para definir el «modelo de elementos».

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- edited with XMLSpy v2008 sp1 (http://www.altova.com) by Gustavo Berzunza (A) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="modelo" type="tipoModelo"/>
  <xs:complexType name="tipoModelo">
    <xs:sequence>
      <xs:element name="instancia" type="tipoInstancia" maxOccurs="unbounded"/>
      <xs:element name="entidad" type="tipoEntidad" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="tipoInstancia">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="id" type="xs:string" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <xs:complexType name="tipoEntidad">
    <xs:sequence>
      <xs:element name="interfaz" type="tipoInterfaz"/>
      <xs:element name="referencia" type="tipoReferencia" minOccurs="0"/>
      <xs:element name="adaptacion" type="tipoAdaptacion" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:string" use="optional" default=""/>
    <xs:attribute name="prioridad" type="xs:integer" use="optional" default="1"/>
  </xs:complexType>
  <xs:complexType name="tipoInterfaz">
    <xs:sequence>
      <xs:element name="forma" type="tipoForma" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="etiqueta" type="xs:string" use="optional" default=""/>
    <xs:attribute name="control" type="tipoControl" use="optional" default="input"/>
  </xs:complexType>
  <xs:complexType name="tipoReferencia">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="mediaType" type="xs:string" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <xs:simpleType name="tipoControl">
    <xs:restriction base="xs:string">
      <xs:enumeration value="input"/>
      <xs:enumeration value="output"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="tipoForma">
    <xs:sequence>
      <xs:element name="fuenteOpcion" type="xs:string" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="controlForma" type="tipoControlForma" use="required"/>
  </xs:complexType>
  <xs:simpleType name="tipoControlForma">
    <xs:restriction base="xs:string">
      <xs:enumeration value="select"/>
      <xs:enumeration value="select1"/>
      <xs:enumeration value="textarea"/>
      <xs:enumeration value="secret"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="tipoAdaptacion">
    <xs:restriction base="xs:string">
      <xs:enumeration value="definePrivado"/>
      <xs:enumeration value="defineRelevante"/>
      <xs:enumeration value="actualizaDatos"/>
      <xs:enumeration value="defineObjetivo"/>
      <xs:enumeration value="defineAviso"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

Apéndice B. Documento Esquema para las reglas de adaptación.

Documento esquema en *XML Schema* utilizado para definir las «reglas de adaptación».

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- edited with XMLSpy v2008 sp1 (http://www.altova.com) by Gustavo Berzunza (A) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="reglas" type="tipoReglas"/>
  <xs:complexType name="tipoReglas">
    <xs:sequence>
      <xs:element name="regla" type="tipoRegla" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="tipoRegla">
    <xs:sequence>
      <xs:element name="condicion" type="tipoCondicion"/>
      <xs:element name="acciones" type="tipoAcciones"/>
    </xs:sequence>
    <xs:attribute name="tipo" type="tipoRt" use="required"/>
    <xs:attribute name="objetivo" type="tipoObjetivo" use="optional" default="generico"/>
    <xs:attribute name="idEntidad" type="xs:string" use="optional" default=""/>
  </xs:complexType>
  <xs:complexType name="tipoCondicion">
    <xs:choice>
      <xs:element name="contexto" type="tipoContexto"/>
      <xs:element name="entidad" type="tipoEntidad"/>
      <xs:element name="condicion" type="tipoCondicion" maxOccurs="unbounded"/>
    </xs:choice>
    <xs:attribute name="tipo" type="tipoDeCondicion" use="required"/>
  </xs:complexType>
  <xs:complexType name="tipoContexto">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="fuenteCtx" type="tipoFuenteCtx" use="required"/>
        <xs:attribute name="nombreCtx" type="xs:string" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <xs:complexType name="tipoEntidad">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="propiedad" type="tipoPropiedad" use="optional" default="control"/>
        <xs:attribute name="adaptacion" type="tipoAdaptacion" use="optional" default="none"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <xs:simpleType name="tipoRt">
    <xs:restriction base="xs:string">
      <xs:enumeration value="estatica"/>
      <xs:enumeration value="dinamica"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="tipoObjetivo">
    <xs:restriction base="xs:string">
      <xs:enumeration value="generico"/>
      <xs:enumeration value="particular"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="tipoDeCondicion">
    <xs:restriction base="xs:string">
      <xs:enumeration value="y"/>
      <xs:enumeration value="o"/>
      <xs:enumeration value="igual"/>
      <xs:enumeration value="noIgual"/>
      <xs:enumeration value="lt"/>
      <xs:enumeration value="gt"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="tipoFuenteCtx">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Cliente"/>
      <xs:enumeration value="Entorno"/>
      <xs:enumeration value="Conocido"/>
    </xs:restriction>
  </xs:simpleType>
```

Apéndice B. (continuación)

```
  <xs:simpleType name="tipoPropiedad">
    <xs:restriction base="xs:string">
      <xs:enumeration value="control"/>
      <xs:enumeration value="etiqueta"/>
      <xs:enumeration value="mediaType"/>
      <xs:enumeration value="prioridad"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="tipoAdaptacion">
    <xs:restriction base="xs:string">
      <xs:enumeration value="none"/>
      <xs:enumeration value="definePrivado"/>
      <xs:enumeration value="defineRelevante"/>
      <xs:enumeration value="actualizaDatos"/>
      <xs:enumeration value="defineObjetivo"/>
      <xs:enumeration value="defineAviso"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="tipoAcciones">
    <xs:sequence>
      <xs:choice>
        <xs:element name="accion" type="tipoAccion"/>
        <xs:element name="monitor" type="tipoMonitor"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="tipoAccion">
    <xs:sequence>
      <xs:element name="define" type="tipoDefine" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="defineAccion" type="tipoDefineAccion" use="required"/>
  </xs:complexType>
  <xs:simpleType name="tipoDefineAccion">
    <xs:restriction base="xs:string">
      <xs:enumeration value="defineControl"/>
      <xs:enumeration value="noIncluye"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="tipoMonitor">
    <xs:sequence>
      <xs:element name="registro" type="tipoRegistro" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="tipoRegistro">
    <xs:simpleContent>
      <xs:extension base="xs:boolean">
        <xs:attribute name="fuenteCtx" type="tipoFuenteCtx" use="required"/>
        <xs:attribute name="nombreCtx" type="xs:string" use="required"/>
        <xs:attribute name="estado" type="xs:string" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <xs:complexType name="tipoDefine">
    <xs:attribute name="control" type="tipoControl" use="required"/>
  </xs:complexType>
  <xs:simpleType name="tipoControl">
    <xs:restriction base="xs:string">
      <xs:enumeration value="input"/>
      <xs:enumeration value="output"/>
      <xs:enumeration value="secret"/>
      <xs:enumeration value="textarea"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

Apéndice C. Documento esquema del contexto del entorno para el primer escenario.

Documento esquema en *DRF Schema* utilizado para definir el «modelo del contexto» relacionado al cliente (dispositivo, usuario) en el escenario 1 del Capítulo V.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <!-- Perfil -->
  <rdf:Description rdf:ID="Cliente">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
    <rdfs:comment>Clase que incluye los diferentes componentes para el Perfil</rdfs:comment>
  </rdf:Description>
  <!-- Componentes del Perfil -->
  <rdf:Description rdf:ID="Plataforma">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="#Cliente"/>
    <rdfs:comment>Describe la plataforma del dispositivo cliente Hw y Sw</rdfs:comment>
  </rdf:Description>
  <rdf:Description rdf:ID="Usuario">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="#Cliente"/>
    <rdfs:comment>Describe las preferencias del usuario</rdfs:comment>
  </rdf:Description>
  <!-- Atributos del Componente - Plataforma -->
  <rdf:Description rdf:ID="TipoDispositivo">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdfs:domain rdf:resource="#Plataforma"/>
    <rdfs:comment>Tipo de dispositivo, ej.: "PDA, PC, Laptop"</rdfs:comment>
  </rdf:Description>
  <rdf:Description rdf:ID="SoporteImagenes">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdfs:domain rdf:resource="#Plataforma"/>
    <rdfs:comment>Define si soporta o no imagenes, p. ej.: "verdadero, falso"</rdfs:comment>
  </rdf:Description>
  <rdf:Description rdf:ID="TamanioPantallaCaracteres">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdfs:domain rdf:resource="#Plataforma"/>
    <rdfs:comment>Define el número de caracteres posibles, p. ej.: "30x22"</rdfs:comment>
  </rdf:Description>
  <rdf:Description rdf:ID="SistemaOperativo">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdfs:domain rdf:resource="#Plataforma"/>
    <rdfs:comment>El sistema operativo, p. ej.: "Windows CE", "win9x"</rdfs:comment>
  </rdf:Description>
  <!-- Atributos de Componente - Usuario -->
  <rdf:Description rdf:ID="Lenguaje">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Bag"/>
    <rdfs:domain rdf:resource="#Usuario"/>
    <rdfs:comment>Da el conjunto de lenguajes que el usuario utiliza.
    </rdfs:comment>
  </rdf:Description>
  <rdf:Description rdf:ID="Navegacion">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdfs:domain rdf:resource="#Usuario"/>
    <rdfs:comment>Tipo de navegación de preferencia: textual, gráfica</rdfs:comment>
  </rdf:Description>
  <rdf:Description rdf:ID="Permisos">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdfs:domain rdf:resource="#Usuario"/>
    <rdfs:comment>Nivel privilegio del usuario: Administrador, Cliente</rdfs:comment>
  </rdf:Description>
  <!-- Clase para Atributos Dinámicos -->
  <rdf:Description rdf:ID="Dinamico">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
    <rdfs:comment>Clase para definir atributos dinámicos</rdfs:comment>
  </rdf:Description>
  <rdf:Description rdf:ID="fuente">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdfs:domain rdf:resource="#Dinamico"/>
    <rdfs:comment>Describe la fuente de información de los datos</rdfs:comment>
  </rdf:Description>

```

Apéndice C. (continuación)

```
  <rdf:Description rdf:ID="valor">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdfs:domain rdf:resource="#Dinamico"/>
    <rdfs:comment>Describe el valor actual del atributo</rdfs:comment>
  </rdf:Description>
  <rdf:Description rdf:ID="protocolo">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdfs:domain rdf:resource="#Dinamico"/>
    <rdfs:comment>Describe al protocolo de comunicación con la fuente</rdfs:comment>
  </rdf:Description>
</rdf:RDF>
```

Apéndice D. Perfil *CC/PP* para el cliente del primer escenario de aplicación.

Documento «perfil» utilizado para definir el «modelo del contexto» relacionado al cliente (dispositivo, usuario) en el escenario 1 del Capítulo V.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:ccpp="http://www.w3.org/2000/07/04-ccpp#" xmlns:esq="esq-elder-cliente.rdfs#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description rdf:ID="Cliente">
    <ccpp:component>
      <rdf:Description rdf:ID="Plataforma">
        <rdf:type rdf:resource="esq-elder-cliente.rdfs#Plataforma"/>
        <esq:TipoDispositivo>PC</esq:TipoDispositivo>
        <esq:SoporteImagenes>verdadero</esq:SoporteImagenes>
        <esq:TamanioPantallaCaracteres>80x25</esq:TamanioPantallaCaracteres>
      </rdf:Description>
    </ccpp:component>
    <ccpp:component>
      <rdf:Description rdf:ID="Usuario">
        <rdf:type rdf:resource="esq-elder-cliente.rdfs#Usuario"/>
        <esq:Lenguaje>
          <rdf:Bag>
            <rdf:li>ingles</rdf:li>
            <rdf:li>espanol</rdf:li>
          </rdf:Bag>
        </esq:Lenguaje>
        <esq:Navegacion>grafica</esq:Navegacion>
        <esq:Permisos>usuario</esq:Permisos>
      </rdf:Description>
    </ccpp:component>
  </rdf:Description>
</rdf:RDF>
```

Apéndice E. Documento esquema del contexto del entorno para el primer escenario.

Documento esquema en *DRF Schema* utilizado para definir el «modelo del contexto» relacionado al entorno (ubicación, tarea, etc.) en el escenario 1 del Capítulo V.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <!-- P e r f i l -->
  <rdf:Description rdf:ID="Entorno">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
    <rdfs:comment>Clase que incluye los diferentes componentes para el Perfil</rdfs:comment>
  </rdf:Description>
  <!-- C o m p o n e n t e s  d e l  P e r f i l -->
  <rdf:Description rdf:ID="Ubicacion">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="#Entorno"/>
    <rdfs:comment>Describe la ubicación actual del usuario</rdfs:comment>
  </rdf:Description>
  <rdf:Description rdf:ID="Compania">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="#Entorno"/>
    <rdfs:comment>Describe si el usuario se encuentra en compania</rdfs:comment>
  </rdf:Description>
  <rdf:Description rdf:ID="Actividades">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="#Entorno"/>
    <rdfs:comment>Describe las actividades del usuario</rdfs:comment>
  </rdf:Description>
  <!-- A t r i b u t o s  d e l  C o m p o n e n t e  U b i c a c i ó n -->
  <rdf:Description rdf:ID="Fisica">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdf:type rdf:resource="#Dinamico"/>
    <rdfs:domain rdf:resource="#Ubicacion"/>
    <rdfs:comment>Localización Física de la persona. Lugares de la casa.</rdfs:comment>
  </rdf:Description>
  <rdf:Description rdf:ID="Logica">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdf:type rdf:resource="#Dinamico"/>
    <rdfs:domain rdf:resource="#Ubicacion"/>
    <rdfs:comment>Carga actual restante de la Bateria</rdfs:comment>
  </rdf:Description>
  <!-- A t r i b u t o s  d e  C o m p o n e n t e  -  C o m p a n i a -->
  <rdf:Description rdf:ID="Estado">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdf:type rdf:resource="#Dinamico"/>
    <rdfs:domain rdf:resource="#Compania"/>
    <rdfs:comment>Denotará un valor booleano, verdadero o falso</rdfs:comment>
  </rdf:Description>
  <!-- A t r i b u t o s  d e  C o m p o n e n t e  -  A c t i v i d a d -->
  <rdf:Description rdf:ID="BanioMatutino">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdf:type rdf:resource="#Dinamico"/>
    <rdfs:domain rdf:resource="#Actividades"/>
  </rdf:Description>
  <rdf:Description rdf:ID="Desayuno">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdf:type rdf:resource="#Dinamico"/>
    <rdfs:domain rdf:resource="#Actividades"/>
  </rdf:Description>
  <rdf:Description rdf:ID="MedicinaMatutina">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdf:type rdf:resource="#Dinamico"/>
    <rdfs:domain rdf:resource="#Actividades"/>
  </rdf:Description>
  <rdf:Description rdf:ID="CitaMedica">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdf:type rdf:resource="#Dinamico"/>
    <rdfs:domain rdf:resource="#Actividades"/>
  </rdf:Description>
  <rdf:Description rdf:ID="Almuerzo">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdf:type rdf:resource="#Dinamico"/>
    <rdfs:domain rdf:resource="#Actividades"/>
  </rdf:Description>
```

Apéndice E. (continuación)

```
  <rdf:Description rdf:ID="MedicinaVespertina">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdf:type rdf:resource="#Dinamico"/>
    <rdfs:domain rdf:resource="#Actividades"/>
  </rdf:Description>
  <rdf:Description rdf:ID="ActividadActual">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdf:type rdf:resource="#Dinamico"/>
    <rdfs:domain rdf:resource="#Actividades"/>
  </rdf:Description>
  <!-- Clase para Atributos Dinámicos -->
  <rdf:Description rdf:ID="Dinamico">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
    <rdfs:comment>Clase para definir atributos dinámicos</rdfs:comment>
  </rdf:Description>
  <rdf:Description rdf:ID="fuente">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdfs:domain rdf:resource="#Dinamico"/>
    <rdfs:comment>Describe la fuente de información de los datos</rdfs:comment>
  </rdf:Description>
  <rdf:Description rdf:ID="valor">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdfs:domain rdf:resource="#Dinamico"/>
    <rdfs:comment>Describe el valor actual del atributo</rdfs:comment>
  </rdf:Description>
  <rdf:Description rdf:ID="protocolo">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdfs:domain rdf:resource="#Dinamico"/>
    <rdfs:comment>Describe al protocolo de comunicación con la fuente</rdfs:comment>
  </rdf:Description>
</rdf:RDF>
```

Apéndice F. Perfil *CC/PP* para el entorno del primer escenario de aplicación.

Documento «perfil» utilizado para definir el «modelo del contexto» relacionado al entorno (ubicación, tarea, etc.) en el escenario 1 del Capítulo V.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:ccpp="http://www.w3.org/2000/07/04-ccpp#" xmlns:esq="esq-elder-entorno.rdfs#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description rdf:ID="Entorno">
    <ccpp:component>
      <rdf:Description rdf:ID="Ubicacion">
        <rdf:type rdf:resource="esq-elder-entorno.rdfs#Ubicacion"/>
        <esq:Fisica rdf:parseType="Resource">
          <rdf:type rdf:resource="esq-elder-entorno.rdfs#Dinamico"/>
          <esq:fuente>idsensor</esq:fuente>
          <esq:valor>cuarto</esq:valor>
          <esq:protocolo>protocolo</esq:protocolo>
        </esq:Fisica>
        <esq:Logica rdf:parseType="Resource">
          <rdf:type rdf:resource="esq-elder-entorno.rdfs#Dinamico"/>
          <esq:fuente>192.0.0.254</esq:fuente>
          <esq:valor>192.0.0.9</esq:valor>
          <esq:protocolo>HTTP</esq:protocolo>
        </esq:Logica>
      </rdf:Description>
    </ccpp:component>
    <ccpp:component>
      <rdf:Description rdf:ID="Compania">
        <rdf:type rdf:resource="esq-elder-entorno.rdfs#Compania"/>
        <esq:Estado rdf:parseType="Resource">
          <rdf:type rdf:resource="esq-elder-entorno.rdfs#Dinamico"/>
          <esq:fuente>idsensor</esq:fuente>
          <esq:valor>falso</esq:valor>
          <esq:protocolo>protocolo</esq:protocolo>
        </esq:Estado>
      </rdf:Description>
    </ccpp:component>
    <ccpp:component>
      <rdf:Description rdf:ID="Actividades">
        <rdf:type rdf:resource="esq-elder-entorno.rdfs#Actividades"/>
        <esq:BanioMatutino rdf:parseType="Resource">
          <rdf:type rdf:resource="esq-elder-entorno.rdfs#Dinamico"/>
          <esq:fuente>idsensor</esq:fuente>
          <esq:valor>falso</esq:valor>
          <esq:protocolo>protocolo</esq:protocolo>
        </esq:BanioMatutino>
        <esq:Desayuno rdf:parseType="Resource">
          <rdf:type rdf:resource="esq-elder-entorno.rdfs#Dinamico"/>
          <esq:fuente>idsensor</esq:fuente>
          <esq:valor>falso</esq:valor>
          <esq:protocolo>protocolo</esq:protocolo>
        </esq:Desayuno>
        <esq:MedicinaMatutina rdf:parseType="Resource">
          <rdf:type rdf:resource="esq-elder-entorno.rdfs#Dinamico"/>
          <esq:fuente>idsensor</esq:fuente>
          <esq:valor>falso</esq:valor>
          <esq:protocolo>protocolo</esq:protocolo>
        </esq:MedicinaMatutina>
        <esq:CitaMedica rdf:parseType="Resource">
          <rdf:type rdf:resource="esq-elder-entorno.rdfs#Dinamico"/>
          <esq:fuente>idsensor</esq:fuente>
          <esq:valor>falso</esq:valor>
          <esq:protocolo>protocolo</esq:protocolo>
        </esq:CitaMedica>
        <esq:Almuerzo rdf:parseType="Resource">
          <rdf:type rdf:resource="esq-elder-entorno.rdfs#Dinamico"/>
          <esq:fuente>idsensor</esq:fuente>
          <esq:valor>falso</esq:valor>
          <esq:protocolo>protocolo</esq:protocolo>
        </esq:Almuerzo>
        <esq:MedicinaVespertina rdf:parseType="Resource">
          <rdf:type rdf:resource="esq-elder-entorno.rdfs#Dinamico"/>
          <esq:fuente>idsensor</esq:fuente>
          <esq:valor>falso</esq:valor>
          <esq:protocolo>protocolo</esq:protocolo>
        </esq:MedicinaVespertina>
```

Apéndice F. (continuación)

```
        <esq:ActividadActual rdf:parseType="Resource">
          <rdf:type rdf:resource="esq-elder-entorno.rdfs#Dinamico"/>
          <esq:fuente>idsensor</esq:fuente>
          <esq:valor>null</esq:valor>
          <esq:protocolo>protocolo</esq:protocolo>
        </esq:ActividadActual>
      </rdf:Description>
    </ccpp:component>
  </rdf:Description>
</rdf:RDF>
```

Apéndice G. Modelo de elementos para el caso uno del primer escenario.

Documento del «modelo de elementos» relacionado al primer caso del escenario 1 del Capítulo V.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<modelo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="modelo-elementos-esquema.xsd">

  <instancia id="actividades">actividades.xml</instancia>

  <entidad prioridad="1">
    <interfaz control="output"/>
    <referencia mediaType="text/plain">instance('actividades')/servicio</referencia>
  </entidad>

  <entidad prioridad="1">
    <interfaz control="output"/>
    <referencia mediaType="text/plain">instance('actividades')/medicina</referencia>
  </entidad>

  <entidad prioridad="1">
    <interfaz control="output"/>
    <referencia mediaType="text/plain">instance('actividades')/consulta</referencia>
  </entidad>

  <entidad prioridad="1" id="idConsultaDetalles">
    <interfaz control="output"/>
    <referencia mediaType="text/plain">instance('actividades')/consultad</referencia
    <adaptacion>defineRelevante</adaptacion>
  </entidad>

  <entidad prioridad="1">
    <interfaz control="output"/>
    <referencia mediaType="text/plain">instance('actividades')/comida</referencia>
  </entidad>

</modelo>
```

Apéndice H. Modelo de elementos para el caso dos del primer escenario.

Documento del «modelo de elementos» relacionado al segundo caso del escenario 1 del Capítulo V.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<modelo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="modelo-elementos-esquema.xsd">

  <instancia id="cuenta">cuenta.xml</instancia>

  <entidad prioridad="1">
    <interfaz control="output"/>
    <referencia mediaType="text/plain">instance('cuenta')/servicio</referencia>
  </entidad>

  <entidad prioridad="1">
    <interfaz control="output" etiqueta="Cliente"/>
    <referencia mediaType="text/plain">instance('cuenta')/nombre</referencia>
  </entidad>

  <entidad prioridad="1">
    <interfaz control="output" etiqueta="Cuenta"/>
    <referencia mediaType="text/plain">instance('cuenta')/cuenta</referencia>
    <adaptacion>definePrivado</adaptacion>
  </entidad>

  <entidad prioridad="1">
    <interfaz control="output" etiqueta="Tarjeta"/>
    <referencia mediaType="text/plain">instance('cuenta')/tarjeta</referencia>
    <adaptacion>definePrivado</adaptacion>
  </entidad>

</modelo>
```

Apéndice I. Reglas de adaptación del primer escenario de aplicación.

Documento de las «reglas de adaptación » para el escenario 1 del Capítulo V.

```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<reglas xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="reglas-esquema.xsd">

  <regla tipo="estatica" objetivo="generico">
    <condicion tipo="y">
      <condicion tipo="igual">
        <contexto fuenteCtx="Cliente" nombreCtx="Plataforma/SoporteImagenes">falso</contexto>
      </condicion>
      <condicion tipo="igual">
        <entidad propiedad="mediaType">image/*</entidad>
      </condicion>
    </condicion>
    <acciones>
      <accion defineAccion="noIncluye"/>
    </acciones>
  </regla>

  <regla tipo="estatica" objetivo="generico">
    <condicion tipo="y">
      <condicion tipo="igual">
        <contexto fuenteCtx="Cliente" nombreCtx="Plataforma/TipoDispositivo">PDA</contexto>
      </condicion>
      <condicion tipo="o">
        <condicion tipo="igual">
          <entidad propiedad="prioridad">3</entidad>
        </condicion>
        <condicion tipo="gt">
          <entidad propiedad="prioridad">3</entidad>
        </condicion>
      </condicion>
    </condicion>
    <acciones>
      <accion defineAccion="noIncluye"/>
    </acciones>
  </regla>

  <regla tipo="dinamica" objetivo="generico">
    <condicion tipo="igual">
      <entidad adaptacion="definePrivado"/>
    </condicion>
    <acciones>
      <monitor>
        <registro fuenteCtx="Entorno" nombreCtx="Compania/Estado" estado="noSolo">true</registro>
        <registro fuenteCtx="Entorno" nombreCtx="Compania/Estado" estado="solo">false</registro>
      </monitor>
    </acciones>
  </regla>

  <regla tipo="dinamica" objetivo="particular" idEntidad="idConsultaDetalles">
    <condicion tipo="igual">
      <entidad adaptacion="defineRelevante" />
    </condicion>
    <acciones>
      <monitor>
        <registro fuenteCtx="Entorno" nombreCtx="Actividades/MedicinaMatutina"
                  estado="verdadero">true</registro>
        <registro fuenteCtx="Entorno" nombreCtx="Actividades/MedicinaMatutina"
                  estado="falso">false</registro>
      </monitor>
    </acciones>
  </regla>
```

Apéndice I. (continuación)

```
  <regla tipo="estatica" objetivo="generico">
    <condicion tipo="y">
      <condicion tipo="igual">
        <entidad propiedad="control">input</entidad>
      </condicion>
      <condicion tipo="igual">
        <entidad adaptacion="definePrivado"/>
      </condicion>
    </condicion>
    <acciones>
      <accion defineAccion="defineControl">
        <define control="secret"/>
      </accion>
    </acciones>
  </regla>

</reglas>
```

Apéndice J. Implementación de *Comet.*

Rutinas implementadas para el envío de datos del servidor al cliente en el escenario 1, implementadas utilizando dojo 1.0.2 y Jetty 6.1.7.

Implementación dojo 1.0.2 (cliente)

```
<script type="text/javascript" src="src/dojox/cometd.js"></script>
<script type="text/javascript" >

  dojo.require("dojox.cometd");
  dojox.cometd.init(new
String(document.location).replace(/http:\/\/[^\/]*/,'').replace(/\/aplicaciones/,'')+"/cometd", {});

  publishHandler = function(msg) {
    var temp = new Array();
    temp = msg.data.test.split("_");
    var e = document.createEvent('Events');
    e.initEvent(temp[0], true, true);
    document.getElementById(temp[1]).dispatchEvent(e);
  }

  dojox.cometd.subscribe("/channel/events", publishHandler);

</script>
```

Implementación Jetty 6.1.7 (servidor)

```
Bayeux b = (Bayeux)MyServlet.getServletContext().getAttribute(Bayeux.DOJOX_COMETD_BAYEUX);
Channel c = b.getChannel("/channel/events", true);
if (c != null){
  Map<String,Object> message = new HashMap<String,Object>();
  message.put("test", "Event"+identificadorEvento+"_Action"+ identificadorEvento);
  c.publish(b.newClient("server",null),message, "");
}
```

Apéndice K. Extensión para el caso uno del primer escenario de aplicación.

Extensiones realizadas a los elementos de adaptación para el caso 1 del escenario 1 del Capítulo V.

«regla de adaptación» agregada

```
...
  <regla tipo="dinamica" objetivo="generico">
    <condicion tipo="igual">
      <entidad adaptacion="defineRelevante" />
    </condicion>
    <acciones>
      <monitor>
        <registro fuenteCtx="Entorno" nombreCtx="Actividades/MedicinaMatutina"
                  estado="verdadero">false</registro>
        <registro fuenteCtx="Entorno" nombreCtx="Actividades/MedicinaMatutina"
                  estado="falso">true</registro>
      </monitor>
    </acciones>
  </regla>
...
```

Entidad del «modelo de elementos» modificada

```
...
  <entidad prioridad="1">
    <interfaz control="output"/>
    <referencia mediaType="text/plain">instance('actividades')/medicina</referencia>
    <adaptacion>defineRelevante</adaptacion>
  </entidad>
...
```

Entidad del «modelo de elementos» agregada

```
...
  <entidad prioridad="1">
    <interfaz control="output"/>
    <referencia mediaType="image/*">instance('actividades')/medicinaimg</referencia>
    <adaptacion>defineRelevante</adaptacion>
  </entidad>
...
```

Apéndice L. Documento esquema del contexto del entorno para el segundo escenario.

Documento esquema en *DRF Schema* utilizado para definir el «modelo del contexto» relacionado al entorno en el escenario 2 del Capítulo V.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <rdf:Description rdf:ID="Entorno">  <!-- P e r f i l -->
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
  </rdf:Description>
  <rdf:Description rdf:ID="Ubicacion"> <!-- C o m p o n e n t e s  d e l  P e r f i l -->
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="#Entorno"/>
  </rdf:Description>
  <rdf:Description rdf:ID="Actividades">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="#Entorno"/>
  </rdf:Description>
  <rdf:Description rdf:ID="Paciente">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="#Entorno"/>
  </rdf:Description>
  <rdf:Description rdf:ID="Fisica"> <!-- Atributos de Ubicación -->
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdf:type rdf:resource="#Dinamico"/>
    <rdfs:domain rdf:resource="#Ubicacion"/>
  </rdf:Description>
    <rdf:Description rdf:ID="AlertaBasicaAtendida"> <!-- Atributos de Actividades -->
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdf:type rdf:resource="#Dinamico"/>
    <rdfs:domain rdf:resource="#Actividades"/>
  </rdf:Description>
  <rdf:Description rdf:ID="ActividadActual">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdf:type rdf:resource="#Dinamico"/>
    <rdfs:domain rdf:resource="#Actividades"/>
  </rdf:Description>
  <rdf:Description rdf:ID="IdPaciente"> <!-- Atributos de Paciente -->
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdfs:domain rdf:resource="#Paciente"/>
  </rdf:Description>
  <rdf:Description rdf:ID="Enfermedad">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdfs:domain rdf:resource="#Paciente"/>
  </rdf:Description>
  <rdf:Description rdf:ID="Nivelorina">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdf:type rdf:resource="#Dinamico"/>
    <rdfs:domain rdf:resource="#Paciente"/>
  </rdf:Description>
  <rdf:Description rdf:ID="VecesEvacuacion">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdf:type rdf:resource="#Dinamico"/>
    <rdfs:domain rdf:resource="#Paciente"/>
  </rdf:Description>
  <rdf:Description rdf:ID="NuevaEvacuacion">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdf:type rdf:resource="#Dinamico"/>
    <rdfs:domain rdf:resource="#Paciente"/>
  </rdf:Description>
  <rdf:Description rdf:ID="Dinamico"> <!-- Clase para Atributos Dinámicos -->
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
  </rdf:Description>
  <rdf:Description rdf:ID="fuente">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdfs:domain rdf:resource="#Dinamico"/>
  </rdf:Description>
  <rdf:Description rdf:ID="valor">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdfs:domain rdf:resource="#Dinamico"/>
  </rdf:Description>
  <rdf:Description rdf:ID="protocolo">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Property"/>
    <rdfs:domain rdf:resource="#Dinamico"/>
  </rdf:Description>
</rdf:RDF>
```

Apéndice M. Perfil *CC/PP* para el entorno del segundo escenario de aplicación.

Documento «perfil» utilizado para definir el «modelo del contexto» relacionado al entorno en el escenario 2 del Capítulo V.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:ccpp="http://www.w3.org/2000/07/04-ccpp#" xmlns:esq="esq-hosp-entorno.rdfs#"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description rdf:ID="Entorno">
    <ccpp:component>
      <rdf:Description rdf:ID="Ubicacion">
        <rdf:type rdf:resource="esq-hosp-entorno.rdfs#Ubicacion"/>
        <esq:Fisica rdf:parseType="Resource">
          <rdf:type rdf:resource="esq-hosp-entorno.rdfs#Dinamico"/>
          <esq:fuente>idsensor</esq:fuente>
          <esq:valor>vestibulo</esq:valor>
          <esq:protocolo>protocolo</esq:protocolo>
        </esq:Fisica>
      </rdf:Description>
    </ccpp:component>
    <ccpp:component>
      <rdf:Description rdf:ID="Actividades">
        <rdf:type rdf:resource="esq-hosp-entorno.rdfs#Actividades"/>
        <esq:AlertaBasicaAtendida rdf:parseType="Resource">
          <rdf:type rdf:resource="esq-hosp-entorno.rdfs#Dinamico"/>
          <esq:fuente>idsensor</esq:fuente>
          <esq:valor>falso</esq:valor>
          <esq:protocolo>protocolo</esq:protocolo>
        </esq:AlertaBasicaAtendida>
        <esq:ActividadActual rdf:parseType="Resource">
          <rdf:type rdf:resource="esq-hosp-entorno.rdfs#Dinamico"/>
          <esq:fuente>idsensor</esq:fuente>
          <esq:valor>Revisión</esq:valor>
          <esq:protocolo>protocolo</esq:protocolo>
        </esq:ActividadActual>
      </rdf:Description>
    </ccpp:component>
    <ccpp:component>
      <rdf:Description rdf:ID="Paciente">
        <rdf:type rdf:resource="esq-hosp-entorno.rdfs#Paciente"/>
        <esq:IdPaciente>Pedro</esq:IdPaciente>
        <esq:Enfermedad>Fallo Renal</esq:Enfermedad>
        <esq:NivelOrina rdf:parseType="Resource">
          <rdf:type rdf:resource="esq-hosp-entorno.rdfs#Dinamico"/>
          <esq:fuente>idsensor</esq:fuente>
          <esq:valor>730</esq:valor>
          <esq:protocolo>protocolo</esq:protocolo>
        </esq:NivelOrina>
        <esq:VecesEvacuacion rdf:parseType="Resource">
          <rdf:type rdf:resource="esq-hosp-entorno.rdfs#Dinamico"/>
          <esq:fuente>idsensor</esq:fuente>
          <esq:valor>3</esq:valor>
          <esq:protocolo>protocolo</esq:protocolo>
        </esq:VecesEvacuacion>
        <esq:NuevaEvacuacion rdf:parseType="Resource">
          <rdf:type rdf:resource="esq-hosp-entorno.rdfs#Dinamico"/>
          <esq:fuente>idsensor</esq:fuente>
          <esq:valor>verdadero</esq:valor>
          <esq:protocolo>protocolo</esq:protocolo>
        </esq:NuevaEvacuacion>
      </rdf:Description>
    </ccpp:component>
  </rdf:Description>
</rdf:RDF>
```

Apéndice N. Extensión al modelo de elementos y ejemplo.

Extensión realizada al documento esquema del «modelo de elementos» y la instancia correspondiente para el escenario 2 del Capítulo V.

Elemento modificado del documento esquema

```
<xs:simpleType name="tipoAdaptacion">
  <xs:restriction base="xs:string">
    <xs:enumeration value="definePrivado"/>
    <xs:enumeration value="defineRelevante"/>
    <xs:enumeration value="actualizaDatos"/>
    <xs:enumeration value="defineObjetivo"/>
    <xs:enumeration value="defineAviso"/>
    <xs:enumeration value="defineCondicion"/>
  </xs:restriction>
</xs:simpleType>
```

«modelo de elementos»

```
<?xml version="1.0" encoding="UTF-8"?>
<modelo
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="modelo-elementos-esquema-ext.xsd">
  <instancia id="actualizacion">actualizacion.xml</instancia>
  <entidad prioridad="2">
    <interfaz control="output" etiqueta="Paciente"/>
    <referencia mediaType="text/plain">instance('actualizacion')/paciente</referencia>
  </entidad>
  <entidad prioridad="2">
    <interfaz control="output" etiqueta="Enfermedad"/>
    <referencia mediaType="text/plain">instance('actualizacion')/enfermedad</referencia>
  </entidad>
  <entidad prioridad="2">
    <interfaz control="output" etiqueta="Hora"/>
    <referencia mediaType="text/plain">instance('actualizacion')/hora</referencia>
  </entidad>
  <entidad prioridad="2">
    <interfaz control="output" etiqueta="Veces"/>
    <referencia mediaType="text/plain">instance('actualizacion')/veces</referencia>
  </entidad>
  <entidad prioridad="1" id="IdCantidad">
    <interfaz control="output" etiqueta="Cantidad"/>
    <referencia mediaType="text/plain">instance('actualizacion')/cantidad</referencia>
    <adaptacion>defineCondicion</adaptacion>
  </entidad>
  <entidad prioridad="2">
     <interfaz control="output"/>
     <referencia mediaType="image/*">instance('actualizacion')/imgBolsa</referencia>
  </entidad>
</modelo>
```

Apéndice O. Extensión a las reglas de adaptación y ejemplo.

Extensión realizada al documento esquema de las «reglas de adaptación» y la instancia correspondiente para el escenario 2 del Capítulo V.

Elementos agregados al documento esquema

```
<xs:simpleType name="tipoAdaptacion">
  <xs:restriction base="xs:string">
    <xs:enumeration value="none"/>
    <xs:enumeration value="definePrivado"/>
    <xs:enumeration value="defineRelevante"/>
    <xs:enumeration value="actualizaDatos"/>
    <xs:enumeration value="defineObjetivo"/>
    <xs:enumeration value="defineAviso"/>
    <xs:enumeration value="defineCondicion"/>
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="tipoAcciones">
  <xs:sequence>
    <xs:choice>
      <xs:element name="accion" type="tipoAccion"/>
      <xs:element name="monitor" type="tipoMonitor"/>
      <xs:element name="constructor" type="tipoConstructor"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="tipoConstructor">
  <xs:sequence>
    <xs:element name="expresion" type="tipoExpresion"/>
    <xs:element name="registro" type="tipoRegistro"/>
  </xs:sequence>
  <xs:attribute name="condicional" type="tipoCondicional" use="required"/>
</xs:complexType>
<xs:complexType name="tipoExpresion">
  <xs:attribute name="fuenteCtx" type="tipoFuenteCtx" use="required"/>
  <xs:attribute name="nombreCtx" type="xs:string" use="required"/>
  <xs:attribute name="estado" type="xs:string" use="required"/>
</xs:complexType>
<xs:simpleType name="tipoCondicional">
  <xs:restriction base="xs:string">
    <xs:enumeration value="igual"/>
    <xs:enumeration value="noIgual"/>
    <xs:enumeration value="lt"/>
    <xs:enumeration value="gt"/>
  </xs:restriction>
</xs:simpleType>
```

«reglas de adaptación»

```
<regla tipo="dinamica" objetivo="particular" idEntidad="idCantidad">
  <condicion tipo="igual">
    <entidad adaptacion="defineCondicion"/>
  </condicion>
  <acciones>
    <constructor condicional="gt">
      <expresion fuenteCtx="Entorno" nombreCtx="Paciente/NivelOrina" estado="600"/>
      <registro fuenteCtx="Entorno" nombreCtx="Paciente/NuevaEvacuacion"
                estado="verdadero">true</registro>
    </constructor>
  </acciones>
</regla>
```