UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Título: Solución de Modelado para la Integración de Servicios Gastronómicos Contextuales
Autores: Francisco Manuel Ayr – Imanol Alvarez
Director: Dra. Cecilia Challiol
Codirector: Dra. Silvia Gordillo
Carrera: Licenciatura en Sistemas

Resumen

En la actualidad, las personas invierten cada vez más tiempo delante de las pantallas de los dispositivos móviles. El avance tecnológico en estos dispositivos ha sido determinante para brindar servicios de distintas índoles. También, se han desarrollado numerosas aplicaciones que brindan diferentes tipos de servicios basados en contexto. Estos servicios contextuales se basan en el contexto actual tanto del usuario como del ambiente que lo rodea; cuando el contexto cambia, estos servicios adaptan la funcionalidad o la información que brindan a las aplicaciones móviles.

Uno de los objetivos principales de esta tesina, es proponer una solución de modelado donde se integren servicios contextuales del ámbito gastronómico, ya que si bien existen modelos que brindan soluciones para servicios contextuales, ninguno de ellos está enfocado en resolver características del dominio gastronómico. Esta solución hace hincapié en integrar este tipo de servicios, con el fin de facilitar la construcción de aplicaciones gastronómicas sensibles al contexto. En base al modelo propuesto, se desarrolló un prototipo denominado "HoyNoCocino" que permite apreciar cómo se comportan distintos servicios contextuales.

Palabras Claves

Modelo de Contexto, Servicios Contextuales Gastronómicos, Aplicaciones Móviles Sensibles al Contexto, Sensado

Trabajos Realizados

Se realizó un relevamiento de aplicaciones y modelos existentes en el dominio gastronómico. Luego, se propuso un primer modelo gastronómico y una capa de contexto basada en los modelos estudiados. Finalmente, se integraron dichos modelos para obtener un nuevo modelo gastronómico considerando el desacople de los servicios contextuales gastronómicos.

Por otro lado, se desarrolló un prototipo funcional en base al modelo propuesto para probar el comportamiento de algunos servicios contextuales concretos.

Conclusiones

Se logró diseñar un modelo flexible y escalable con dos capas desacopladas: una de contexto y una de dominio. Con la relación de estas dos capas se obtiene un modelo gastronómico general independiente de los mecanismos de observación de cambios y servicios contextuales que se brindan.

El prototipo desarrollado permitió apreciar como los servicios gastronómicos son brindados de acuerdo a diferentes contextos, por ejemplo, del usuario o del pedido.

Trabajos Futuros

- *Extender el modelo de dominio agregando las clases necesarias para contar con la posibilidad de pagar los servicios consumidos.*
- *Contemplar cuestiones de privacidad del usuario, por ejemplo, que el usuario pueda indicar cuando brindar su posición a otros usuarios.*
- *Implementación de realidad aumentada para visualizar información sobre los sitios gastronómicos.*
- *Contar con la posibilidad de proveer una versión "off-line" del prototipo que contenga la información y el contenido del momento actual del usuario*

Fecha de la presentación: **Marzo 2018**

Agradecimientos

Este agradecimiento es para nuestras familias y para todos aquellos que nos apoyaron desde el primer momento en el que emprendimos este proyecto; para la Dra. Cecilia Challiol, que gracias a su colaboración y esfuerzo para guiarnos en este proceso que llevamos a cabo juntos se pudieron lograr los objetivos planteados.

Índice

1. Introducción

1.1 Motivación

En la actualidad, las personas invierten cada vez más tiempo delante de las pantallas de los dispositivos móviles. Para muchos, estas pantallas se han convertido en el centro neurálgico de acceso a la información y de interacción con distintos servicios. El avance tecnológico tanto del hardware como del software en estos dispositivos, ha sido determinante para brindar servicios de distintas índoles. En los últimos años, se han desarrollado numerosas aplicaciones que brindan diferentes tipos de servicios basados en contexto [Emmanouilidis et al., 2013]. Estos servicios contextuales se basan en el contexto actual tanto del usuario como del ambiente que rodea al mismo; cuando estos valores contextuales cambian, estos servicios adaptan de alguna manera la funcionalidad o la información que brindan las aplicaciones móviles.

Dentro de las aplicaciones más utilizadas a nivel mundial que proveen servicios contextuales, se encuentran *Google Maps*[1] y *Apple Maps*[2] como las principales aplicaciones que brindan servicios de contexto de posicionamiento [Nielsen, 2016]. En el ámbito del entretenimiento, se puede mencionar la aplicación que lanzó *Niantic, Inc. -Pokemon Go-*[3], un videojuego de realidad aumentada basado en la posición del usuario. Por otro lado, *Uber*[4] se destacó brindando un servicio de transporte basado no solo en la posición del usuario sino también en la posición actual de los conductores. Otro ejemplo que tuvo una gran repercusión fue *Happn*[5], la cual es una aplicación que cuenta con servicios basados en posicionamiento, facilitando al usuario la posibilidad de relacionarse con gente que se ha cruzado en su camino.

En el dominio gastronómico, entre las aplicaciones más destacadas que brindan servicios gastronómicos se encuentra *PedidosYa* [PedidosYa, 2009], la cual permite realizar pedidos a restaurantes de comida según la posición del usuario. Otra aplicación conocida en el mercado es *Yelp* [Yelp, 2004], brindando información sobre restaurantes, bares y deliverys, basada en las opiniones de otros usuarios, además de recomendar y tener un buscador de sitios cercanos a la posición actual del usuario. Por otro lado, *Restorando*[6] también provee un buscador de restaurantes cercanos a la posición actual, y además, es posible realizar reservas en el momento. Como se puede observar en las diferentes aplicaciones mencionadas, los servicios provistos por éstas no siempre son los mismos, sino que cada una se focaliza en servicios gastronómicos particulares (reservas, deliverys, recomendaciones, información de sitios, entre otros), es decir, no hay una integración entre las distintas aplicaciones. Por otro lado, dichas aplicaciones tampoco cuentan con un modelo de solución para brindar servicios gastronómicos contextuales.

---

[1] Página de *Google Maps*: https://maps.google.com.ar (Último acceso 21-9-2017)

[2] Página de *Apple Maps*: https://www.apple.com/ios/maps (Último acceso 21-9-2017)

[3] Página de *Pokemon Go*: http:/www.pokemongo.com (Último acceso 23-9-2017)

[4] Página de *Uber*: https://www.uber.com (Último acceso 23-9-2017)

[5] Página de *Happn*: https:/www.happn.com (Último acceso 23-9-2017)

[6] Página de *Restorando*: https://buenos-aires.restorando.com.ar (Último acceso 23-9-2017)

Dentro del ámbito gastronómico, suele suceder que en ciertos momentos del día, al querer concurrir a un restaurante, se hace prácticamente imposible poder encontrar alguno con capacidad disponible. También, se presentan situaciones en las que no se conocen sitios cercanos para comer algún tipo de comida particular, o simplemente, no se tiene información sobre tipos de comidas, menúes y promociones que ofrecen los sitios gastronómicos. Otras veces, se tiene conocimiento sobre dónde se puede adquirir el servicio pero, al arribar al lugar, existe una demora excesiva para consumirlo. Cabe mencionar que, hoy en día existen diversos medios de comunicación cómo son las redes sociales *Facebook* e *Instagram*, y otros buscadores web donde es posible encontrar información sobre cualquier tema en cuestión, como *Google Search*. No obstante, resulta dificultoso encontrar un único medio en donde se provean servicios que involucran a la gastronomía, contemplando la disponibilidad en los restaurantes, avisos de llegada a los sitios por parte de los clientes, recomendaciones de lugares por cercanía y preferencias de los clientes, y toda aquella información relevante que pueda serle útil a una persona al momento de encontrar un lugar. Sería deseable, disponer de estos servicios gastronómicos, ya sean contextuales o no, en una única aplicación con la capacidad de reaccionar ante los cambios de contexto.

Una de las motivaciones principales de esta tesina, es proponer una solución de modelado donde se puedan integrar servicios contextuales en el ámbito gastronómico, ya que si bien existen modelos que brindan soluciones para aplicaciones sensibles al contexto [Bauer and Dey, 2016], ninguno de ellos está enfocado en resolver cuestiones del dominio gastronómico. Por ejemplo, en [Fortier et al., 2010] se provee un modelo de solución general para aplicaciones sensibles al contexto que brinda variabilidad tanto a nivel de diseño como en tiempo de ejecución. Sin embargo, al tratarse de un modelo general no se abordan características particulares del dominio gastronómico. Por otro lado, en [Musi Gentile, 2015] se presenta una solución de modelado de gestos considerando contexto, sin embargo el mismo está destinado a abordar características particulares de sensado de gestos y no contempla el concepto de servicio. En particular, la solución de modelo propuesto en esta tesina se focaliza en abordar características de los servicios contextuales en el ámbito gastronómico. Esta solución hará hincapié en integrar este tipo de servicio, con el fin de facilitar la construcción de aplicaciones gastronómicas sensibles al contexto.

1.2 Objetivo

El objetivo de ésta tesina es proponer una solución de modelado para brindar la integración de servicios gastronómicos contextuales. Donde estos servicios no solo tengan en cuenta la posición actual del usuario, sino además otros contextos relevantes del usuario como así también de los sitios gastronómicos involucrados, por ejemplo, promociones vigentes o estados de los pedidos.

Para diseñar la solución de modelado a proponer en esta tesina, primero se analizan las características de algunos sitios gastronómicos existentes y, por otro lado, soluciones de modelado brindadas por diferentes autores, que contemplen el concepto de contexto como una característica relevante de dicha solución. A partir de este análisis, se determinará qué características se deben considerar en una solución de modelado, como así también cuales de las soluciones existentes de modelado de contexto se ajustan mejor para ser usada de base en la solución de esta tesina.

Se busca que la solución de modelado propuesta en esta tesina permita desacoplar el contexto del modelo de dominio específico de la aplicación y, a su vez, obtener variabilidad en los cambios de contexto no solo en la etapa de diseño sino también en tiempo de ejecución. Esto permite contemplar la posibilidad de incorporar nuevos servicios y sensores, como así también nuevos contextos. Facilitando así, la evolución de las aplicaciones diseñadas [Weyns et al., 2015] y logrando que la solución de modelado de integración de servicios contextuales gastronómico sea escalable.

En base a la solución de modelado propuesta se desarrolló un prototipo denominado "*HoyNoCocino*" con el objetivo de implementar algunos servicios contextuales, y poder apreciar así el funcionamiento de los mismos. Este prototipo usa distintas herramientas de última generación como son, por ejemplo: HTML5 [HTML5, 2014], CSS3 [CSS, 1996], *Bootstrap* [Bootstrap, 2011], *JQuery* [jQuery, 2006].

En particular, el prototipo permite apreciar cómo funcionan distintos servicios contextuales como por ejemplo:

- visualizar los sitios gastronómicos cercanos a la posición actual de un usuario.
- notificar a un usuario si alguno de sus amigos se encuentra en un sitio gastronómico cercano a su posición actual.
- notificar al cliente de las promociones que ofrecen los diferentes sitios, según sus gustos y posición actual.
- notificar al cliente que su pedido se encuentra listo para ser retirado.

El prototipo permite apreciar cómo se comportan distintos contextos, donde algunos toman valores de sensores particulares, como es el contexto de posicionamiento. Mientras que otros valores de contextos son indicados por el usuario de manera explícita, por ejemplo, que un pedido está listo.

1.3 Organización de la tesina

La tesina se estructura como se detalla a continuación.

En el Capítulo 2 se describen los conceptos generales de contexto, también se detallan un conjunto de aplicaciones móviles que brindan servicios en el ámbito de sitios gastronómicos. Posteriormente, se describen arquitecturas y soluciones de modelados, con el fin de enunciar los diferentes enfoques los cuales son la base para la solución de modelado propuesta en esta tesina.

En el Capítulo 3 se realiza un análisis y descripción de la problemática a resolver. Se presenta un modelo de aplicación gastronómica, explicando su funcionamiento y objetivos. Este modelo inicial permite comprender las características de este tipo de aplicaciones. Luego, se explica el modelo de aplicaciones sensibles al contexto en el cual se usa de base para la solución de modelado final propuesta en esta tesina. Usando de base este modelo de aplicaciones sensibles al contexto, se refactoriza el modelo inicial para lograr una solución de modelado donde se desacoplen los conceptos de contexto de características propias del domino.

En el Capítulo 4 se describe la implementación del prototipo propuesto, realizando una descripción de las consideraciones establecidas durante el desarrollo. Este prototipo se plantea usando como base la solución final propuesta en el Capítulo 3.

En el Capítulo 5 se presentan ejemplos de uso del prototipo mostrando el paso a paso de cada uno de ellos mediante capturas de pantallas.

En el Capítulo 6 se enuncian las conclusiones obtenidas en relación a la tesina como así también se describen algunas de las futuras líneas de investigación que se pueden seguir a partir de esta tesina.

2. Background

En este capítulo se presentan conceptos generales de contexto. Se hace un relevamiento de servicios brindados por aplicaciones de sitios gastronómicos existentes, en particular, haciendo hincapié en aquellos servicios posicionados o contextuales. Luego, se describen distintas arquitecturas para aplicaciones sensibles al contexto. Finalmente, se presentan algunas soluciones de modelado para aplicaciones sensibles al contexto.

2.1 Conceptos Generales de Contexto

En esta sección se describirán conceptos generales sobre contexto como así también sobre aplicaciones sensibles al contexto.

El concepto de contexto se ha discutido durante los últimos veinte años, aún en la actualidad no se cuenta con una definición unificada. A continuación, se listan algunas perspectivas de diferentes autores para poder apreciar como cada uno de ellos considera este concepto:

- En [Schilit et al., 1994], se describe el contexto como *la información que puede ser usada para caracterizar la situación de una entidad*. Entendiendo por entidades, a las personas, posiciones u objetos que son relevantes para el comportamiento de una aplicación, y que por sí mismas, son consideradas como parte de su contexto. En particular, uno de estos autores define en [Schilit, 1994] contexto como: *ubicación del usuario, identidades de objetos y personas cercanas, así como los cambios a dichos objetos*.
- Por su parte, en [Brown, 1997] se considera al contexto como la *ubicación, identidad de las personas alrededor del usuario, la hora del día, temperatura y estación del año*.
- Para [Ward, 1997] el contexto es el *estado de las cosas alrededor de la aplicación*.
- En [Dey, 1998], se define como *el estado emocional del usuario, su foco de atención, su ubicación y orientación, la fecha, hora, las personas y objetos que se encuentran dentro del ambiente del usuario*.
- Por otro lado, en [Ryan et al., 1999] lo describen como la información del entorno que permite a las computadoras detectar y actuar en base a ella, tales como la *ubicación, medio ambiente, identidad de los usuarios y tiempo*.
- Así mismo, [Pascoe, 1998] lo define como el subconjunto de los *estados físico y conceptual de interés para una entidad en particular*.

Se puede apreciar que cada uno de estos autores considera el contexto desde diferentes perspectivas, acorde a esto, en [Abowd et al., 1999] los autores tratan de brindar una definición propia pensando en ayudar a los desarrolladores a determinar que es el contexto para una aplicación determinada, definiendo lo siguiente:

> *"Contexto es cualquier información que pueda ser usada para caracterizar la situación de una entidad. Una entidad es una persona, lugar u objeto que se considera relevante para la interacción entre un usuario y una aplicación, incluyendo el usuario y las propias aplicaciones"*

La definición presentada en [Abowd et al., 1999] facilita a los desarrolladores de aplicaciones a identificar el contexto para un escenario o dominio determinado. Si una información puede usarse para caracterizar la situación de un usuario en una interacción, entonces esa información puede considerarse contexto.

En la Figura 2.1.1 se puede apreciar una taxonomía sobre diferentes conceptos que pueden considerarse contexto, donde estos pueden abarcar desde características de los usuarios como así también del ambiente o de los sistemas en sí.

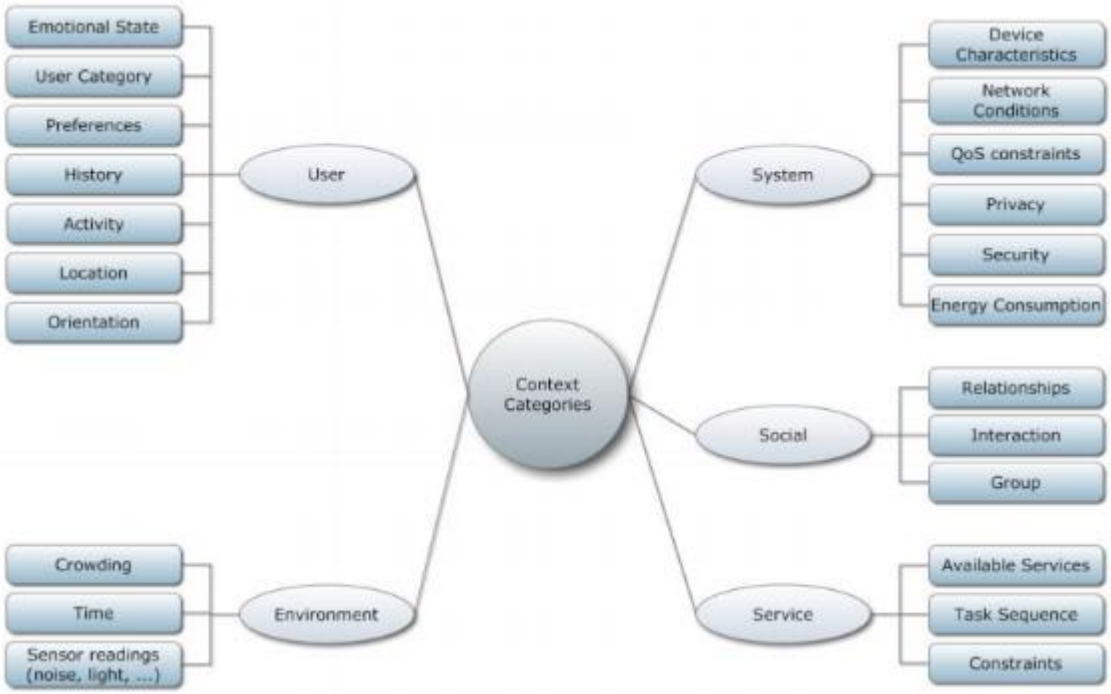


Figura 2.1.1: Taxonomía de Contextos propuesta en [Emmanouilidis et al., 2013]

Teniendo en cuenta que uno de los mayores retos dentro de la computación móvil es poder relacionarse con el entorno cambiante en el cual se ejecutan las aplicaciones y adaptarse rápidamente de acuerdo a las evoluciones tecnológicas, surge el concepto de aplicaciones sensibles al contexto. En [Schilit, 1994] se plantea este concepto de la siguiente manera:

> "*Las aplicaciones sensibles al contexto son aplicaciones que cambian dinámicamente o adaptan su comportamiento basándose en el contexto de la aplicación o del usuario*"

A lo largo de los años, se han adoptado diferentes enfoques para proporcionar una arquitectura común para las aplicaciones sensibles al contexto y la funcionalidad de los mecanismos de razonamiento contextual. Por lo general, estos enfoques se implementan en forma de *Middleware* o en forma de *Frameworks* de aplicación. En la Sección 2.3 se brindarán más detalles respecto de algunas soluciones existentes.

2.2 Aplicaciones Móviles en al Ámbito de Sitos Gastronómicos

En esta sección se presentarán algunas aplicaciones móviles destinadas al ámbito de los restaurantes. Esto permitirá comprender los servicios de las aplicaciones móviles existentes, en particular, se hará hincapié en describir los servicios contextuales que las mismas ofrecen.

A continuación se hace una descripción breve de algunas aplicaciones móviles de restaurantes:

- *Yelp* [Yelp, 2004], es una aplicación que fue diseñada para buscar recomendaciones sobre restaurantes, bares y deliverys contemplando filtros (como, por ejemplo, barrios, distancia, precio, hora de apertura). La información brindada por la aplicación está basada en las opiniones de otros usuarios que escriben reviews de los distintos lugares para que se pueda encontrar la opción que más se adapte a los planes del usuario. Además, también tiene una función adicional para hacer reservas y buscar opciones en el lugar donde esté posicionado actualmente el usuario. Provee información de promociones que los negocios favoritos ofrecen y vínculo con redes sociales.

- *GoChef* [GoChef, 2016], brinda la posibilidad de poder navegar y ver los menús de cocina local de cocineros certificados. Permite que cada cocina cualificada funcione como su propio local. Posee la funcionalidad de navegar por las cocinas locales, ver los menús, enviar un mensaje a un chef para que personalice la comida que el usuario comprará. Contemplando los servicios que provee, se encuentran: reservas, pago online, notificaciones cuando su comida está lista, notificaciones de ofertas, compartir en redes sociales y calificaciones.

- *PedidosYa* [PedidosYa, 2009], provee un buscador de servicios de delivery cerca de la posición actual del usuario. En esta aplicación, se pueden ver todas las opciones de los lugares de comida, como menús detallados, precios y medios de pago. Permite seleccionar el lugar donde se va a realizar el pedido de comida, para luego ser enviado a la ubicación del usuario por el restaurante seleccionado. Además, se pueden ver comentarios y puntuaciones de otros usuarios, agregar notas a los pedidos y repetir los últimos pedidos de comida.

- *Restaurantes.com* [Restaurantes.com, 2016], es una aplicación donde se puede realizar búsquedas de restaurantes cercanos a la ubicación del usuario y efectuar reservas. También ofrece filtros de búsqueda, agregar restaurantes favoritos y compartir actividad mediante email, *SMS*, *Whatsapp*, *Facebook*, *Twitter* o *Google+*. Además, se pueden gestionar opiniones propias sobre restaurantes y ver las de los demás usuarios.

- *ElTenedor* [ElTenedor, 2011], aplicación líder en España para reservar lugares en restaurantes. Posibilita la reserva, brinda información detallada de cada restaurante, posee opiniones reales de la comunidad, facilita búsqueda de tipos de restaurante y notifica sobre promociones cercanas. Permite compartir fotos de platos y por cada reserva realizada se acumulan puntos con los que se pueden obtener descuentos.

En la Tabla 2.2.1 se presenta un resumen de los servicios provistos por las aplicaciones móviles mencionadas anteriormente. Este resumen se realizó acorde al análisis detallado de cada una de estas aplicaciones móviles.

Tabla 2.2.1: Resumen de los servicios provistos por las aplicaciones móviles analizadas

		APLICACIONES				
		Yelp	GoChef	PedidosYa	Restaurantes.com	ElTenedor
SERVICIOS	Visualización de lugares según la ubicación	✓	✓	✓	✓	✓
	Búsqueda con filtros de distancia, precio y horarios de apertura	✓	-	✓	✓	✓
	Notificar por promociones	✓	✓	✓	✓	✓
	Lista de favoritos	✓	-	✓	✓	-
	Añadir reseñas, sugerencias, fotos y calificaciones	✓	✓	✓	✓	✓
	Compartir en redes sociales	✓	✓	-	✓	-
	Realizar reservas	-	✓	-	✓	-
	Realidad aumentada	✓	-	-	-	-
	Seguir y ver actividad de amigos	✓	-	-	-	-
	Descripción menú y especialidades	-	✓	✓	✓	✓
	Sumar puntos para obtener descuentos	-	-	-	-	✓
	Suscripción Newsletter	-	-	-	-	✓
	Notificación de inasistencia al lugar	-	-	-	-	✓
	Notificación de pedido listo	-	✓	-	-	-
	Solicitar delivery de pedidos	✓	✓	✓	-	-
	Otros servicios no gastronómicos	✓	-	-	-	-
	Personalizar menú y realizar pedido	-	✓	-	-	-
	Como llegar al lugar	✓	-	-	-	-
	Pago virtual	-	✓	-	-	-

A partir de la Tabla 2.2.1, y del análisis realizado, se puede destacar algunos servicios o información faltantes en estas aplicaciones móviles analizadas, por ejemplo:

- *Yelp* no posee reservas directas, sino que solo ofrece el número telefónico para su gestión.

- *Go chef* ofrece variados servicios gastronómicos en sitios donde se encuentran cheffs particulares, pero no contempla otros tipos de empresas gastronómicas como restaurantes, cervecerías, etc.

- *PedidosYa* es utilizada para realizar pedidos de delivery cerca de la posición del usuario, pero no cubre otros servicios como, por ejemplo, reservas en sitios.

- *Restaurantes.com* y *El Tenedor*, no proveen servicio de delivery ya que están focalizadas en brindar servicios orientados a la visita de restaurantes.

Se puede apreciar cómo cada una de las aplicaciones móviles analizadas proveen diferentes servicios, sin embargo, estos servicios podrían combinarse entre las distintas aplicaciones, para proveer así, servicios más completos.

Otras aplicaciones que fueron relevadas para el dominio gastronómico se presentan en el Anexo A, donde se destacan los servicios principales que estas proveen, dado que no incorporan nada significativo respecto de las aplicaciones analizadas en la Tabla 2.2.1, se decidió detallarlas en un anexo.

A continuación se presentará un análisis de los servicios contextuales que se detectaron a partir de las aplicaciones móviles detalladas anteriormente. Para lograr una mejor clasificación, dividimos los servicios en aquellos que se determinan a partir del posicionamiento, y por otro lado, los que involucran además otros contextos.

Los servicios basados en el posicionamiento que se detectaron, en las aplicaciones analizadas, son los siguientes:

- *Búsqueda de sitios cercanos a la posición actual del usuario.*

- *Proveer un camino desde la posición actual del usuario hasta un sitio gastronómico mediante el uso de un mapa.*

Cabe destacar que si bien todas proveen la búsqueda de sitios cercano a la posición actual del usuario, solo una provee el servicio de buscar un camino, como se puede apreciar en la Tabla 2.2.1 detallada anteriormente.

Como se mencionó anteriormente, los dos servicios descriptos son basados en posicionamiento. Estos son contextuales ya que en este caso la posición es un contexto relevante. Además, se detectaron otros servicios contextuales, algunos de estos consideran no solo la posición sino otros contextos relevantes. Estos servicios se pueden apreciar en la Tabla 2.2.2. En la tabla se puede apreciar otros servicios contextuales que no consideran la posición actual del usuario, sino que contemplan otros contextos, por ejemplo, las fecha y hora.

Tabla 2.2.2: Análisis de servicios contextuales

SERVICIOS CONTEXTUALES	CONTEXTO INVOLUCRADOS	APLICACIONES QUE LO BRINDAN
Notificar por promociones	Ubicación y Lugares favoritos	5
Brindar información de los restaurantes a través de realidad aumentada	Ubicación y Orientación	1
Notificaciones pedidos listo	Fecha/hora y Estado del Pedido	1
Notificaciones de inasistencia al lugar	Fecha/hora	1
Seguir y ver actividad de amigos	Lista de Amigos (actividad actual de cada uno)	1

Como se puede apreciar en la Tabla 2.2.2, si bien se detectan servicios contextuales, luego estos generalmente no son fácilmente puestos en práctica. Esto se debe a que el diseño de los mismos puede resultar complicado, en esta tesina nos focalizaremos en brindar una solución de modelado que sirva de base para agilizar así el diseño de los mismos.

Las aplicaciones móviles analizadas cuentan con otros servicios que no se consideran contextuales, estos no se abordarán ya que escapan del foco de esta tesina.

De esta manera se pudo apreciar como distintos contextos son considerados para brindar diferentes servicios. Cabe destacar, que estas aplicaciones no cuentan con un modelo de solución asociado.

2.3 Arquitecturas de aplicaciones sensibles al contexto

A lo largo de los años se han propuesto diferentes enfoques para proporcionar arquitecturas para aplicaciones sensibles al contexto. Por lo general, estas se presentan como arquitecturas *Middleware*[7] o también como *Frameworks* (generalmente conceptuales). A continuación se presentan algunas de estas arquitecturas:

- *Framework Conceptual Context Toolkit [Dey et al., 2001]*

  En [Dey et al., 2001] se presenta un framework conceptual para aplicaciones sensibles al contexto. Cabe destacar que este trabajo es uno de los que sientan las bases en la temática de este tipo de aplicaciones. Relacionados a este tipo de aplicaciones, se detectan en [Dey et al., 2001] tres principales problemas de las aplicaciones sensibles al contexto:

  - *abstracción de la programación de alto nivel*
  - *datos de contextos ambiguos*
  - *control de acceso*

  El framework conceptual propuesto en [Dey et al., 2001] es muy sencillo pero capaz de incorporar nuevos componentes en tiempo de ejecución. Consiste en tres abstracciones principales: *widgets*, *agregadores* (*aggregator*) e *intérpretes* (*interpreter*). En la Figura 2.3.1 se puede observar un ejemplo de ésta arquitectura. En este caso las flechas indican el flujo de datos.

[7] *Middleware* o lógica de intercambio de información entre aplicaciones es un software que conecta componentes de software o aplicaciones para que puedan intercambiar datos entre éstas. Es utilizado a menudo para soportar aplicaciones distribuidas. Esto incluye servidores web, servidores de aplicaciones, sistemas de gestión de contenido y herramientas similares.

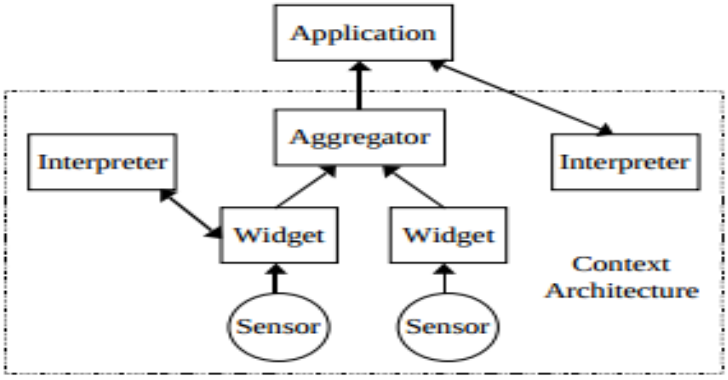


Figura 2.3.1: Ejemplo de los componentes del Framework conceptual [Dey & Abowd, 2000]

A continuación se describen cada una de las abstracciones presentadas en la Figura 2.3.1. Los *Widgets* de contexto son mediadores entre un usuario y su entorno. Encapsulan información sobre una sola pieza de contexto, como por ejemplo la posición o actividad. Proporcionan una interfaz uniforme a los componentes o aplicaciones que utilizan el contexto, ocultando los detalles de los mecanismos de detección del contexto. Evitan al programador tener que involucrarse con la complejidad del sensor usado para la aplicación. Por ejemplo, si se sensa la posición de un usuario dentro de un edificio, usando tarjetas magnéticas y se cambia por un sistema de sensado Wifi, este cambio no debería provocar ninguna alteración en el comportamiento de la aplicación, dado que dicha abstracción la encapsularía el *widget*. A su vez, los *widgets* abstraen esta información y la proveen según las necesidades de la aplicación. En el ejemplo de sensado dentro de un edificio, el *widget* notificará cuando el usuario pase de una habitación a la otra.

La responsabilidad de los *Agregadores* reside en agrupar los diferentes *widgets* en un solo componente de software, ya que los *widgets* pueden estar distribuidos. Así, cada *agregador* puede facilitar a la aplicación recuperar el contexto completo de una entidad de una manera simple y reutilizable. Al actuar como una puerta de enlace entre las aplicaciones y los *widgets* elementales, los *agregadores* ocultan incluso más complejidad sobre los mecanismos de detección del contexto.

Usualmente las aplicaciones necesitan un nivel mayor de abstracción del contexto, más allá de los valores que proveen los sensores o los *widgets*. Por ejemplo, las coordenadas geográficas pueden ser traducidas a nombres de calles o avenidas usando alguna información geográfica del lugar, o se puede inferir nueva información en base a datos más simples, por ejemplo, el ruido ambiente elevado puede llevar a la deducción de que el usuario se encuentra en una reunión. Los *Intérpretes* son los responsables de llevar a cabo esta abstracción. Generalmente toman datos de uno o más *widgets* y lo transforman en información más abstracta para la aplicación.

Por otro lado, en [Dey et al., 2001] hay otros conceptos como son los *Servicios* y los *Descubridores*. Los *servicios* tienen la responsabilidad de llevar a cabo la acción vinculada al contexto detectado según la lógica de la aplicación. Las implementaciones de estos *servicios* son, generalmente, acciones recurrentes en varias aplicaciones, como, por ejemplo, mostrar un mensaje. Los *Servicios* facilitan el desarrollo de aplicaciones dado que, al igual que los *widgets*, no es necesario que el desarrollador

sea consciente de la implementación del mismo y pueda acceder a acciones comunes de manera instantánea. Por otro lado, los *descubridores* tienen como tarea mantener un registro de todos los componentes existentes. Esto es, conocer qué *widgets*, *interpretadores*, *agregadores* y *servicios* están disponibles, notificar ante la aparición o falla de uno de ellos, y proveer la capacidad de buscar entre todos por algún criterio. Las aplicaciones utilizan estos *descubridores* para encontrar los componentes que le son de interés y abstraen a la misma de saber en dónde se encuentra cada uno de estos.

Si bien la arquitectura genérica para los sistemas sensibles al contexto presentada en [Dey et al., 2001] fue un avance para su época, la misma no cubre algunos aspectos que son relevantes para este tipo de aplicaciones. A continuación se presentan algunos de los aspectos detectados a partir de la bibliografía relevada.

- *No existe una noción explícita de un modelo de contexto*

  Acorde a lo descripto en [Dey et al., 2001], el diseñador se encarga de modelar el contexto basado en los *widgets* y *agregadores* disponibles. Esto tiene como ventaja ser extremadamente flexible, pero esa flexibilidad tiene un costo. Aunque en [Dey et al., 2001] se destaca la separación entre datos de sensores e información de contexto, derivar en un modelo de contexto desde la información de los *widgets* no es trivial. Este problema no es nuevo y se puede hacer un paralelismo con programadores inexpertos y la falta de directrices de diseño, donde las aplicaciones se escriben colocando *widgets* en una ventana y luego escribiendo sus manejadores de acciones (por ejemplo, el evento *onClick*). Este estilo de programación ha producido infinidades de programas en los que no hay un modelo de dominio explícito, sino una colección de procesos dispersos y entrelazados a través de la aplicación.

- *No hay un soporte arquitectónico para separar la lógica de la aplicación de la lógica de adaptación*

  Un *servicio* en el framework conceptual presentado en [Dey et al., 2001], se encarga de realizar acciones basadas en cambios de contexto que pueden abarcar tanto la adaptación como la lógica de la aplicación. Esto tiene como desventaja la escalabilidad limitada de la aplicación.

- *Arquitectura Middleware [Duran-Limon et al., 2003]*

  Una arquitectura *Middleware* es presentada en [Duran-Limon et al., 2003]. Ésta arquitectura introduce entidades especiales, llamados *objetos sensibles*, que son responsables de recibir, procesar y proporcionar información relacionada con el contexto. Los *objetos sensibles*, son definidos como objetos autónomos que son capaces de percibir su entorno y actuar en consecuencia. Una de las ventajas de éste enfoque es la posibilidad de reorganizar esos objetos dependiendo de su tarea principal.

  La comunicación entre los objetos sensibles, sensores (Sensor) y actuadores (Actuator) se realiza mediante un mecanismo basado en eventos que se establece dinámicamente durante la operación del sistema. Dicha comunicación se lleva a cabo utilizando el modelo de ejecución evento-condición-acción. Los objetos sensibles que forman parte de ésta arquitectura cuentan con dos interfaces:

- Sensado de eventos percibidos por los sensores físicos (Sensor).
- Emisión de eventos para adaptarse al contexto actual (Actuator).

En la Figura 2.3.2 se puede apreciar la arquitectura middleware presentada en [Duran-Limon et al., 2003]. En dicha figura se pueden apreciar los *objetos sensibles*, como así también los *Sensor* y *Actuator.*

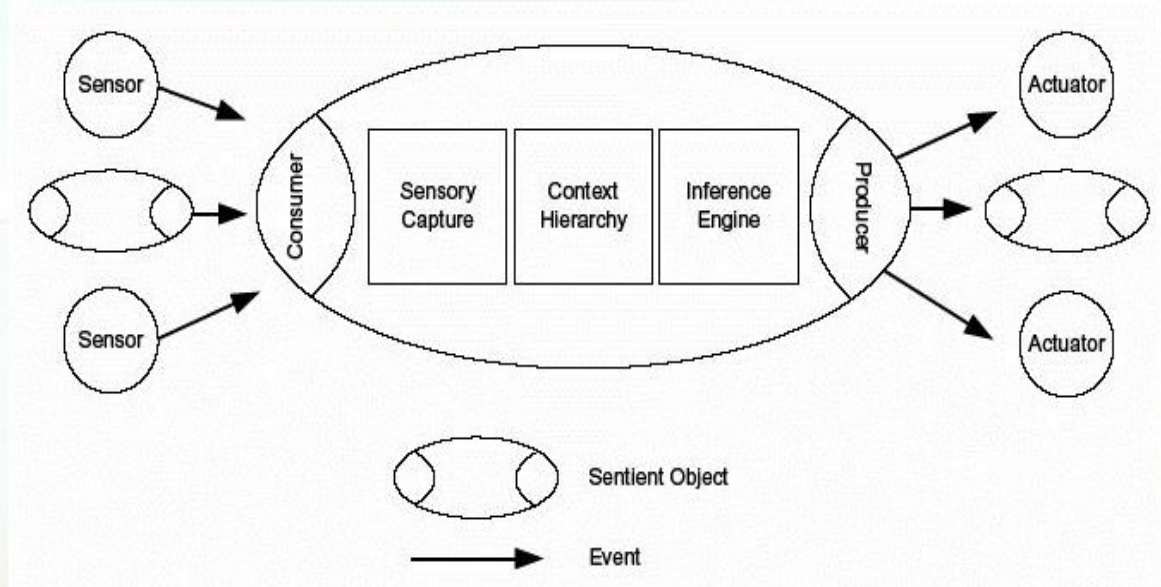


Figura 2.3.2: Arquitectura Middleware [Duran-Limon et al., 2003]

- *Framework para Aplicaciones Sensibles al Contexto [Bazzocco, 2005]*

  Un framework orientado a objetos es planteado en [Bazzocco, 2005], este presenta guías de diseño y soluciones generales a los tipos de problemas usualmente hallados al construir aplicaciones sensibles al contexto.

  El objetivo principal del framework presentado en [Bazzocco, 2005], es brindar servicios de más alto nivel y valor agregado a las distintas aplicaciones que utilicen información contextual para su funcionamiento. De esta manera, se puede liberar a los desarrolladores de realizar una y otra vez tareas de detección y manipulación de información contextual en forma manual, realizándolo de una manera más estándar, flexible y potente.

  Resulta importante aclarar que este framework es utilizado por las aplicaciones en forma local, es decir, que existe en forma de librería que debe incorporarse en el desarrollo para poder hacer uso de sus funciones. En otras palabras, este framework se ejecuta en el ambiente provisto por un dispositivo móvil.

  En la Figura 2.3.3 se puede apreciar un esquema conceptual de los diferentes elementos que componen el framework presentado en [Bazzocco, 2005]. En dicha figura se pueden observar diferentes conceptos como son: *Context Manager*, *Context*

Listeners, *Device Adapter*, *Applications*, *Devices* y *Events*. Estos conceptos se detallaran más adelante.

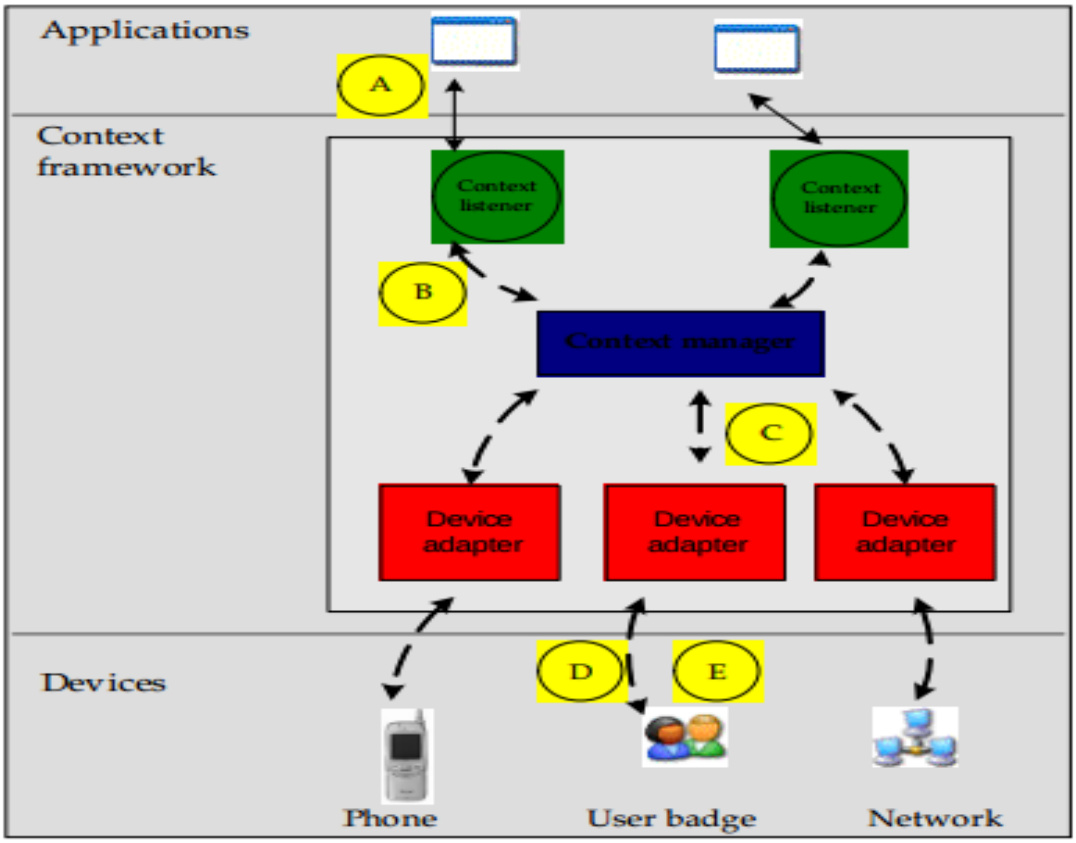


Figura 2.3.3: Principales elementos del Framework presentado en [Bazzocco, 2005]

A continuación se presenta una breve descripción de los conceptos presentados en la Figura 2.3.3:

- *Context Manager*: este elemento es responsable del mantenimiento de los diferentes contextos requeridos por las múltiples aplicaciones que se pueden estar ejecutando en el dispositivo móvil. Cada vez que se da un cambio en algunos de los contextos, arbitra los medios necesarios para notificar a las aplicaciones que correspondan. Su implementación se basa en el patrón de diseño *Singleton* [Gamma et al., 1995] ya que es conveniente asegurarse que en todo momento exista una única instancia de esta clase actuando como "*manejador*" de la información obtenida recientemente.
- *Context Listeners*: estos elementos tienen como principal responsabilidad el notificar a las aplicaciones de los cambios en el contexto. Una aplicación debe registrarse para "escuchar" por un cambio de contexto en particular. Una vez que se produce el cambio esperado, el *listener* enviará una notificación adecuada a la aplicación que lo ha registrado.
- *Device Adapter*: es necesario contar con una capa cuya responsabilidad sea estandarizar el formato de la información recibida desde los dispositivos físicos. Esta capa está conformada por los *adaptadores*, los cuales traducen la

información obtenida a partir de un dispositivo físico a un formato estándar interno del framework.

- *Applications*: estas son las que serán ejecutadas en los dispositivos móviles y serán las que se beneficien de la obtención de la información contextual. Cada *aplicación* deberá registrar su interés (a través de los correspondientes *listeners*) a los diferentes cambios de contexto que puedan ser detectados por el framework. Cabe destacar que cada una de estas *aplicaciones* será responsable de definir un modelo de objetos que se nutra de la información contextual básica enviada a partir del framework.
- *Devices*: estos elementos representan las fuentes a partir de las cuales se obtendrá la información acerca del contexto. Cabe aclarar que la información obtenida a través de este medio es solamente información sobre aspectos físicos, no así la de carácter lógico.
- *Events*: representan un flujo conceptual de la información que en la Figura 2.3.3 se identifican como *A*, *B*, *C*, *D* y *E*. Dicha información está encapsulada en ciertas entidades las cuales sirven como repositorio de cierta información, por ejemplo:
  - Origen de la información capturada, es decir, desde qué dispositivo se ha obtenido la información.
  - Información propiamente dicha, por ejemplo, la ubicación.
  - Precisión o calidad de la información obtenida para la captura en particular.

- *Arquitectura para aplicaciones sensibles al contexto [Fortier et al., 2010]*

  En [Fortier et al., 2010] se describe una arquitectura abstracta basada en el contexto, que contempla los conceptos de todos los dominios, pero que puede configurarse para un dominio especifico. Esta arquitectura define:

  - Un conjunto de abstracciones micro-arquitectónicas que están presentes en la mayoría de las aplicaciones sensibles al contexto (a través de los dominios). Estas abstracciones son independientes del dominio (tanto en términos de aplicación como de adaptación) y como resultado, son reutilizables en muchas situaciones diferentes. En otras palabras, representan los módulos que ayudan en el desarrollo de cualquier sistema sensible al contexto, independientemente de su dominio de aplicación o adaptación.
  - Luego, a partir del conjunto de abstracciones, se puede desarrollar aplicaciones concretas en un dominio de adaptación específico. Desde el punto de vista arquitectónico, esta fase implica la unión de los puntos de variación con una variante específica.

  La arquitectura presentada en [Fortier et al., 2010] facilita reutilizar y extender diferentes características (módulos). En la Figura 2.3.4 se puede apreciar la arquitectura presentada en [Fortier et al., 2010].

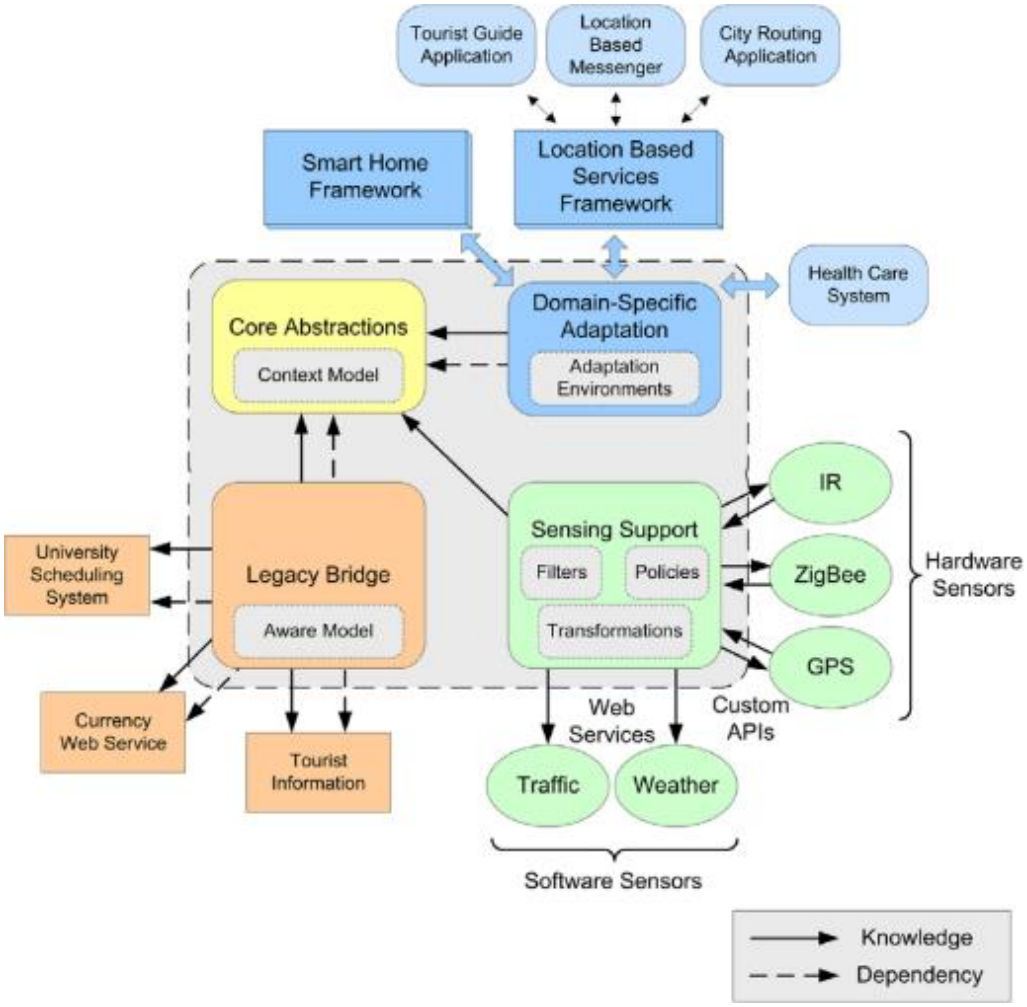


Figura 2.3.4: Arquitectura propuesta en [Fortier et al., 2010]

A continuación se presenta una breve descripción de los módulos de la arquitectura presentada en la Figura 2.3.4:

- *Core Abstractions*: este módulo abarca las estructuras básicas para manejar el contexto y coordinar la comunicación a través de diferentes paquetes. Aunque estas abstracciones pueden extenderse por especialización, se usan la mayor parte del tiempo como cajas negras, es decir, por medio de la composición.

- *Legacy Bridge*: este módulo proporciona una extensión al módulo principal para conectarlo a las aplicaciones existentes. Brinda facilidades para mejorar las aplicaciones "estándar" con un comportamiento sensible al contexto. Vale la pena notar que, si bien todas las abstracciones arquitectónicas pueden implementarse en la mayoría de los lenguajes orientados a objetos, este paquete particular depende en gran medida de las capacidades reflexivas del lenguaje de programación escogido.

- *Sensing Support*: este módulo se encarga de conectar dispositivos de sensado con las abstracciones de núcleo de tal manera que el proceso de detección se vuelva transparente al modelo de contexto. El módulo maneja las políticas de detección, el filtrado de señales y la transformación entre datos de bajo nivel e información de contexto de alto nivel.

- *Domain-Specific Adaptation*: las abstracciones de este módulo proporcionan el esqueleto para definir comportamientos específicos del dominio. Los desarrolladores se concentran en este paquete al diseñar una nueva aplicación o framework. En este caso, amplían las clases existentes, utilizando la abstracción similar a la de la caja blanca (es decir, mediante la especialización de componentes).

2.4 Soluciones de modelado para Aplicaciones Sensibles al Contexto

En esta sección se presenta algunos modelos de aplicaciones sensibles al contexto existentes. Cabe mencionar que en la bibliografía existen más arquitecturas generales (como las descriptas en la Sección 2.3) que soluciones de modelados. A continuación se presentan dos soluciones de modelado.

❖ *Modelo presentado en [Bazzocco, 2005]*

Una solución de modelado es presentada en [Bazzocco, 2005]. Esta solución se presenta como un framework orientado a objetos para la detección de cambios contextuales de manera genérica y que puede ser utilizado en el desarrollo de aplicaciones móviles. La intención fundamental de este framework, es liberar a los programadores de la tediosa tarea de detectar y definir como se manejan los cambios contextuales. Dicho framework define una jerarquía de clases de contextos como se puede observar en la Figura 2.4.1.

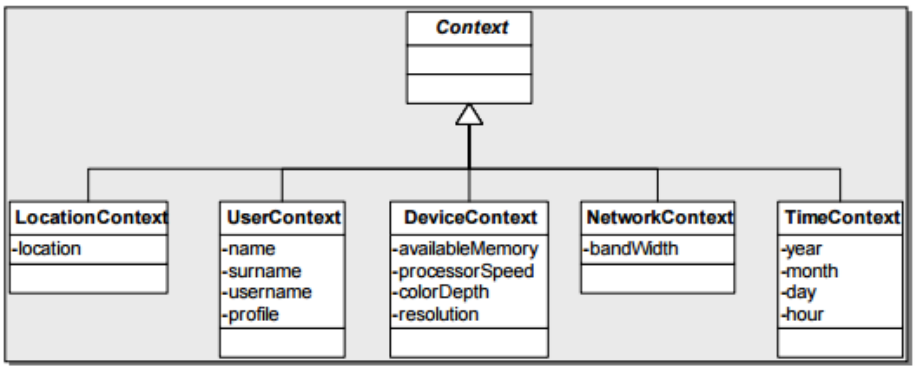


Figura 2.4.1: Jerarquía de Clases de Contextos [Bazzocco, 2005]

A continuación se describen las características de las clases presentadas en la Figura 2.4.1:

- *Context*: es la raíz de la jerarquía de contextos. Establece el protocolo que las subclases deben implementar y les brinda comportamiento en común a las mismas. Es el punto de partida para la creación de una jerarquía polimórfica que no imponga conocimiento en particular de cada subclase concreta a los objetos clientes.

- *LocationContext*: contiene información acerca de la ubicación del usuario ya que es la subclase responsable de modelar el comportamiento relacionado con la ubicación del mismo. Esta posición será obtenida a partir del sensado de algún

tipo de dispositivo físico de ubicación como celulares o GPS.

- *UserContext*: esta subclase representa toda la información personal relevante del usuario. Cada vez que se requieran datos acerca del usuario serán las instancias de esta clase las que proveerán acceso a dicha información.
- *DeviceContext*: esta clase permite almacenar todos los datos relacionados con el dispositivo que el usuario esté utilizando al momento de ejecutar las aplicaciones.
- *NetworkContext*: esta clase es responsable de manejar toda la información y los aspectos a tener en cuenta sobre el estado de la red, a través de la cual el usuario envía y recibe datos.
- *TimeContext*: las instancias de esta subclase serán las encargadas de mantener la información relacionada con el tiempo. Éste contempla diferentes características, tales como el instante en el que se ejecuta la consulta por parte de la aplicación cliente; la restricción de tiempo de respuesta en la ejecución del servicio solicitado y el período durante el cual se debe ejecutar dicho servicio; momento del día a partir del cual se definen los servicios disponibles para el cliente.

Además de la jerarquía presentada en la Figura 2.4.1, el framework está compuesto por otros elementos como se puede observar en la Figura 2.4.2. Estos elementos guardan relación con la arquitectura previamente presentada en la Sección 2.3, Figura 2.3.3.

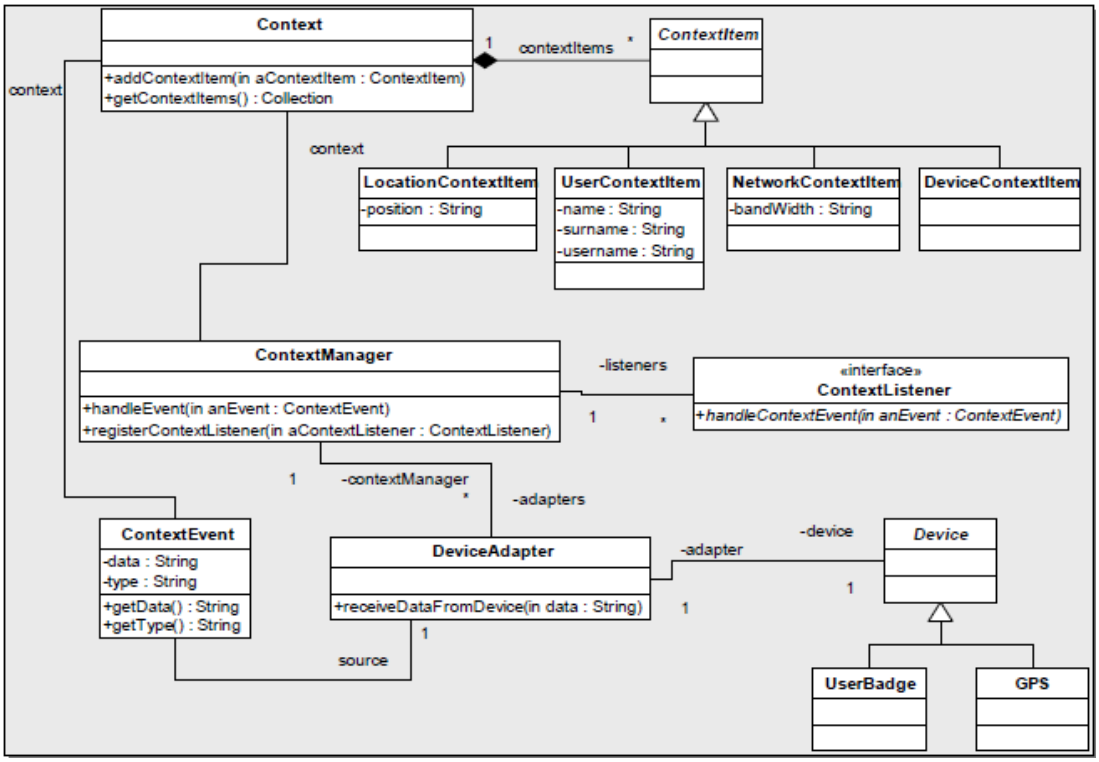


Figura 2.4.2: Diagrama de clases del Framework [Bazzocco, 2005]

En la Figura 2.4.2 se observa como la jerarquía de contextos forma parte del framework. También se pueden observar las clases que guardan relación con la arquitectura presentada en la Sección 2.3, Figura 2.3.3. Estas clases previamente mencionadas para la arquitectura son: *Context Manager (ContextManager), Context Listeners (ContextListener), Eventos (ContextEvent), Adaptadores (DeviceAdapter) y Dispostivos (Device).* El framework presentado en [Bazzocco, 2005] permite ser extendido para soportar nuevos tipos de dispositivos como también integrar contextos lógicos para manejar de manera estándar toda la información contextual. A su vez, se aclara que es responsabilidad exclusiva de las aplicaciones definir la forma en la que se mapean las instancias de la jerarquía de contextos con los objetos propios del dominio de la aplicación.

- *Modelo presentado en [Fortier et al., 2007]*

Otra solución de modelado para aplicaciones sensibles al contexto, es el presentado en [Fortier et al., 2007]. Este modelo guarda relación con la arquitectura previamente presentada en la Sección 2.3, Figura 2.3.4. Cabe mencionar que esta arquitectura provee una separación entre el contexto con el dominio específico de la aplicación. De esta manera, se logra obtener una independencia entre la implementación del dominio de la aplicación y la capa de contexto. Dicha capa, contiene las clases encargadas del contexto y es adaptable a toda la aplicación a través de la sub-clasificación de las mismas, teniendo como resultado, un framework de caja blanca. Este framework se basa en actuar sobre eventos, que se accionan dependiendo de los cambios en el contexto. El funcionamiento que comienza desde el cambio de contexto, hasta la realización de la acción correspondiente, se logra cubrir con las clases que componen la capa de contexto. En la Figura 2.4.3 se presentan las principales clases de la capa de contexto.

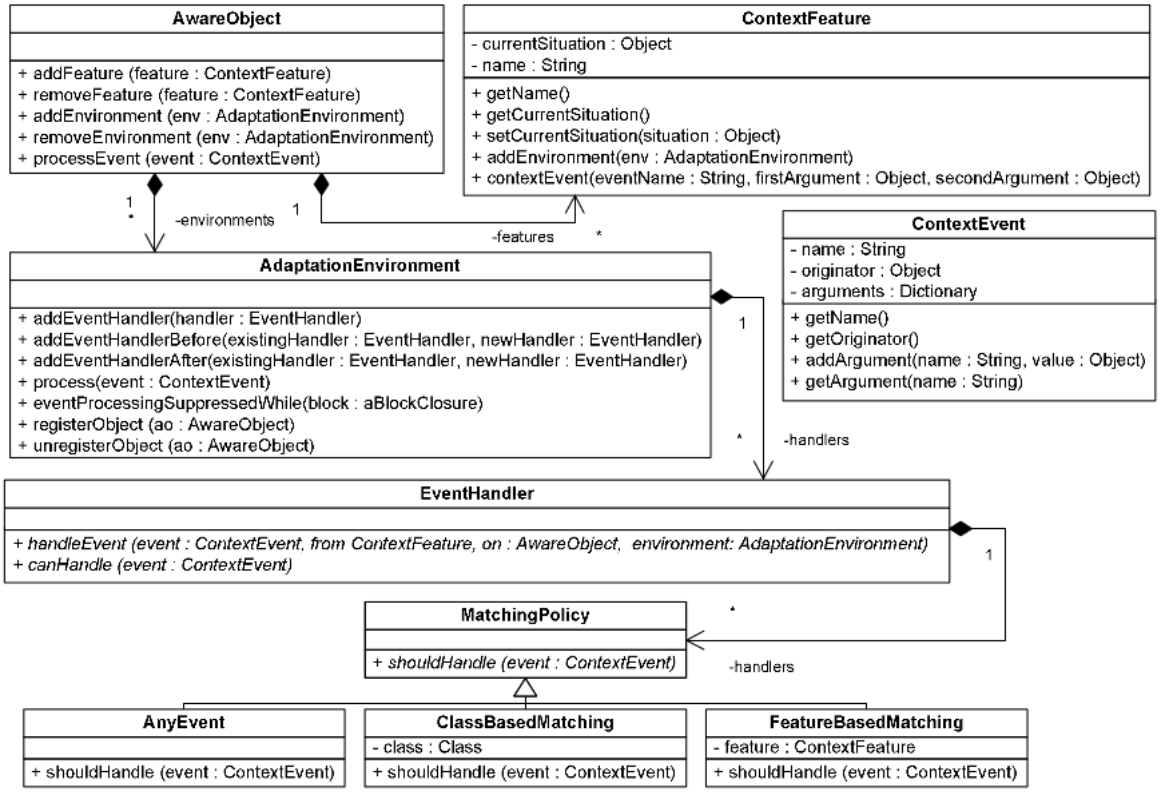


Figura 2.4.3: Modelo de contexto [Fortier et al., 2007]

A continuación se describen las características de las clases presentadas en la Figura 2.4.3:

- *AwareObject:* un *AwareObject* es un objeto de aplicación estándar (por ejemplo, una casa, un restaurant, individuo, etc.) que se ha mejorado con la capacidad de realizar un seguimiento de las características de contexto que son relevantes para él en un momento dado en el tiempo. Un *AwareObject* es un observador de cada característica que posee el contexto, de modo que cualquier cambio en su contexto puede ser capturado y procesado. Para esto, cuenta con un conjunto de *ContextFeatures* que notifican los cambios que se producen en ese objeto del dominio.

- *ContextFeature:* los *AwareObject* se basan en una colección de *ContextFeatures* que, como su nombre sugiere, modelan el contexto relevante. Un *ContextFeature* representa una característica de un objeto del modelo puro de dominio y tiene su "*valor*" actual, que suele ser una vista de alto nivel de la información captada por los sensores. También, como ya se destacó, se encargan de notificar al *AwareObject* adecuado cuando dichos valores cambian.

- *AdaptationEnvironment:* el framework proporciona un conjunto de hot-spots para brindar respuestas de comportamiento diferentes (personalizadas). Uno de estos hot-spot es la clase *AdaptationEnvironment*, utilizada para crear diferentes tipos de mecanismos de adaptación. El *AwareObject* debe estar registrado en uno o más *AdaptationEnvironment* dependiendo de la adaptación que necesita.

- *EventHandlers:* cuando un *ContextFeature* cambia, activa un evento de cambio de contexto, que es capturado por el *AwareObject*. Como respuesta a este

evento, el *AwareObject* lo reenvía a todos los *AdaptationEnvironment* a los que está registrado, para que puedan realizar sus comportamientos específicos. Un *EventHandler*, que está registrado en un entorno, recibirá un mensaje cada vez que se desencadena un evento de contexto. Un *EventHandler* puede especificar ser llamado sólo cuando un determinado aspecto activa el evento (por ejemplo, cambio de ubicación) y/o cuando es disparado por un *AwareObject* específico (esto es útil cuando muchos *AwareObjects* comparten un entorno común.

- *MatchingPolicy:* un *EventHandler* decide si un evento le es de interés o no, a través del uso de la clase *MatchingPolicy*. Algunas implementaciones pueden decidir sobre la clase del evento disparado, sobre el *AwareObject* que lo disparó, o simplemente tratar cualquier evento.

❖ *Comparación entre los modelos presentados en [Bazzocco, 2005] y [Fortier et al., 2007]*

Como se mencionó a lo largo de la Sección 2.4, tanto el modelo presentado en [Bazzocco, 2005] como en [Fortier et al., 2007] están orientados a brindar una solución de modelado a la problemática de las aplicaciones sensibles al contexto. A continuación se describe un análisis y comparación de ambos modelos. En particular, se observaron algunas diferencias.

- Una de ellas, es la manera en que se representan, en los respectivos modelos de clases, los distintos tipos de contextos que la aplicación puede contemplar. En [Fortier et al., 2007] el contexto a sensar, cualquiera sea el tipo, es una instancia de la clase *ContextFeature*, la cual como se explicó, contiene el valor actual observado del contexto. En cambio, en [Bazzocco, 2005] el contexto sensado es una instancia de una de las subclases de la jerarquía presentada en la Figura 2.4.1. Esto último tiene como desventaja, que en el caso de surgir un nuevo tipo de contexto que no pertenece a ninguna de las subclases de dicha jerarquía, es necesario agregar una nueva clase para contemplarlo en la aplicación. Es decir, el modelo presentado en [Fortier et al., 2007] permite la flexibilidad para poder representar con la clase *ContextFeature* cualquier contexto.

- Otra distinción importante que está relacionada con la diferencia anteriormente planteada, es que el modelo de [Fortier et al., 2007] presenta mayor flexibilidad en cuanto a las características del contexto sensado. Esto se debe a que se pueden agregar o remover *ContextFeature* en tiempo de ejecución para adaptarse al contexto actual. Por esta razón, es que se trata a las *ContextFeature* como puntos de variación que puede ser reevaluado en tiempo de ejecución.

- Con respecto a la gestión del sensado, se puede observar que en el modelo de [Bazzocco, 2005] se permite agregar nuevos dispositivos a través de la clase *Device*, pero para esto, es necesario instanciar y configurar una nueva instancia de *DeviceAdapter* que interprete los datos recibidos de dicho dispositivo.

Se pudo apreciar en la Sección 2.4 dos modelos ([Bazzocco, 2005] y [Fortier et al., 2007]) orientados a brindar una solución al modelado de aplicaciones sensibles al contexto. Estos modelos son generales, y servirán de base para en esta tesina brindar un modelo de solución, en particular, en el ámbito de sitios gastronómicos.

3. Modelo Propuesto

En este capítulo se presenta la descripción de la problemática a la que se quiere brindar una solución de modelado. En base a esta problemática, se presentará el modelo propuesto. Se brindará una primera solución de modelado sin considerar características desacopladas de contexto, dando luego lugar a un modelo más flexible el cual incorpora los conceptos de contexto de una manera desacoplada.

3.1 Descripción de la Problemática

En esta sección se describirán aquellas características destacadas relacionadas a las aplicaciones móviles en el dominio gastronómico, para brindar en particular servicios contextuales. A partir del análisis realizado en la Sección 2.2 se presentarán a continuación los conceptos identificados relacionados con los sistemas gastronómicos.

Un sistema gastronómico debería contar por un lado con las personas registradas en el mismo como así también los sitios gastronómicos registrados en el sistema. Los sitios gastronómicos podrían ser, por ejemplo:

- *Restaurantes*, donde puede haber tipos de comida variados, suelen ser los que poseen más cantidad servicios, como reservas de mesa, pedidos de comida y la posibilidad de comer en el sitio.
- *Pizzería*, que se destacan por brindar un tipo de comida particular como es la pizza y pueden ofrecer tanto el servicio de *Delivery* como el de comer en el mismo sitio.
- *Chefs particulares*, que son chefs certificados que ofrecen distintos menús para los clientes en un domicilio particular.

Cada sitio, además, puede tener un nombre, una descripción (relacionada al sitio) y su posición física.

Las personas registradas en el sistema gastronómico, podrían ser tanto los dueños de los sitios como los clientes de los mismos. Para cada una de estas personas registradas se ofrecerán servicios particulares, por ejemplo, los dueños contarán con servicios relacionados con la administración de su sitio, como pueden ser definir:

- *Menús*, para determinar los productos que ofrecen, como por ejemplo un plato de comida específico, describiendo sus características e indicando los precios. Junto a esto, definen los *Tipos de Comidas* que se ofrecen, por ejemplo, comida *Italiana*, *Vegetariana*, etc.
- *Promociones*, donde detallan sus características, por ejemplo un 2x1 en algún menú particular o un determinado descuento, estableciendo también un periodo de validez para las mismas.
- *Horarios de atención*, para establecer los días y horarios que el sitio se encuentra abierto para los clientes.

Todos los sitios gastronómicos pueden facilitarles a sus clientes, servicios como por ejemplo:

- *Anotarse a una lista de espera on-line,* para ingresar al sitio.
- *Reserva de mesas*, para poder garantizarle al cliente lugares disponibles que le permitan consumir en el sitio. Estas reservas pueden servir posteriormente para brindar otros servicios, por ejemplo, avisarle al usuario que dentro de 1 hora tiene una reserva en un determinado lugar.
- *Delivery*, el cual permite realizar pedidos de comida para que sean enviados a la ubicación del cliente. Cada *Delivery* pasará por diferentes estados, por ejemplo, pendiente, entrega, etc.
- *Visualización y búsqueda de distinta información de interés*, por ejemplo:
    - Los sitios gastronómicos, basándose en la posición actual del cliente.
    - Los sitios favoritos relacionados a cada cliente.
    - Actividad actual de los amigos registrados en el sistema.

También habrá servicios *de notificaciones* que son ofrecidos por el sistema gastronómico, considerando alguna característica de contexto, por ejemplo:

- *Promociones a los clientes*, teniendo en cuenta los sitios cercanos a la posición actual del mismo, considerando también sus sitios favoritos, sitos donde haya realizado consumos y tipos de comidas que son de sus preferencias gastronómicas.
- *Cambios de estado de los consumos de los clientes*, por ejemplo, en caso de que la persona haya utilizado el servicio de *Delivery* de comida y se le informe que está listo para retirar.
- *La actividad de los amigos de la persona*, por ejemplo, que un amigo se encuentra comiendo en algún lugar particular.

Estas notificaciones son activadas acorde al cambio de contexto de la persona, por ejemplo, cambio de posición o la modificación del estado de un *consumo*.

Tanto los clientes como los dueños podrían realizar búsquedas de sitios con filtro, por ejemplo, por tipos de comidas, promociones o por otras características que posea el sitio. También podrían hacer *búsquedas de caminos que permita llegar a un sitio gastronómico*. Para esto se necesita contar con la información de la posición actual de la persona.

Cabe mencionar que los sitios gastronómicos pueden encontrarse cerrados o abiertos para el público, o podría estar justo inaugurando. Estos estados, pueden determinar que los mismos sean tenidos en cuenta, o no, a la hora de brindar un servicio. Por ejemplo, si un usuario busca lugares cercanos a su posición actual, y estos están cerrados podrían no listarse. También los servicios podrían tener diferentes estados, como podría ser, estar o no disponibles en un momento dado, por ejemplo, cada sitio podría desactivar algún servicio, y éste no estar disponible para el usuario.

A partir de la información descripta, se puede apreciar que hay servicios, tanto para clientes, dueños y otros generales, que están disponibles para ambos. Además, algunos de estos son requeridos por las personas, mientras que otros son del estilo notificaciones,

brindados por el sistema gastronómico o sitios. En la información detallada anteriormente se puede apreciar como la posición del usuario y otros de los contextos mencionados (como estados o fechas de validación de las promociones) pueden ser claves a la hora de brindar un servicio.

A partir de los conceptos descriptos, en la siguiente sección se presentará una solución de modelado para abordar dichos conceptos identificados en esta sección.

3.2 Descripción de una primera solución de modelado para Sistemas Gastronómicos

Teniendo en cuenta los conceptos mencionados en la Sección 3.1, a continuación se irán presentando los mismos incrementalmente.

Como se describió anteriormente, en un *Sistema Gastronómico* se registran personas y sitios gastronómicos. Para representar cada sistema propiamente dicho, se define la clase *Sistema_Gastronomico* la cual conoce su nombre, las personas registradas, los sitios gastronómicos registrados y los servicios generales que proveerá. Esto se puede apreciar en la Figura 3.2.1. Las personas, están representadas mediante la clase *Persona*, la cual define: nombre, apellido y dni. Se puede apreciar, en dicha figura, que los sitios gastronómicos son representados con una clase abstracta *Sitio_Gastronomico* (más adelante se describirán subclases concretas para representar diferentes sitios). Todo sitio gastronómico define el nombre, una descripción y un puntaje promedio (que proviene de los puntajes que le fueron asignando las personas que asistieron al mismo). Por último, se puede observar la clase abstracta *Servicio*, y luego, se brindará más detalle de subclases concretas de servicios. Cabe mencionar que *Sistema_Gastronomico* funciona como punto de entrada, respetando el patrón de diseño *Facade* [Gamma et al., 1995].

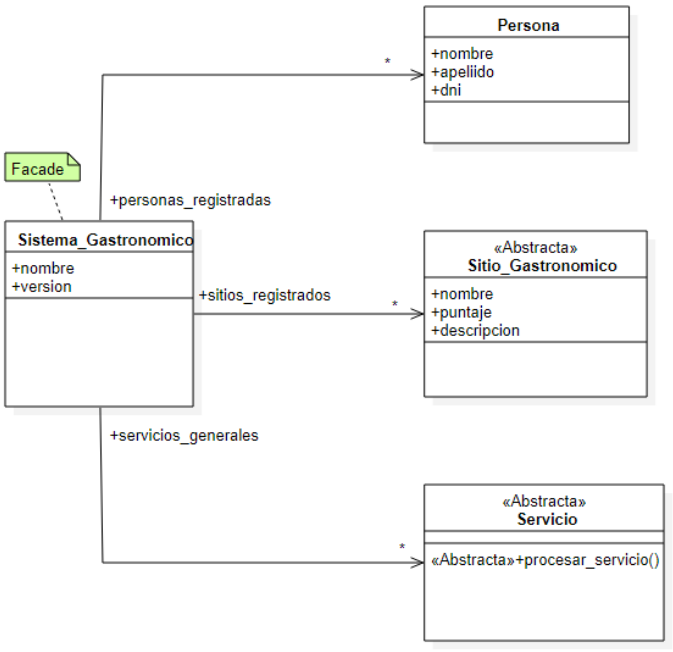


Figura 3.2.1: Clase *Sistema_Gastronomico* y sus relaciones

Como se mencionó en la Sección 3.1, el sistema es capaz de administrar a las personas con diferentes roles: clientes y dueños de los sitios. Por lo tanto, la clase *Persona* conocerá todos sus roles y además el rol actual con el que la persona esta interactuando actualmente con el sistema. Como se puede apreciar en la Figura 3.2.2, la clase abstracta *Rol*, posee dos subclases: *Dueño_Sitio* y *Cliente*. Estos roles como ya se describirá más adelante permite al sistema brindarle servicios específicos para cada rol. Dichos roles se definieron respetando el patrón de diseño *Rol Object* [Bäumer et al., 1998]. Además, la clase *Persona,* cuenta con su lista de amigos, los cuales pueden ser tanto clientes como dueños de sitios gastronómicos. También se puede observar en la Figura 3.2.2, que la clase *Persona* conoce su posición (*Posicion*) y orientación (*Orientacion*) actual. La clase *Orientacion* define los valores (x,y,z), mientras que la clase *Posicion* está diseñada usando el patrón *Composite* [Gamma et al., 1995], a fin de poder tener diferentes formas de ser representada[8]. Acorde a esto se definieron las siguientes subclases: *Coordenada, Etiqueta* y *Coordenada_Etiqueta*.

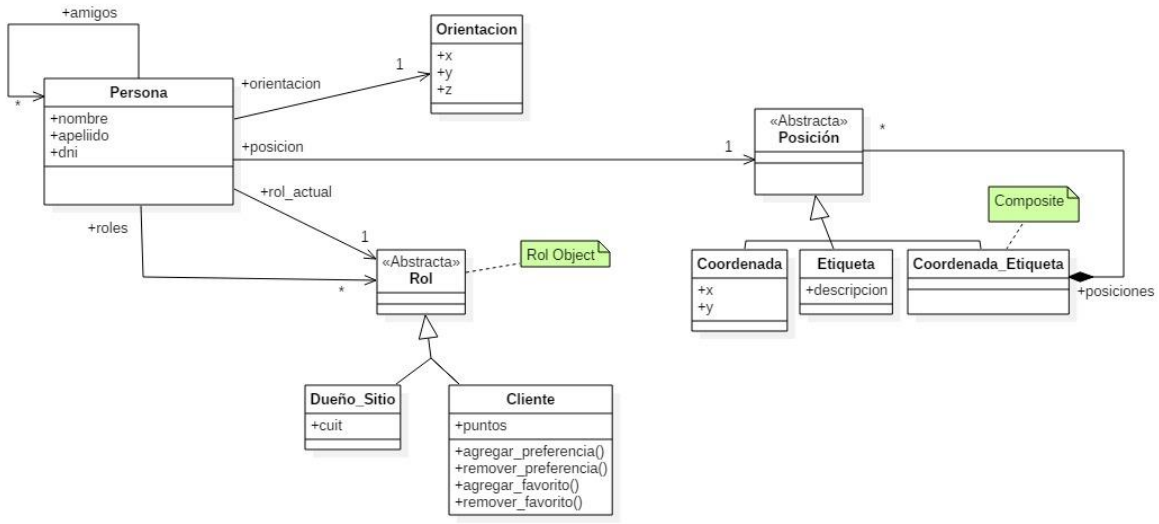


Figura 3.2.2: Clase *Persona* y sus relaciones

Como se pudo observar en la Figura 3.2.2, los dueños del sitio gastronómico poseen un número de cuit que los identifica (además de su dni), mientras que los clientes conocen los puntos que tienen acumulados, estos se van incrementando acorde a los distintos consumos que éstos vayan realizando.

Además, cada *Dueño* tiene una colección de sitios gastronómicos, los cuales puede administrar dentro del sistema, como se puede observar en la Figura 3.2.3. Por otro lado, en dicha figura también el *Cliente* posee sus sitios gastronómicos elegidos como favoritos y sus preferencias gastronómicas, como por ejemplo la comida oriental o francesa.

Ambas colecciones son gestionadas por el mismo *Cliente* mediante los métodos `agregar_favorito()`, `remover_favorito()`, `agregar_preferencia()` *y* `remover_preferencia()`.

[8] Como se menciona en [Leonhardt, 1998] la posición puede ser representada de varias maneras, ya sea, por ejemplo, simbólicamente o geométricamente.

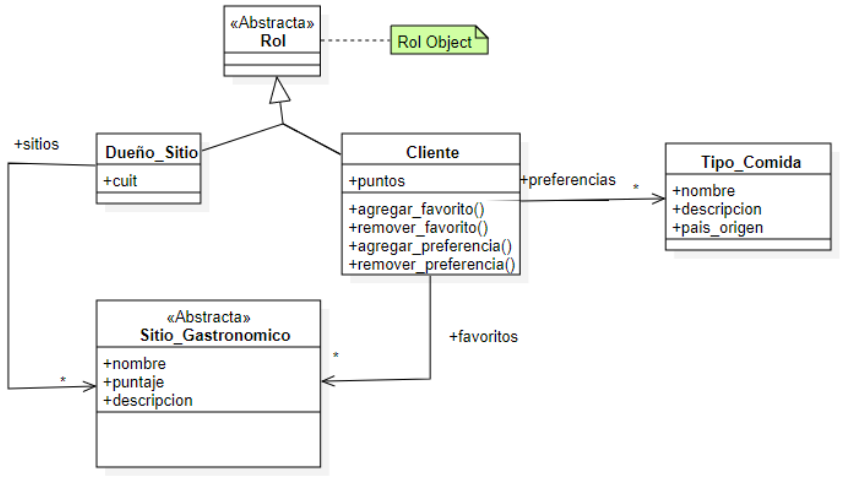


Figura 3.2.3: Clase *Rol* y sus relaciones

Como se mencionó anteriormente, los sitios gastronómicos son representados mediante una jerarquía, la cual consta de la superclase abstracta *Sitio_Gastronomico* y de las subclases concretas que distinguen a los distintos sitios, como pueden ser un *Restaurante*, *Cerveceria*, *Casa_Comida*, *Chef_Particular*, entre otros. Esto se puede apreciar en la Figura 3.2.4.

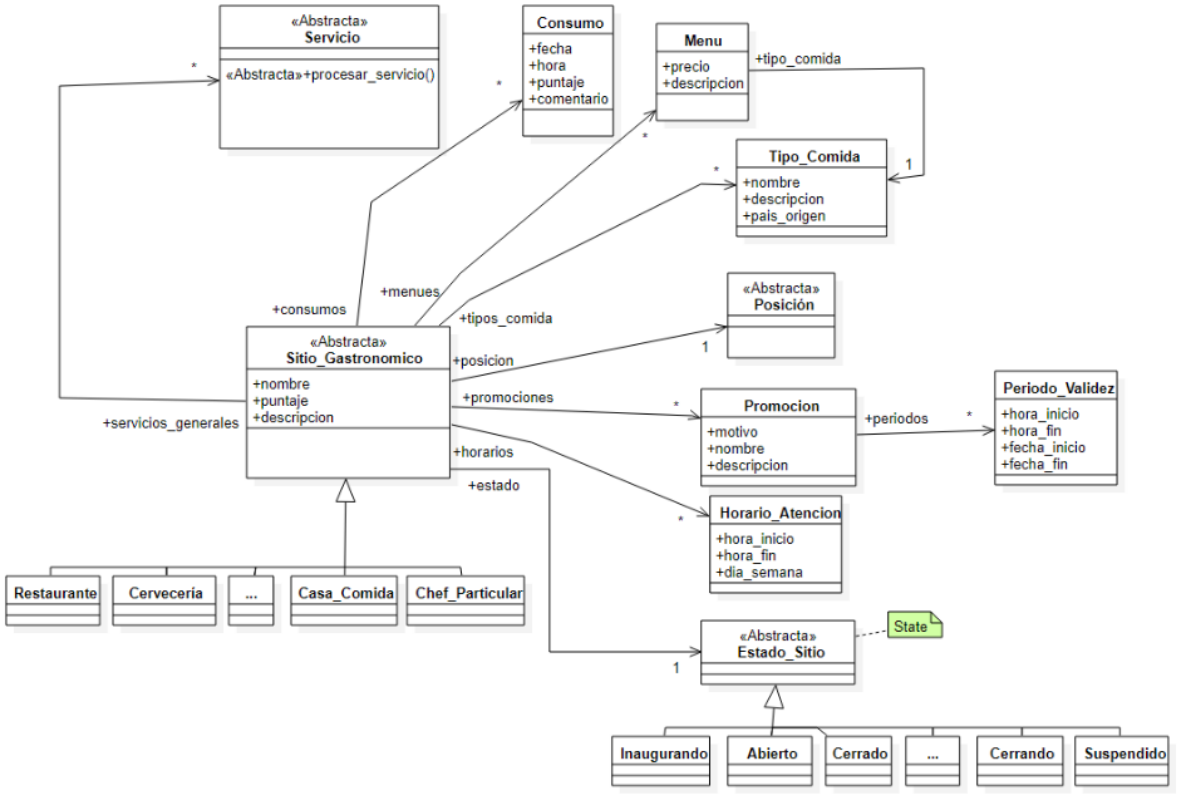


Figura 3.2.4[9]: Clase *Sitio_Gastronomico* y sus relaciones

[9] Se observa en la Figura 3.2.4, que tanto en las jerarquías de las clases *Sitio_Gastronomico* y *Estado_Sitio* existen subclases denominadas "…"; esto se especificó a fines de que el lector puede apreciar que estas jerarquías podrían crecer y contar con nuevas clases en un futuro. Estos son puntos de extensión del modelo.

Como se puede observar en la Figura 3.2.4, cada sitio conoce sus:

- *servicios*, es decir, aquellos servicios que serán provistos específicamente por ese sitio gastronómico. Estos servicios serán clases concretas de la clase abstracta *Servicio*.
- *consumos*, estos representan un registro de los consumos que fueron realizados en dicho sitio. Cada consumo está representado con la clase *Consumo*, la cual cuenta con la fecha y hora de cuando se llevó a cabo, un comentario y un puntaje por parte de la persona que lo consumió.
- *menúes*, que el sitio ofrece o tiene disponible para los clientes. Cada uno de estos se representa con la clase *Menu,* la cual tiene el precio, una descripción detallada y el tipo de comida para clasificarlo.
- *tipos de comida*, esto permite que los clientes puedan, por ejemplo, conocer qué tipos de comida se ofrecen en cada sitio gastronómico. Estos tipos de comidas se representan con la clase *Tipo_Comida*, la cual posee un nombre que lo identifica, una descripción opcional y el país de origen de donde proviene dicha comida.
- *posición*, esta permite indicar donde está ubicado cada sitio gastronómico. Cabe destacar que esta relación va a la clase *Posicion* previamente definida en la Figura 3.2.2.
- *promociones*, donde cada *Promocion* cuenta con un nombre, una descripción opcional, un motivo para detallar de que se trata y los periodos de validez (diagramado con la clase *Periodo_Validez*) que establecen el tiempo en el que la promoción estará vigente para los clientes. Cada *Periodo_Validez* se establece por medio de una fecha y hora de inicio para indicar a partir de cuándo se puede acceder a la promoción, y, una fecha y hora fin para indicar cuando termina.
- *horarios*, los cuales representan los horarios de atención al público del sitio. Cada *Horario_Atencion*, es definido por una hora de inicio, una hora de cierre y el día de la semana al cual corresponde ese rango horario.
- *estado,* esto representa el estado actual del sitio gastronómico, por ejemplo: un sitio se puede encontrar abierto o cerrado. Para esto se definió una jerarquía *Estado_Sitio* con subclases concretas que representan los diferentes estados. Esto respeta el patrón de diseño *State* [Gamma et al., 1995], lo que permitirá luego, que cada sitio gastronómico modifique su comportamiento cada vez que cambie su estado interno. Por ejemplo, no son los mismos servicios los que se ofrecen cuando el sitio está cerrado que abierto.

A continuación se verá con más detalle la clase *Consumo,* además de lo que se describió anteriormente. Esta clase conocerá el *Cliente* que realizo dicho consumo*,* un tipo de consumo representado con la clase *Tipo_Consumo;* esta clase se representa mediante patrón de diseño *Type Object* [Woolf and Johnson, 1996]. Este patrón permite desacoplar el tipo del objeto en sí, en este caso, se desacopla el tipo de consumo que se efectúa; lo cual permite tratar a todos los consumos de manera polimórfica. Los tipos de consumos podrían ser, por ejemplo, un pedido de delivery, una reserva de mesa de algún restaurante particular, etc. Además, la clase *Consumo* va a conocer su estado actual, el cual se representa con la jerarquía *Estado_Consumo* (que respeta el patrón de diseño *State*

[Gamma et al., 1995]). Los estados concretos que se pueden dar son, por ejemplo: *Cancelado*, *Pendiente*, *Confirmado*, etc. Las relaciones de conocimiento descriptas se pueden observar en la Figura 3.2.5.

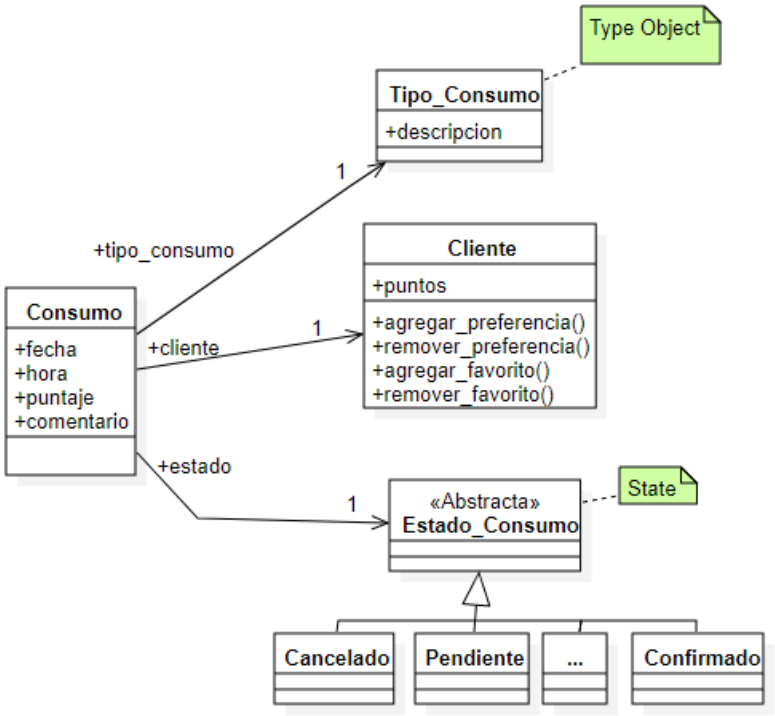


Figura 3.2.5[10]: Clase *Consumo* y sus relaciones

A continuación se brindarán más detalles de la jerarquía *Servicio*. Como se mencionó en la Sección 3.1, hay servicios que son generales a los sistemas gastronómicos, mientras que hay otros destinados a los dueños de los sitios gastronómicos o a los clientes de los mismos. Por lo tanto, se decidió desacoplar el tipo de servicio (clase *Tipo_Servicio*) usando el patrón de diseño *Type Object* [Woolf and Johnson, 1996]. El tipo de servicio podrá ser, por ejemplo, *general*, *para_dueño* o *para_cliente*. Estos son los valores que toma la descripción de la clase *Tipo_Servicio*, lo que permite interactuar con los servicios de manera uniforme, y tener su tipo desacoplado. Por otro lado, los servicios tiene asociado un estado, el cual sirve para determinar cómo se encuentra dicho servicio en un momento dado, como por ejemplo: si el servicio de *delivery* está activo o no. Este estado se representa mediante la utilización del patrón *State* [Gamma et al., 1995], el cual se implementa con una jerarquía que tiene como superclase a *Estado_Servicio* y a sus subclases que representan a los diferentes estados. Lo antes descripto se puede observar en la Figura 3.2.6.

---

[10] Se observa en la Figura 3.2.5, que en la jerarquía *Estado_Consumo* existe una subclase denominada "…"; esto se especificó a fines de que el lector puede apreciar que esta jerarquía podría crecer y contar con nuevas clases en un futuro. Esto es un punto de extensión del modelo.

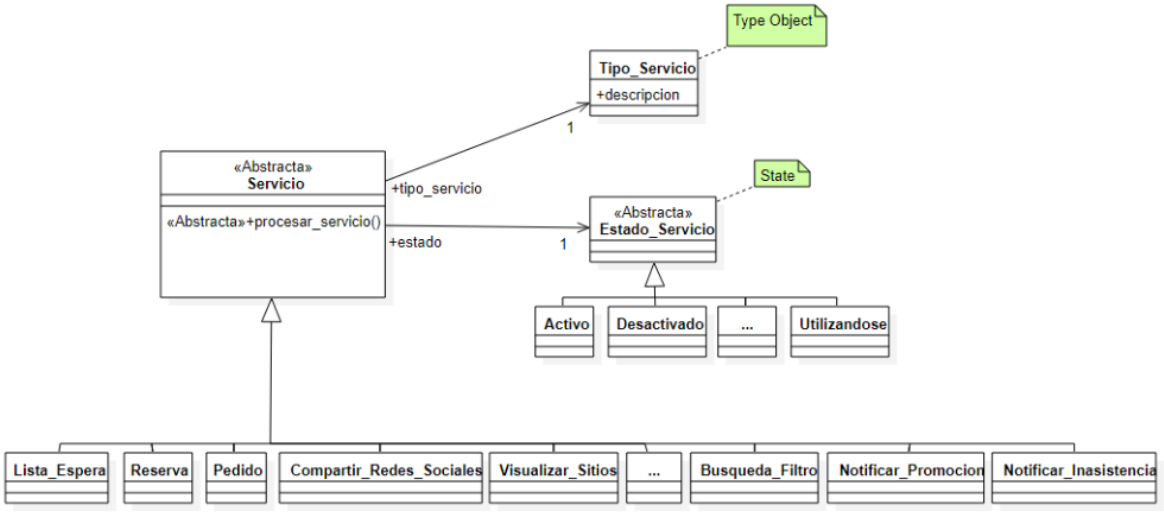


Figura 3.2.6[11]: Clase *Servicio* y sus relaciones

Se puede apreciar en la Figura 3.2.6 algunas subclases de servicios, por ejemplo, aquellos relacionados con generar una *reserva*, *visualizar sitios*, *notificaciones*, etc. Cada servicio será procesado mediante el método polimórfico `procesar_servicio()`, el cual deberá ser redefinido por cada subclase. Por una cuestión de espacio en el diagrama, la implementación de este método por parte de las subclases no se ve reflejado.

En la Figura 3.2.7, se puede observar el modelo generado hasta el momento.

[11] Se observa en la Figura 3.2.6, que la jerarquía *Servicio* existe una subclase denominada "…"; esto se especificó a fines de que el lector puede apreciar que esta jerarquía podría crecer y contar con nuevas clases en un futuro. Esto es un punto de extensión del modelo.

Figura 3.2.7: Modelo presentado hasta al momento

Como se mencionó a lo largo del documento, brindar servicios cuando cambia alguna característica de contexto es de interés para este dominio, Es decir, es necesario estar observando aquella información que varía, para lograr así, poder activar servicios. Para esto, se incorporaron en la Figura 3.2.8 relaciones reflejando el patrón de diseño *Observer* [Gamma et al., 1995]. Dicho patrón define una dependencia del tipo uno a muchos entre objetos, de manera que cuando uno de los objetos cambia su estado, notifica de este cambio a todos los dependientes. En el modelo expuesto en la Figura 3.2.8, se puede apreciar las relaciones de "*observer*" con líneas puntadas.

Las relaciones de "*observer*" que se agregaron en la Figura 3.2.8 son las siguientes:

- la clase *Sistema_Gastronomico* observa a las clases:
  - *Persona*, ante un cambio en una característica de la persona, por ejemplo, se activan servicios nuevos o notificaciones.
  - *Sitio_Gastronomico*, ante un cambio en un sitio, esto dispara la notificación o actualización de información.
  - *Servicio*, cualquier cambio de servicio es propagado.
- la clase *Persona* observa a su *posición*, *orientación*, *rol y amigos*, cualquier cambio se da aviso a quienes están observando, en este caso, al *Sistema_Gastronomico.*
- la clase *Sitio_Gastronomico* observa a su *consumo*, *promociones* y *estados*. Ante cualquier cambio se da aviso al *Sistema_Gastronomico.*
- la clase *Servicio* observa a su *estado*, al producirse un cambio de *estado* se da aviso al *Sistema_Gastronomico.*
- la clase *Promoción* observa a su *Periodo_Validez*, si este periodo cambia se da aviso al *Sitio_Gastronomico.*
- la clase *Consumo* observa a su *estado*, al producirse un cambio de *estado* se da aviso al *Sistema_Gastronomico.*

En la Figura 3.2.8, se puede observar la primera solución de modelado propuesto para *Sistemas Gastronómicos*. En esta solución se cuenta con información del dominio y del contexto de manera acoplada, siendo ésta la forma que habitualmente se modela. Como aquí es de interés las características del contexto, para poder brindar servicios contextuales, desacoplar aquellos aspectos del contexto lograrán más flexibilidad a nivel de modelado y de esta manera, se permitirá la evolución del mismo.

En caso de que surja un nuevo contexto relevante del usuario, por ejemplo, si tiene alguna enfermedad que le impida comer cierto tipo de comida, esto implicaría agregar relaciones nuevas y seguir acoplando contexto a la *Persona*. Las modificaciones de nuevos contextos en *runtime* serían muy complejas.

Acorde a esto, en la siguiente sección se presentará un modelo usado de base para desacoplar el contexto [Fortier et al., 2010] rediseñando el modelo propuesto en la Figura 3.2.8. Se propondrá un nuevo modelo *Sistemas Gastronómicos* desacoplando los conceptos de contexto.

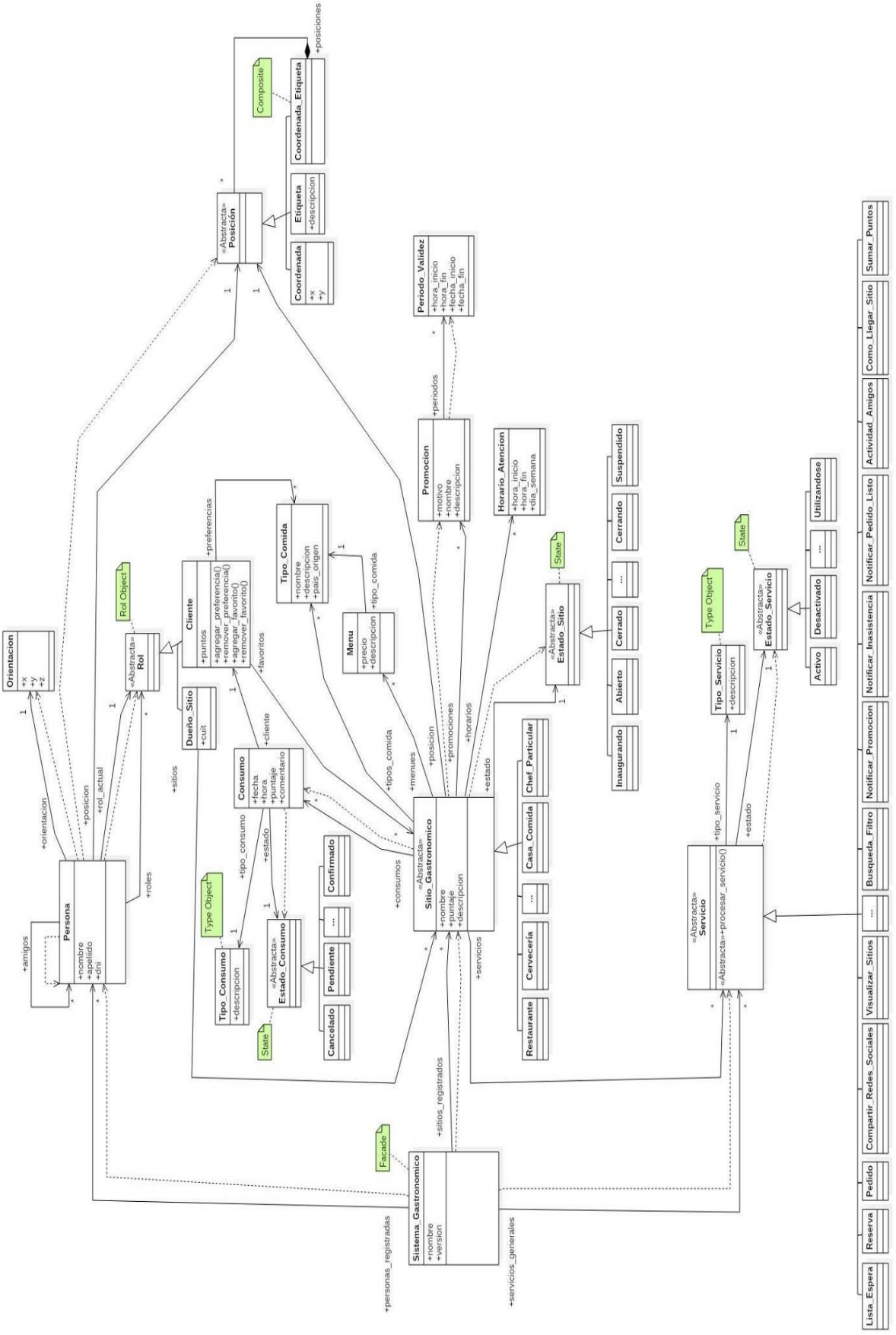


Figura 3.2.8: Primera solución de modelado propuesta para Sistemas Gastronómicos

3.3 Modelo de Contexto usado como base

En base a lo concluido en la sección anterior, se rediseñará el modelo resultante de la Figura 3.2.8 (presentado en la Sección 3.2), separando tanto el contexto como así también el manejo de los cambios de contexto, logrando de esta manera, un diseño más flexible. Para esto, se usará de base un modelo de aplicaciones sensibles al contexto existente, el cual se describirá en esta sección. Luego, en la Sección 3.4 se describirá el rediseño en sí de la Figura 3.2.8.

Para llevar a cabo dicho rediseño del modelo de la Figura 3.2.8, se utiliza como base una extensión del *Framework Context-Aware* [Fortier et al., 2007] presentado en la Sección 2.4.2. Recordemos las clases principales que definía el framework para luego dar paso a la extensión que se usará:

- *AdaptationEnvironment*: esta clase representa cada uno de los entornos de adaptación. Cada *AdaptationEnvironment* hace referencia a un conjunto de *AwareObject* para "escuchar" sus cambios de contexto. Cuando se activa un cambio de contexto, se envía una notificación al entorno para que pueda realizar una acción específica, como puede ser informar al cliente que el pedido realizado a un restaurante está en estado de “listo”. En la solución de modelado propuesta en la Figura 3.2.8 (de la Sección 3.2), un *AdaptationEnvironment* podría ser la clase *Sitio_Gastronomico* donde puede estar escuchando a un *AwareObject* como la clase *Persona*.
- *AwareObject:* son objetos que poseen caracteristicas de contexto, como se menciono anteriormente, el *AwareObject* más evidente en el modelo presentado en la Figura 3.2.8 (de la Sección 3.2), podría ser una *Persona.*
- *ContextFeature*: representa un valor actual de un contexto e informa a cualquier parte interesada si hubo algún cambio. Generalmente, la parte interesada, es un *AwareObject*. En el modelo presentado en la Figura 3.2.8 (de la Sección 3.2), podría ser una *Posicion* de la clase *Persona.*
- *EventHandler*: estos están a cargo de realizar la reacción a un cambio de contexto. Desacoplar el entorno (*AdaptationEnviroment*) de las acciones para realizar cambios de contexto. Un posible *EventHandler* en relacion a la Figura 3.2.8 (de la Sección 3.2) podria ser un handler que maneja que sucede cuando la persona cambia la posición.
- *MatchingPolicy*: permite decidir si un evento es de interés o no. Por ejemplo, esto podria aplicarse para decidir si la posición de la persona cambió solo un metro, no propagar ningun aviso.

En [Challiol et al., 2007] los autores hacen una reformulación de las relaciones entre las clases presentadas en la Figura 2.4.3 (de la Sección 2.4), obteniendo como resultado lo planteado en la Figura 3.3.1. Se puede apreciar que el *AdaptationEnvironment* conoce y observa a los *AwareObject.* Esto cambia respecto de la Figura 2.4.3 donde el *AwareObject* conocía a los *AdaptationEnvironment.*

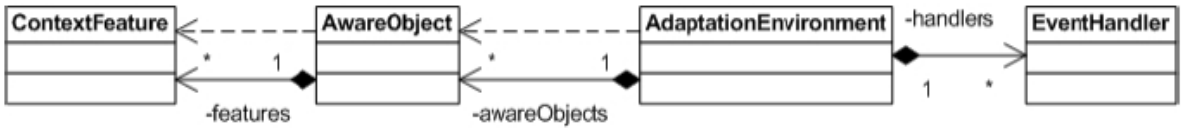


Figura 3.3.1: Modelo de contexto simplificado [Challiol et al., 2007]

Es importante ampliar una extensión particular del *AwareObject* que no fue mencionada en la Sección 2.4. En [Fortier et al., 2010] se describe la clase *AwareModel* como una sublcase particular del *AwareObject*, como se puede apreciar en la Figura 3.3.2. Esta subclase representa un *AwareObject* que materializa conceptos existentes en el modelo de aplicación, y crean una representación para ellos en el modelo de contexto. Para ello, un *AwareModel* tiene una referencia al objeto de aplicación y actúa como un *Proxy dinámico* [Gamma et al., 1995], reenviando cada mensaje que recibe al objeto base. Además, dado que un *AwareModel* es una subclase de *AwareObject*, proporciona soporte para administrar el contexto.

La clase *AwareModel* también agrega comportamiento más complejo [Fortier et al., 2010]. Se puede pedir a un *AwareModel* que anule el mensaje de un objeto base con una *ContextFeature*. Como resultado, en lugar de reenviar el mensaje al objeto original, el *AwareModel* devolverá la situación actual de la *ContextFeature*.

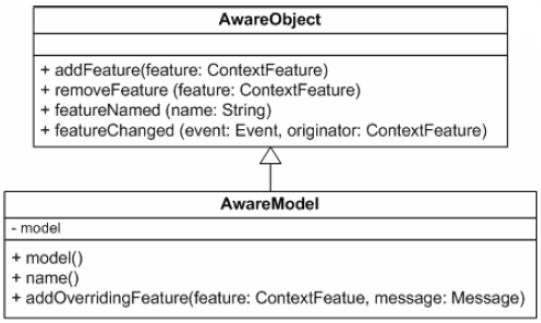


Figura 3.3.2: Modelado del *AwareModel* [Challiol et al., 2012]

En la Figura 3.3.3 se presenta un modelo de instancias donde se puede observar como la instancia *usuario* (*AwareModel*) tiene como *model* a una instancia de la clase *Persona*, la cual pertenece al modelo de aplicación. Dicho *AwareModel* también observa a sus *features* como *Posicion* y *Orientacion*.

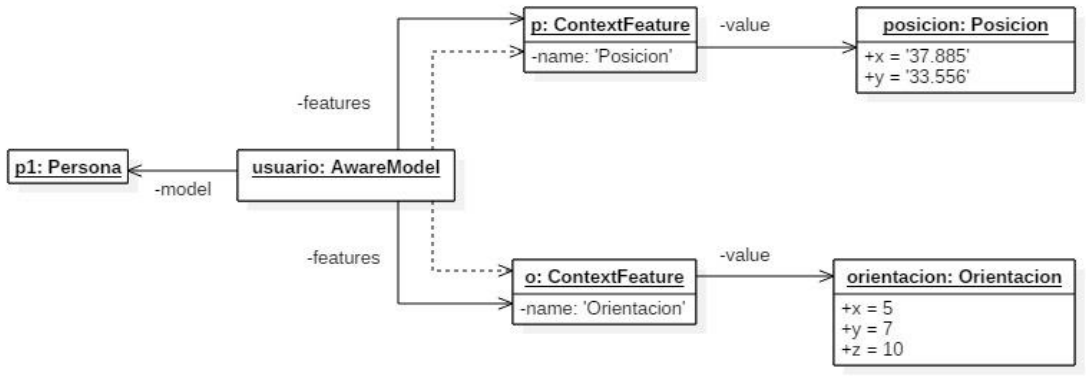


Figura 3.3.3: Ejemplo de modelado con *AwareModel*

Recordemos que lo que busca el *Framework Context-Aware* descripto en [Fortier et al., 2007] y [Challiol et al., 2007] es separar el dominio de aplicación del dominio de adaptación, y así, poder utilizar infraestructuras específicas que puedan ser reutilizadas en diferentes dominios de aplicación.

Los autores del trabajo [Fortier et al., 2007], siguen explorando características de aplicaciones sensibles al contexto orientadas a servicios, y en [Fortier et al., 2010], se presenta una extensión concreta para soportar servicios. Las clases de dicha extensión se pueden apreciar en la Figura 3.3.4. Como se puede observar las clases *AdaptationEnviroment* y *EventHandler* están en color gris para indicar que pertenecen al modelo de [Fortier et al., 2007] y se utilizan para destacar cómo es extendido este framework para soportar servicios sensible al contexto.

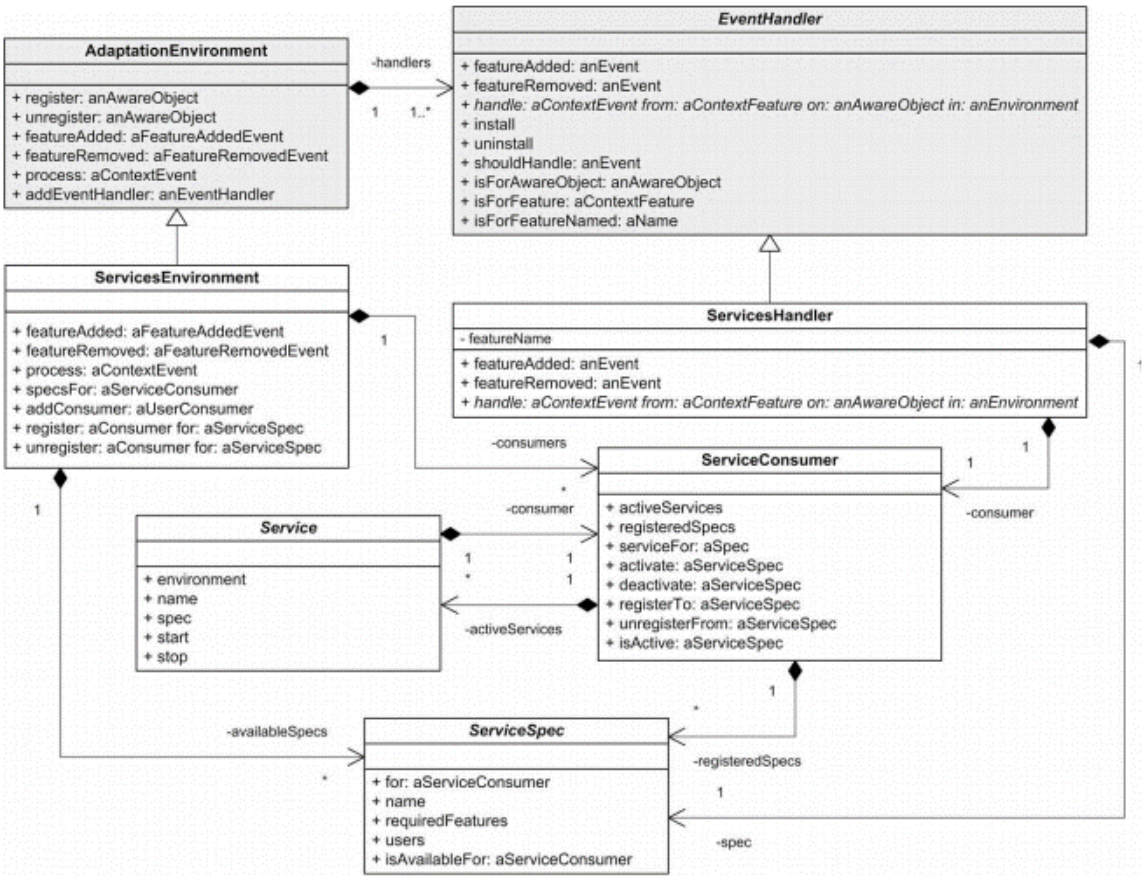


Figura 3.3.4: Framework para servicios sensibles al contexto [Fortier et al., 2010]

A continuación se describen las principales responsabilidades de cada una de las clases de la Figura 3.3.4 [Fortier et al., 2010]:

- *ServiceEnvironment*: es un *AdaptationEnvironment* especializado para manejar los servicios y los usuarios (es decir, a los *ServiceConsumer*). Añade gestión de usuarios y de especificaciones de servicios, y junto con un *ServiceHandler*, coordina en qué momento un servicio está activo para un usuario determinado.
- *ServiceHandler*: este es un manejador especializado, que está relacionado con el *ServiceEnvironment* para cada usuario y para cada uno de los servicios a los que se

ha registrado. Dado que un servicio especifica qué *ContextFeauture* requiere para funcionar, el *ServiceHandler* "escucha" los cambios de esas *ContextFeauture* específicas. Cuando una notificación de un cambio de contexto llega al *ServiceHandler*, éste comprueba si el servicio debe estar activo o no para ese usuario.

- *Service*: la clase abstracta *Service* no proporciona mucho comportamiento por sí sola, sino que define un conjunto de métodos de acceso para la información, y otros que serán utilizados como *callbacks* por el entorno. Algunos ejemplos de estos métodos, son `start` y `stop`, que son llamados cuando se activa y desactiva un servicio.
- *ServiceConsumer*: es un *wrapper* del *AwareObject* que agrega información del servicio. Se mantiene un registro de los servicios registrados, y de los que están actualmente disponibles. Dado que este framework está centrado en servicios para el usuario, se hace referencia a estos consumidores, como usuarios, pero debe notarse que cualquier *AwareObject* puede actuar como un *ServiceConsumer*.
- *ServiceSpec*: los usuarios interesados en un servicio en particular deben registrarse a su especificación. Cada especificación brinda información sobre el servicio (nombre, descripción, etc.), e indica también cuándo el servicio puede estar disponible para un usuario determinado. Por ejemplo, en un sistema basado en posicionamiento, si el usuario no tiene una *ContextFeature* de posición, entonces no se puede determinar su posición y, por lo tanto, no se puede decidir si debe proporcionar el servicio o no. Se aclara que por defecto, el servicio comprueba una colección de *ContextFeature* requeridas, las cuales pueden ser extendidas como subclases de *ServiceSpec*.

Dado que en las Figuras 3.3.1, 3.3.2 y 3.3.4 (modelos definidos en [Challiol et al., 2007] [Challiol et al., 2012] y [Fortier et al., 2010] respectivamente) quedan muchos conceptos separados, en la Figura 3.3.5 se presenta la integración de estos para facilitar así la comprensión del modelo propuesto en la siguiente sección.

Los modelos presentados en esta sección, fueron diseñados a lo largo de varios papers por los mismos autores, sin embargo, no cuentan con las implementaciones de las clases mostradas ni tampoco cómo interactúan para que los eventos sean propagados, logrando ejecutar la acción determinada por los *EventHandler*.

Cabe aclarar que el modelo presentado en [Fortier et al., 2007] fue usado en [Musi Gentile, 2015] con el objetivo de contar con un modelo general de gestos desacoplado del mecanismo de sensado. Sin embargo, dicho modelo no esta orientación a servicios, acorde a esto, se decidió no considerarlo en esta tesina, ya que tenía una orientación distinta a la que se busca resolver en relación a los servicios.

En la próxima sección, el modelo de la Figura 3.2.8 (presentado en la Sección 3.2) se rediseñará acorde al modelo de la Figura 3.3.5. Donde no solo se abordarán las extensiones necesarias para soportar servicios del domino gastronómico sino también se evaluarán que clases o relaciones que deberán rediseñarse.

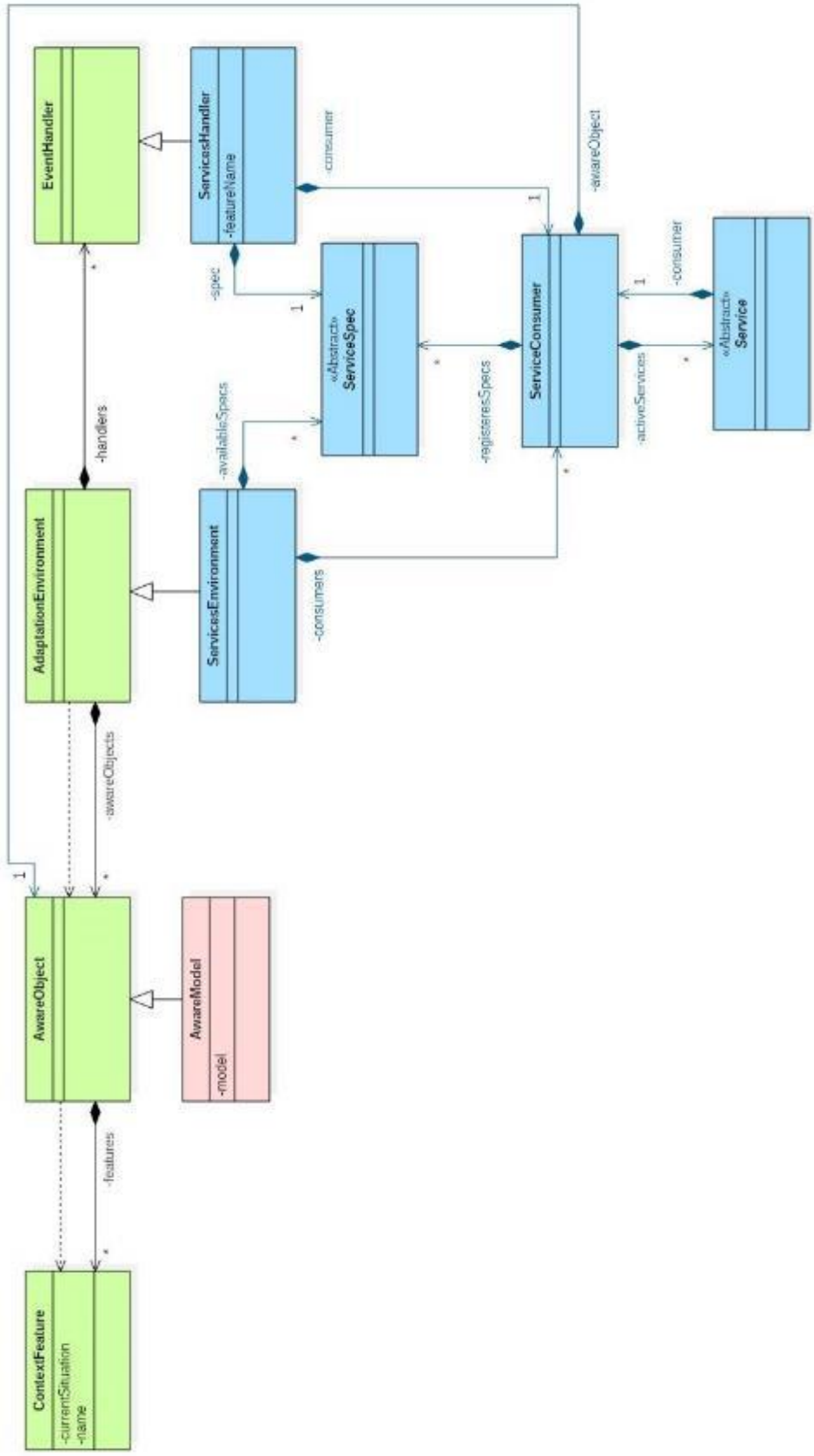


Figura 3.3.5: Integración de los modelos presentados en las Figuras 3.3.1, 3.3.2 y 3.3.4

3.4 Solución de modelado propuesto considerando conceptos de contexto

En esta sección se describirá paso a paso el modelo propuesto, contemplando los conceptos mencionados en la sección anterior a fin de proveer una solución de modelado brindando servicios gastronómicos contextuales. Esta solución se basa en la refactorización del modelo de la Figura 3.2.8 desacoplando los conceptos de contexto acorde a lo presentado en la Sección 3.3.

A continuación se describe la solución de modelado propuesta, la cual cuenta de tres partes bien diferenciadas: el modelo de contexto, las clases involucradas en los valores de contexto y el modelo de dominio.

- **Modelo de Contexto Propuesto considerando servicios contextuales gastronómicos**

  Se tomó de base el modelo de la Figura 3.3.5, el cual provee:

  - Representación para aumentar objetos del dominio considerando características contextuales (*ContextFeature*), como son los *AwareModel*.
  - Representación de los ambientes que reaccionan ante los cambios de contexto, en particular aquellos ambientes que brindan servicios (*ServicesEnvironment*) acorde a los cambios contextuales.
  - Representación de servicios contextuales, involucrando diferentes clases, por ejemplo, *Service*.

  Se analizan que clases específicas surgen para brindar soporte a servicios contextuales gastronómicos. Se detectó que, tanto el *AwareModel* como los *ContextFeature* no necesitaban ser extendidos, ya que con estos se puede representar la información necesaria para estos servicios, debido a que dichas clases son genéricas a cualquier tipo de aplicación.

  En cuanto a los ambientes, se detectó que para los servicios contextuales gastronómicos, se tenían dos niveles de ambientes:

  - *Ambiente Servicios Gastronómico*, como un ambiente general que contiene todos los sitios gastronómicos, y también todo el espacio físico relacionado a la ciudad.
  - *Ambiente Servicios Sitio*, representa el ambiente de cada sitio gastronómico. Cada uno de estos es conocido por el ambiente más general.

  Estos conceptos se representaron como clases específicas de *ServicesEnvironment*, como se puede apreciar en la Figura 3.4.1. Además, se detectó la necesidad de representar en este modelo los servicios contextuales gastronómicos, ya mencionados en las Secciones 3.1 y 3.2. Para esto, se extendió la clase *Service* como se puede observar en la Figura 3.4.1. En dicha figura aparecen en color gris las clases del modelo usado de base (presentado en la Figura 3.3.5).

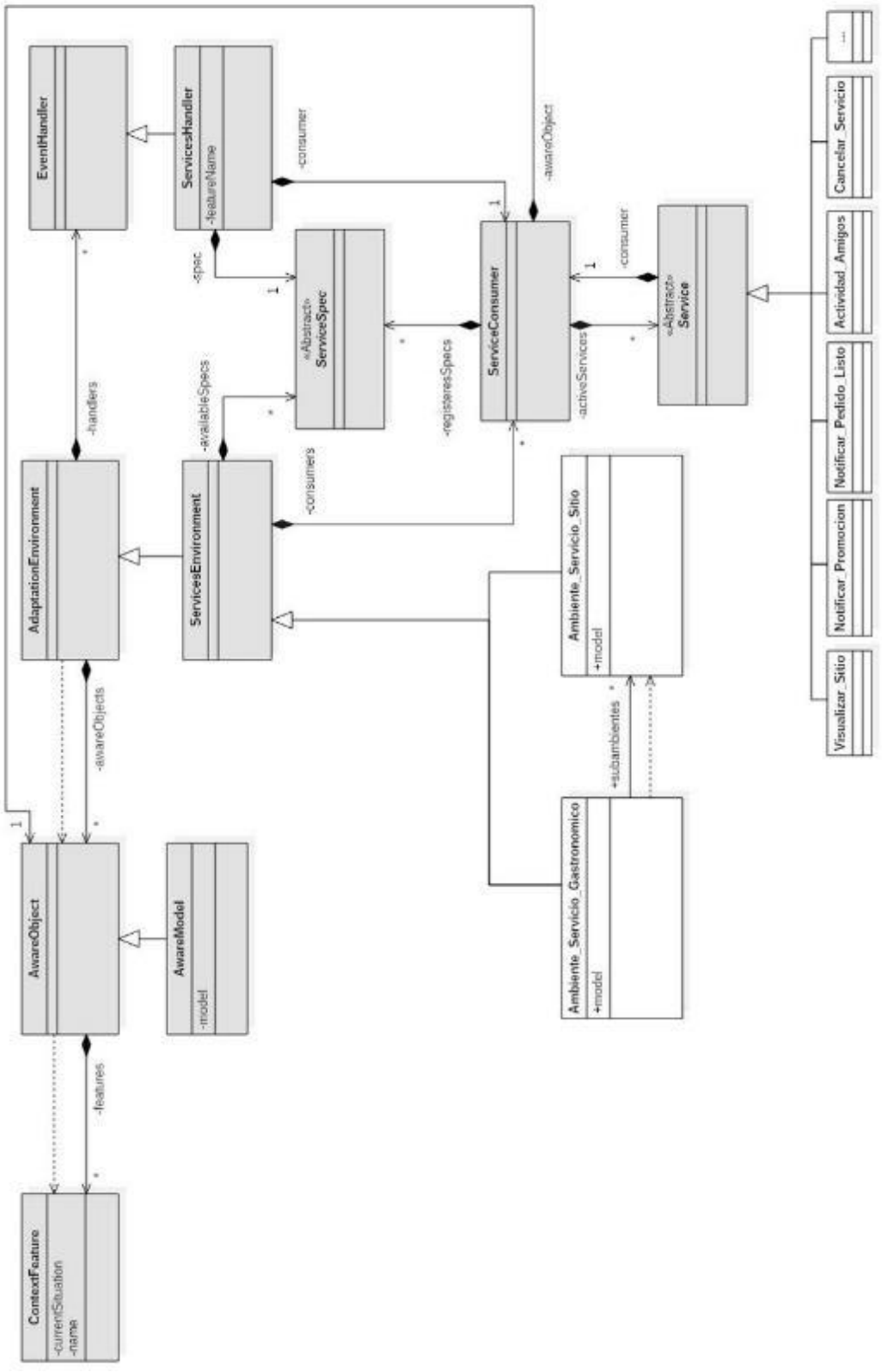


Figura 3.4.1[12]: Modelo de Contexto Propuesto para el dominio Gastronómico

[12] Se observa en la Figura 3.4.1, que la jerarquía *Service* existe una subclase denominada "…"; esto se especificó a fines de que el lector puede apreciar que pueden haber otros servicios, que por claridad, solo se especificaron algunos de ellos.

Se puede observar en la Figura 3.4.1 que la clase *Ambiente_Servicio_Gastronomico* conoce a sus "*subambientes*" (*Ambiente_Servicio_Sitio*). Esto permite tener, como se mencionó anteriormente, un ambiente más general que engloba todos los sitios gastronómicos. En este caso, se usa el patrón de diseño *Observer* [Gamma et al., 1995] para poder desencadenar así servicios generales, que son activados a través de servicios de algún sitio gastronómico. Esta relación de *observer* es la que se tenía en la Figura 3.2.8 (de la Sección 3.2) entre el *Sistema_Gastronomico* y *Sitio_Gastronomico*, que ahora con esta nueva solución de modelado pasa a estar en el modelo de contexto propuesto en la Figura 3.4.1.

También se puede apreciar en la Figura 3.4.1, que las clases *Ambiente_Servicio_Gastronomico* y *Ambiente_Servicio_Sitio*, cuentan con una variable denominada *model*, la cual permitirá conocer la clase del dominio aumentada con servicios contextuales. Esto último, funciona de la misma manera que la clase *AwareModel.* En particular, la clase *Ambiente_Servicio_Gastronomico* tendrá como *model* a un *Sistema_Gastronomico*, y la clase *Ambiente_Servicio_Sitio* a un *Sitio_Gastronomico*. De esta manera, queda desacoplada la información propia del dominio y en otro nivel, los servicios contextuales manejados por los distintos ambientes y subambientes.

Cabe mencionar que los servicios detallados en la Figura 3.2.8 (de la Sección 3.2) ahora pasan a ser subclases de la clase *Service* en la Figura 3.4.1. En este caso, cambian los métodos que se deben implementar, pero conceptualmente son los mismos servicios los que se proveen. Tampoco serán necesarias las clases *Estado_Servicio* y *Tipo_Servicio* de la Figura 3.2.8 (de la Sección 3.2), ya que esta información es representada con la clase *ServiceSpec* (ver Figura 3.4.1), la cual permite definir la especificación de cada servicio.

- **Clases involucradas en los valores de contexto**

  En la Figura 3.4.1 se presentó el modelo de contexto propuesto para brindar servicios gastronómicos contextuales. Al tener un modelo tan general, hay relaciones que surgen a nivel de instancias, como es el caso de los valores que toman los contextos.

  A continuación se presentan cada uno de los objetos del modelo que tendrá contexto y las clases involucradas en los valores de contexto. De esta manera, se irá detallando que refactorizaciones surgen de la Figura 3.2.8 (de la Sección 3.2), dado que algunas relaciones ahora se van a dar de manera dinámica a nivel de instancias y no quedarán explicitas en a nivel de modelo de clases.

  A continuación se describen las instancias de *AwareModel* que se deberán considerar:

  - *AwareModel* que representa una *Persona* con contexto, para cada persona se deberá instanciar un *AwareModel* con las siguientes *ContextFeatures*:
    - *posicion*, deberá tomar un valor de la clase *Posicion* que se había definido en la Figura 3.2.8 (de la Sección 3.2).
    - *orientacion*, deberá tomar un valor de la clase *Orientacion* que se había definido en la Figura 3.2.8 (de la Sección 3.2).

- *rol*, deberá tomar un valor de la clase *Rol* que se había definido en la Figura 3.2.8 (de la Sección 3.2).
- *amigos*, definido en la Figura 3.2.8 (de la Sección 3.2) que deberá tomar un valor de la clase *AwareModel*, el cual posee como *model* a una instancia de la clase *Persona* y sus *ContextFeatures*.

Acorde a esto, en la Figura 3.4.2 se puede apreciar las clases relacionadas con los valores de contexto del *AwareModel* que representa una *Persona.* Tener en cuenta que por simplicidad no se agrega la clase *AwareModel* que servirá para indicar los amigos de una persona.

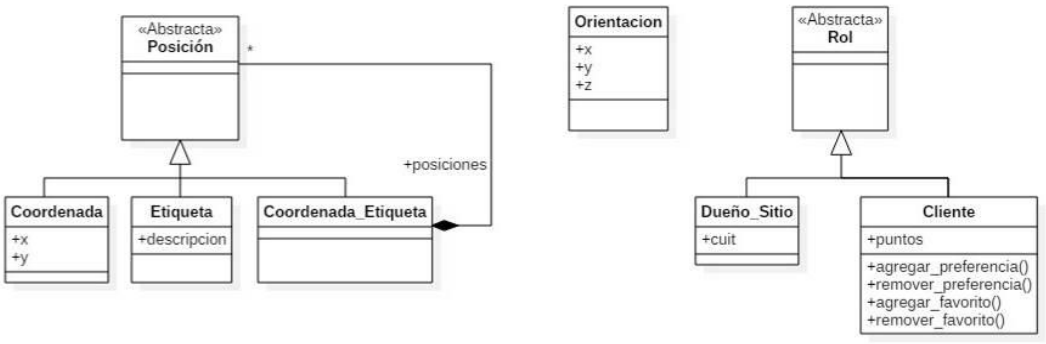


Figura 3.4.2: Clases relacionadas con los valores de contexto del *AwareModel* que representa una *Persona*

Para una mayor comprensión de cómo son usados los valores de contexto a nivel de instancia, en la Figura 3.4.3 se puede apreciar un diagrama de instancias de un *AwareModel* de una persona. En dicha figura, se pueden observar las instancias relacionadas al contexto (*AwareModel* y *ContextFeature*), las instancias relacionadas a los valores de contexto, y el valor de *model* que en este caso es una instancia de la *Persona*.

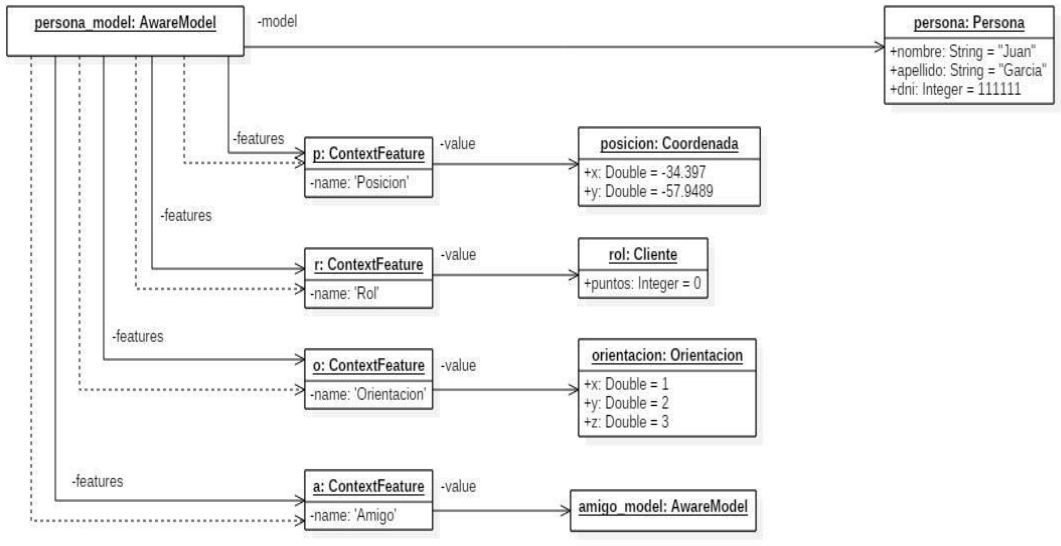


Figura 3.4.3: Instancia de un *AwareModel* que representa una *Persona*

- *AwareModel* que representa una *Sitio_Gastronomico* con contexto, para cada sitio se deberá instanciar un *AwareModel* con la *ContextFeature estado*, la cual deberá tomar un valor de la clase *Estado_Sitio* que se había definido en la Figura 3.2.8 (de la Sección 3.2).
- *AwareModel* que representa una *Promocion* con contexto, para cada promoción se deberá instanciar un *AwareModel* con la *ContextFeature periodos*, la cual deberá tomar un valor de la clase *Periodo_Validez* que se había definido en la Figura 3.2.8 (de la Sección 3.2).
- *AwareModel* que representa un *Consumo* con contexto, para cada consumo se deberá instanciar un *AwareModel* con la *ContextFeature estado*, la cual deberá tomar un valor de la clase *Estado_Consumo* que se había definido en la Figura 3.2.8 (de la Sección 3.2).

Para cada uno de los *AwareModel* descriptos anteriormente: *Sitio_Gastronomico*, *Promocion* y *Consumo*, se presentan en la Figura 3.4.4 las clases relacionadas con sus respectivos valores de contexto.

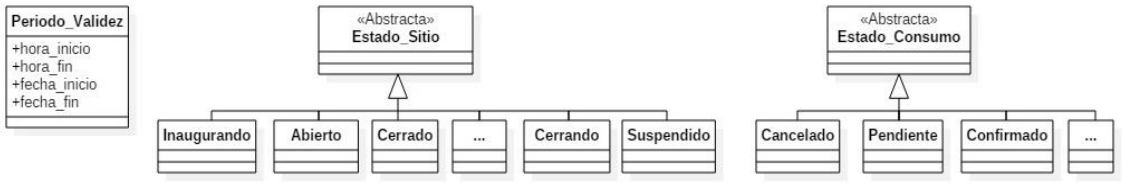


Figura 3.4.4[13]: Clases relacionadas con los valores de contexto de los *AwareModel* que representan un *Sitio_Gastronomico*, una *Promocion* y un *Consumo*

- **Modelo de Dominio Propuesto sin incluir las características de contexto**

  El modelo de dominio propuesto a continuación es una refactorización del modelo de *Sistema Gastronómico* presentado en la Sección 3.2 (Figura 3.2.8). Esta refactorización consistió en sacar todas aquellas características contextuales, como se fueron mencionando previamente en esta sección, por ejemplo, los servicios pasaron al nivel de contexto.

  A continuación se mencionan las refactorizaciones realizadas al modelo de la Figura 3.2.8, respecto a relaciones que ya no estarán presentes dado que las mismas se representan dinámicamente con los valores de contexto:

  - Relación *posicion* entre las clases *Persona* y *Posicion*.
  - Relación *rol_actual* entre las clases *Persona* y *Rol*.
  - Relación *orientacion* entre las clases *Persona* y *Orientacion*.

[13] Se observa en la Figura 3.4.4, que tanto en las jerarquías de las clases *Estado_Consumo* y *Estado_Sitio* existen subclases denominadas "…"; esto se especificó a fines de que el lector puede apreciar que estas jerarquías podrían crecer y contar con nuevas clases en un futuro, que por claridad, solo se especificaron algunas de ellos.

- Relación *amigos* de la clase *Persona* a sí misma.
- Relación *estado* entre las clases *Sitio_Gastronomico* y *Estado_Sitio*.
- Relación *estado* entre las clases *Consumo* y *Estado_Consumo*.

Otra refactorización (de la Figura 3.2.8), está relacionada con aquellas clases que son usadas solo para valores de contexto y por esa razón no se incluyen en esta refactorización del modelo de domino. Estas clases son:

- *Estado_Consumo*
- *Estado_Sitio*
- *Orientacion*

Cabe destacar que las clases *Rol*, *Posicion* y *Periodo_Validez* siguen estando presentes en esta refactorización del modelo de dominio dado que son conocidas por otras clases no de manera contextual, como por ejemplo: la clase *Rol* se sigue manteniendo debido a que el sistema necesita conocer todos los *roles* que tienen cada *Persona*, esto es diferente al *rol_actual* que posee cada una de las personas en un determinado momento. Algo similar ocurre con la posición de un sitio gastronómico, la cual se considera que no es contextual, sin embargo la posición del usuario si es contextual. En cuanto al periodo de validez sigue siendo conocido por las promociones que se fueron dando a lo largo del tiempo por cada sitio gastronómico. Mientras que a nivel contextual se dan las promociones cuyo periodo de validez está vigente.

Continuando con la refactorización del modelo, se quitan todas las relaciones referidas al patrón *Observer* [Gamma et al., 1995] que era utilizado en la Figura 3.2.8, como por ejemplo: la relación *observer* entre las clases *Sitio_Gastronomico* y *Consumo*. Esto se debe a que estas relaciones de *observer* estarán dadas a nivel del modelo de contexto.

Como se mencionó al principio de ésta sección, al describir el modelo de contexto, todas las clases relacionadas con los servicios (*Servicio*, *Estado_Servicio* y *Tipo_Servicio*) pasaron a formar parte del mismo, y por ende, en esta refactorización no son consideradas en el modelo de dominio.

En la Figura 3.4.5 se puede observar el modelo de dominio propuesto sin incluir características contextuales relacionadas a los servicios gastronómicos.

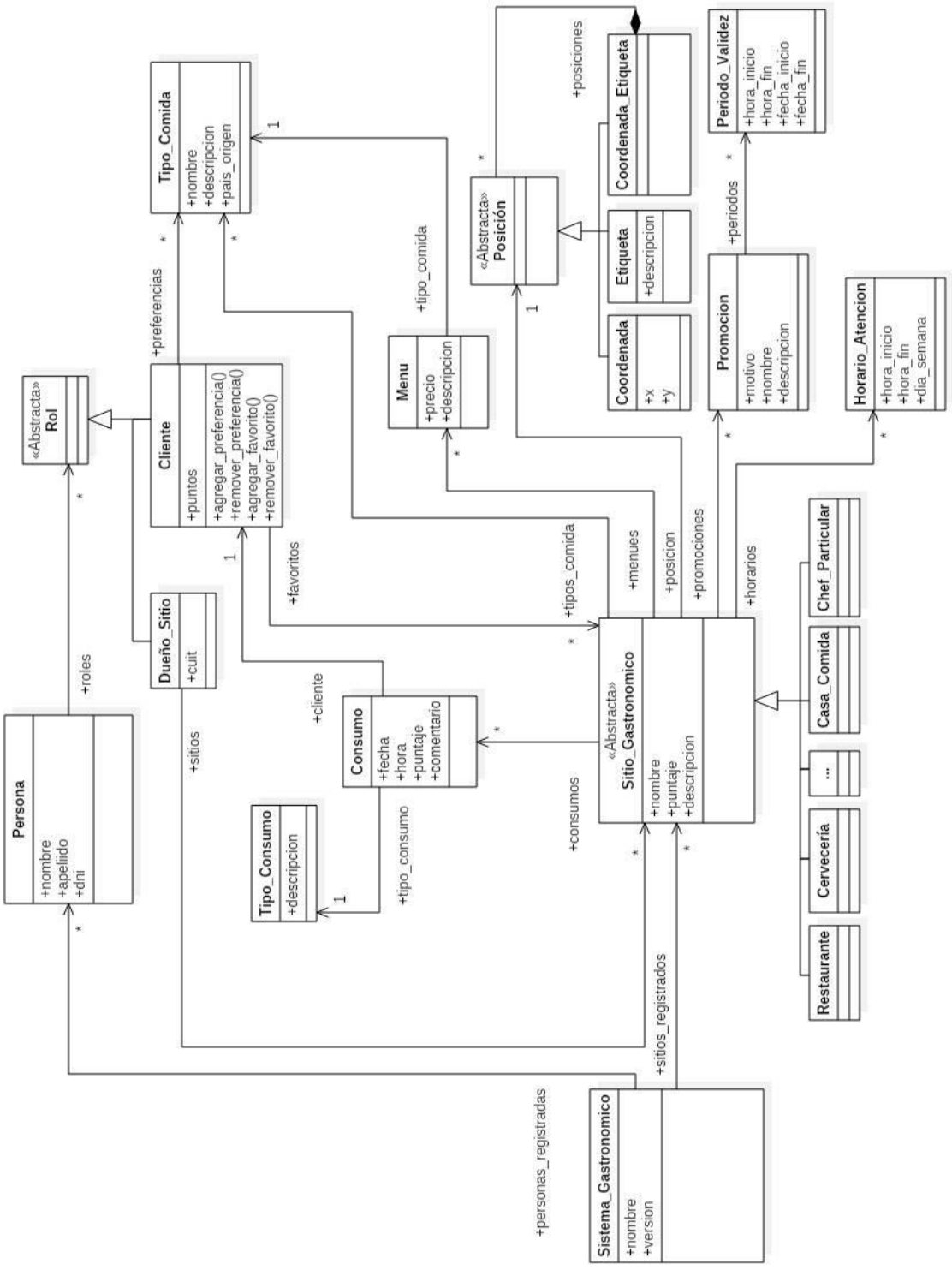


Figura 3.4.5[14]: Modelo de Dominio Propuesto sin incluir características contextuales

[14] Se observa en la Figura 3.4.5, que en la jerarquía de la clase *Sitio_Gastronomico* existe la subclase denominada "…"; esto se especificó a fines de que el lector puede apreciar que esta jerarquía podría crecer y contar con nuevas clases en un futuro, que por claridad, solo se especificaron algunas de ellas.

3.5 Funcionamiento del modelado propuesto considerando conceptos de contexto

En ésta sección se presentará un modelo de instancias de las principales clases pertenecientes a los modelos de: *Contexto* (Figura 3.4.1), *Valores del Contexto* (Figuras 3.4.2 y 3.4.4) y *Dominio* (Figura 3.4.5) presentados anteriormente. Además, se detallará el comportamiento de los principales métodos de dichos modelos a través de diagramas de secuencia. De esta manera, se podrá apreciar como los servicios son activados según cambien los valores del contexto actual.

Para lograr facilitar la comprensión del lector, el modelo de instancias de la Figura 3.5.1, se divide en tres columnas denominadas:

- *Instancias de las clases del modelo de contexto*, para indicar que son instancias de las clases del *Modelo de Contexto* presentado en la Figura 3.4.1.
- *Instancias de las clases que son solo usados como valores de contexto*, para indicar que son instancias de las clases del *Modelo de Valores de Contexto* presentado en las Figuras 3.4.2 y 3.4.4.
- *Instancias de las clases del modelo de dominio*, para indicar que son instancias de las clases del *Modelo de Dominio* presentado en la Figura 3.4.5.

En particular, se puede observar en la Figura 3.5.1 las siguientes instancias:

- *ambiente_sistema* (instancia de la clase *Ambiente_Servicio_Gastronomico*), la cual tiene como:
  - *model* un *sistema* (instancia de la clase *Sistema_Gastronomico*).
  - *subambiente* un *ambiente_sitio* (instancia de la clase *Ambiente_Servicio_Sitio*).
  - *awareObjects:*
    - *persona_model* (instancia de la clase *AwareModel*).
    - *amigo_model* (instancia de la clase *AwareModel*, no se dan más detalles de esta instancia porque tomaría valores similares a *persona_model*).
    - *sitio_mode*l (instancia de la clase *AwareModel*).

  Estos *awareObjects* además son observados por *ambiente_sistema*.
- *persona_model* (instancia de la clase *AwareModel*), la cual tiene como:
  - *model* una *persona* (instancia de la clase *Persona*).
  - *contextFeatures:*
    - *p* (instancia de la clase *ContextFeature*) que tiene como *value* a una instancia de la clase *Coordenada*.
    - *r* (instancia de la clase *ContextFeature*) que tiene como *value* a una instancia de la clase *Cliente*.
    - *o* (instancia de la clase *ContextFeature*) que tiene como *value* a una instancia de la clase *Orientacion*.
    - *a* (instancia de la clase *ContextFeature*) que tiene como *value* a una instancia de la clase *AwareModel* (la cual se denomina *amigo_model*).

Estas *contextFeatures* además son observadas por *persona_model*.

- *sitio_model* (instancia de la clase *AwareModel*), la cual tiene como:
  - *model* un *sitio* (instancia de la clase *Sitio_Gastronomico*).
  - *contextFeatures:*
    - *es* (instancia de la clase *ContextFeature*) que tiene como *value* a una instancia de la clase *Abierto*. Esta *contextFeature* además es observada.
- *ambiente_sitio* (instancia de la clase *Ambiente_Servicio_Sitio*), la cual tiene como:
  - *model* un *sitio* (instancia de la clase *Sitio_Gastronomico*).
  - *awareObjects:*
    - *promocion_model* (instancia de la clase *AwareModel*), la cual tiene como:
      - *model* un *promocion* (instancia de la clase *Promocion*).
      - *contextFeatures:*
        - *pv* (instancia de la clase *ContextFeature*) que tiene como *value* a una instancia de la clase *Periodo_Validez.* Esta *contextFeature* además es observada.
    - *consumo_model* (instancia de la clase *AwareModel*), la cual tiene como:
      - *model* un *consumo* (instancia de la clase *Consumo*).
      - *contextFeatures:*
        - *ec* (instancia de la clase *ContextFeature*) que tiene como *value* a una instancia de la clase *Confirmado.* Esta *contextFeature* además es observada.

Estos *awareObjects* además son observados por *ambiente_sitio*.

Se puede observar en la Figura 3.5.1 como una instancia de la clase *Sitio_Gastronomico* puede estar aumentada con contexto, pero además ser un *subambiente* donde existen elementos que también son aumentados con contexto, como son el consumo y las promociones. De esta manera, se pueden tener un ambiente general representado por una instancia de la clase *Ambiente_Servicio_Gastronomico*, y un *subambiente* representado por la clase *Ambiente_Servicio _Sitio.*

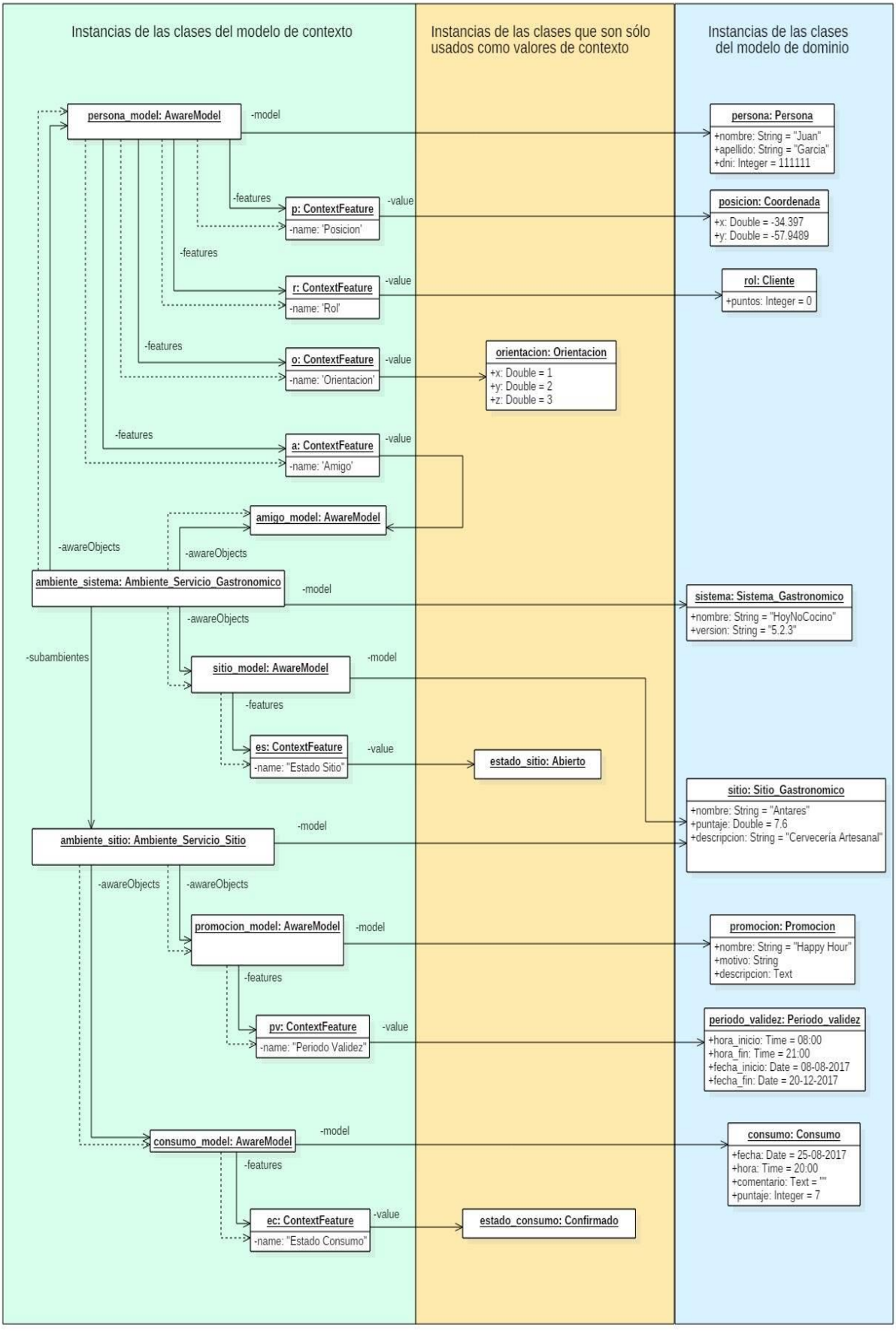


Figura 3.5.1: Ejemplo de posibles instancias de los modelos propuestos

Para continuar con ejemplos que faciliten la comprensión, se presenta en la Figura 3.5.2 un modelo de instancia que extiende el presentado en la Figura 3.5.1, con instancias de las clases del *Modelo de Contexto Propuesto* de la Sección 3.4 (Figura 3.4.1). En particular, se instanciarán servicios generales como la visualización de sitios gastronómicos, la notificación de estados de un pedido y relacionados a cambios en las promociones. Para esta instanciación, dichos servicios son para la instancia de *persona_model*.

En particular, se puede observar en la Figura 3.5.2 las siguientes instancias:

- *ambiente_sistema* (ya presentada en la Figura 3.5.1), la cual tiene además:
  - *handlers* un *handler_posicion* (instancia de la clase *ServicesHandler*), el cual tiene como:
    - *spec* una *spec_posicion* (instancia de la clase *ServiceSpec*).
    - *consumers* un *consumer_persona* (instancia de la clase *ServiceConsumer*), el cual tiene como:
      - *registeredSpecs* una *spec_posicion* (la misma instancia de *ServiceSpec* que tiene el *handler_posicion*).
      - *activeServices* un *service_visualizar_sitio* (instancia de la clase *Visualizar_Sitio*).
      - *awareObject* una *persona_model* (ya presentada en la Figura 3.5.1).
  - *availableSpecs* una *spec_posicion* (la misma instancia de la clase *ServiceSpec* que tiene *handler_posicion*).
  - *consumers* un *consumer_persona* (la misma instancia de la clase *ServiceConsumer* que tiene el *handler_posicion*).
- *ambiente_sitio* (ya presentada en la Figura 3.5.1), la cual tiene como:
  - *handlers*:
    - *handler_notificar_promocion* (instancia de la clase *ServicesHandler*), el cual tiene como:
      - *spec* una *spec_notificar_promocion* (instancia de la clase *ServiceSpec*).
      - *consumer_promocion* (instancia de la clase *ServiceConsumer*), el cual tiene como:
        - *registeredSpecs* una *spec_notificar_promocion* (la misma instancia de *ServiceSpec* que tiene el *handler_notificar_promocion*).
        - *activeServices* un *service_notificar_promocion* (instancia de la clase *Notificar_Promocion*).
        - *awareObject* una *persona_model* (instancia de la clase *AwareModel*).
    - *handler_estado_consumo* (instancia de la clase *ServicesHandler*), el cual tiene como:
      - *spec* una *spec_estado_consumo* (instancia de la clase *ServiceSpec*).
      - *consumer_consumo* (instancia de la clase *ServiceConsumer*), el cual tiene como:
        - *registeredSpecs* una *spec_estado_consumo* (la misma instancia de *ServiceSpec* que tiene el *handler_estado_consumo*).
        - *activeServices:*

        - *service_notificar_pedido_listo* (instancia de la clase *Notificar_Pedido_Listo*).
        - *service_notificar_pedido_cancelado* (instancia de la clase *Notificar_Pedido_Cancelado*).
    - *awareObject* una *persona_model* (instancia de la clase *AwareModel*).
- *availableSpecs:*
    - *spec_notificar_promocion* (la misma instancia de la clase *ServiceSpec* que tiene el *handler_notificar_promocion*).
    - *spec_estado_consumo* (la misma instancia de la clase *ServiceSpec* que tiene el *handler_notificar_consumo*).
- *consumers:*
    - *consumer_promocion* (la misma instancia de la clase *ServiceConsumer* que tiene el *handler_notificar_promocion*).
    - *consumer_consumo* (la misma instancia de la clase *ServiceConsumer* que tiene el *handler_notificar_consumo*).

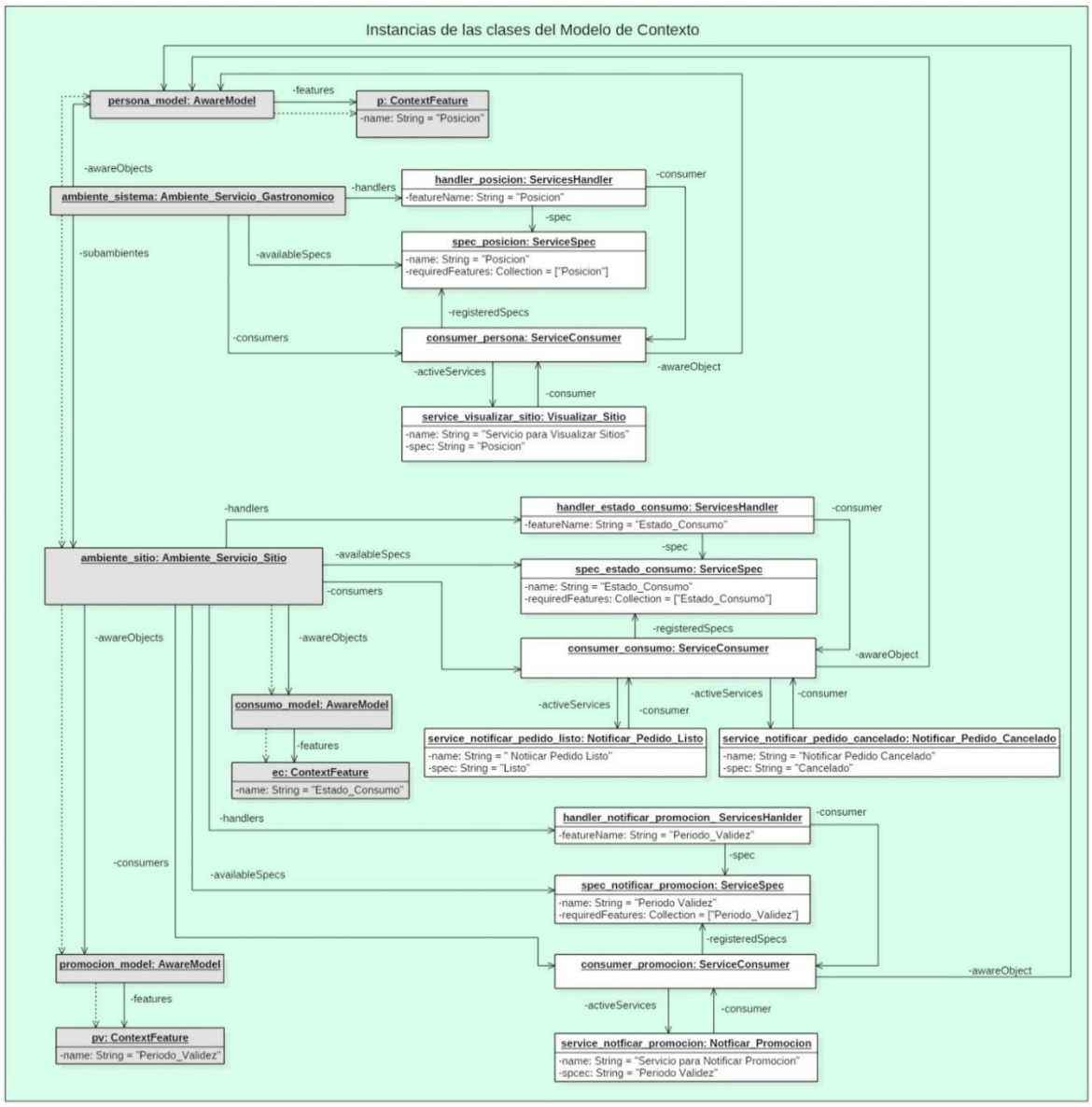


Figura 3.5.2: Ejemplo de instancias de servicios de los modelos propuestos

De la Figura 3.5.2 se desea resaltar que, la instancia *persona_model* se ve aumentada con servicios. Es decir, mediante los *ServiceConsumer* instanciados (*consumer_consumo, consumer_persona y consumer_promocion*) se logra representar que los servicios son para un usuario, y que los mismos conozcan a su *AwareObject,* ya sea por ejemplo, para brindarle información o para modificar algún valor del mismo. Otra característica interesante de esta instanciación es que *consumer_consumo* provee dos servicios asociados, estos serán activados acorde al estado actual del consumo como se mostrará más adelante.

A continuación, se presentará a través de diagramas de secuencia como son activados algunos servicios ante un cambio de contexto:

- *Servicio Visualizar Sitios*

  En la Figura 3.5.3 se puede apreciar el diagrama de secuencia asociado al servicio de visualizar sitios. A continuación se detallan cada uno de los pasos relacionados a dicho diagrama.

  1. La ejecución comienza a partir del cambio en la posición del usuario. Dicha posición se asume que es obtenida del GPS que posee el dispositivo del usuario.
  2. A partir de aquí se produce un cambio en el valor de la *ContextFeature* instanciada (con el nombre de *p*), en este caso una coordenada, que provoca que la instancia del *AwareModel* (en este caso *persona_model*) que lo está observando reaccione.
  3. Posteriormente, dicho *AwareModel* notificará que se ha cambiado la posición del usuario al *Ambiente_Servicio_Gastronomico* (en este caso la instancia *ambiente_sistema),* a través del método `notify(p, persona_model).`
  4. Luego, el *Ambiente_Servicio_Gastronomico* se comunica con cada uno de sus *handlers*, para que sea manipulado dicho cambio de contexto, a través del método `handle(p, persona_model, ambiente_sistema).`
  5. Cada *ServicesHandler* que hace una comprobación para identificar si como *featureName* tiene la característica de contexto que cambio *(*en este caso el nombre p).
  6. Si se cumple las comprobaciones del punto anterior, el *ServicesHandler* (*handler_posicion*) delega a su *ServiceConsumer* (*consumer_persona*) para que se encargue de ejecutar los servicios correspondientes al cambio de contexto producido.
  7. Luego, el *ServiceConsumer* (en este caso *consumer_persona*) comprobará que la *ServiceSpec* (*spec_posicion*) está incluida dentro de sus *registeredSpecs* (acción que se realiza mediante el método `comprobarRegisteredSpec()`). De ser así, se continúa con el siguiente punto.
  8. El *ServiceConsumer* ejecuta su método `serviceFor(spec_posicion, p)` para delegar la activación de los servicios correspondientes.
  9. Dentro del método `serviceFor(spec_posicion, p)` se recorre a todos los *activeServices* que tiene el *ServicesHandler* (en este caso *handler_posicion),* y se le pregunta a cada servicio si cumple con las especificaciones. Dicha

comprobación se realiza a través del método `comprobarServiceSpec(spec_posicion, p)`.

10. Cada servicio sabe comprobarse a sí mismo, si cumple o no con la *ServiceSpec* (*spec_posicion*) y/o C*ontextFeature* (*p*) que le envía el *ServiceConsumer* (en este caso *consumer_persona*). Dicha comprobación retorna un booleano al *ServiceConsumer* que envió el mensaje `comprobarServiceSpec()`.
11. Finalmente, el *ServiceConsumer* (*consumer_persona*) ejecutará los servicios que hayan retornado *true* (en la comprobación del punto anterior) a través del método `start(p,persona_model,ambiente_sistema,consumer_persona.awareObject)` enviando cómo parámetro al *ContextFeature* (*p*)*, AwareModel (persona_model), Ambiente_Servicio_Gastronomico* (*ambiente_sistema*) y al *AwareObject* que posee el *ServiceConsumer* (*consumer_persona.awareObject*).
12. Cómo último paso, el servicio *Visualizar_Sitio,* procederá a accionarse mostrando los sitios al usuario que corresponda a través del método *mostrar_sitios*().

- *Servicio Notificar Promoción*

  En la Figura 3.5.4 se puede apreciar la secuencia de mensajes que se desencadenan cuando el Dueño de algún sitio gastronómico realiza una acción concreta. A diferencia del diagrama de secuencia de la Figura 3.5.3, en el cual la cadena de mensajes se iniciaba mediante un cambio de posición detectado por un sensor GPS.

  Cabe destacar, que tanto en Figura 3.5.3 como en la Figura 3.5.4., se usa el modelo genérico de contexto presentado en la Sección 3.4 (Figura 3.4.1). Debido a esto, la cadena de mensajes de cómo se propaga el cambio de contexto es la misma para ambos servicios, pero cada uno tiene su *ServiceHandler* que maneja dichos cambios de contexto. De esta manera, se puede apreciar como para dos cambios de contextos diferentes, el *ServiceHandler* que maneja cada uno se obtiene dinámicamente.

  Los pasos del 2 al 12 son similares en ambas figuras (Figura 3.5.3 y Figura 3.5.4). Para la Figura 3.5.4, el *ServiceConsumer* instanciado interactúa con el servicio *Notificar_Promocion*, para luego generar la notificación a todos los usuarios que corresponda. Como se mencionó anteriormente, en el último paso de ésta cadena de mensajes se le notifica la promoción a varios usuarios, mientras que en la Figura 3.5.3 solo se ejecutaba el servicio *Visualizar_Sitio* para un solo usuario.

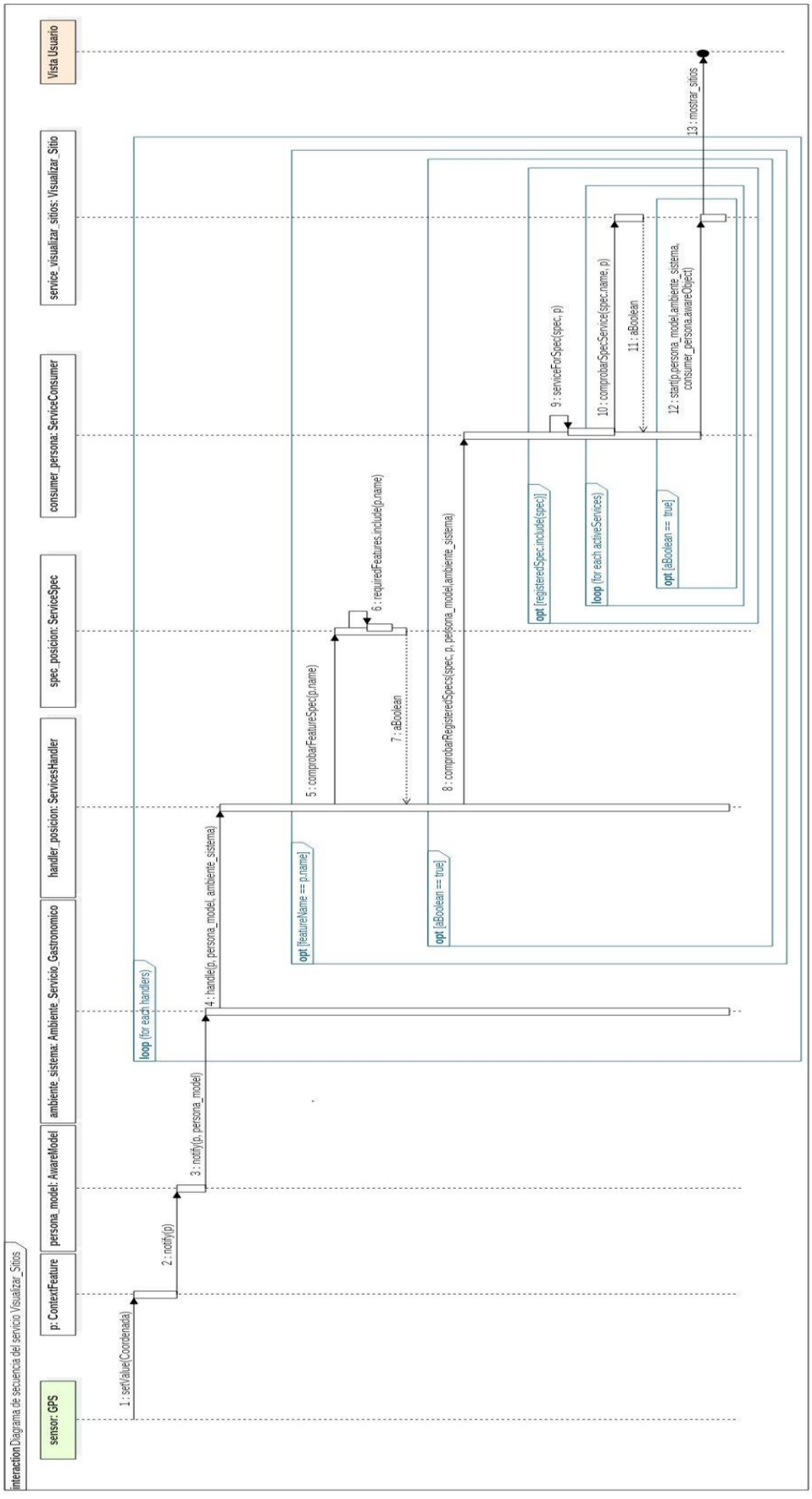


Figura 3.5.3: Diagrama de secuencia del servicio *Visualizar Sitios*

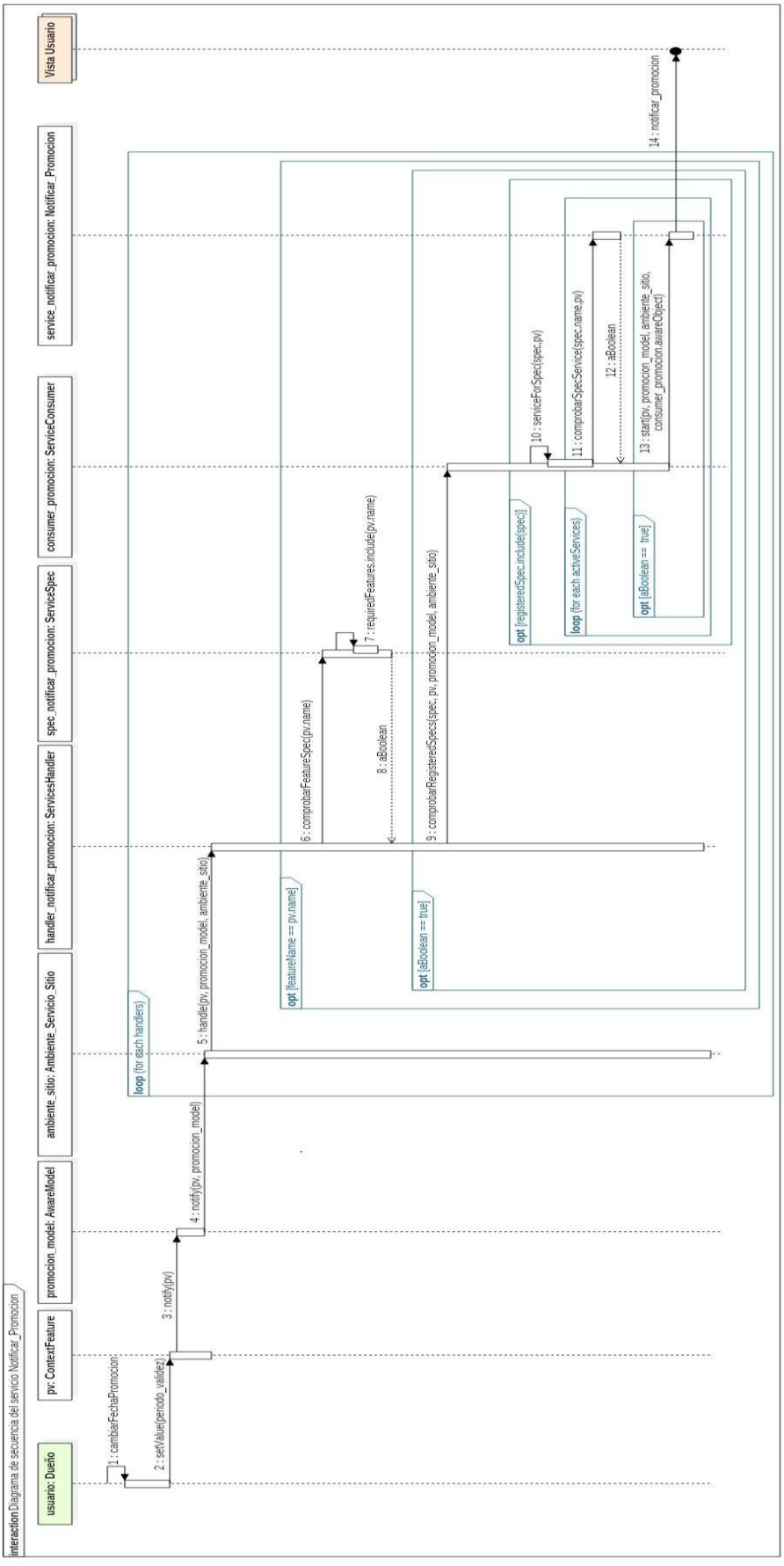


Figura 3.5.4: Diagrama de secuencia del servicio *Notificar Promoción*

- *Servicio Notificar Pedido Listo*

  En el diagrama que se presenta en la Figura 3.5.5, donde la secuencia de mensajes es similar a la presentada en la Figura 3.5.4, el evento es iniciado por un usuario *Dueño* de algún sitio, el cual cambia el estado de un pedido a "*Listo*". Al finalizar, se notifica al cliente que su pedido se encuentra listo.

  Cómo se puede apreciar en la Figura 3.5.5, en este caso es otro *ServiceHandle* quién maneja el evento disparado por el *Dueño*, como así también, es otro el servicio el que se ejecuta (en este caso *Notificar_Pedido_Listo)*, lo que permite seguir observando la ejecución dinámica del modelo propuesto para este trabajo.

- *Servicio Notificar Pedido Cancelado*

  El diagrama que se presenta en la Figura 3.5.6, es muy similar al presentado en la Figura 3.5.5, pero difiere en que el inicio de la propagación de mensajes, lo genera un usuario *Cliente* cancelando su pedido y este cambio de estado, es notificado al *Dueño* del sitio donde se realizó dicho pedido.

  A diferencia de las Figuras 3.5.3 y 3.5.4, en este caso es el mismo *ServiceConsumer* tanto en la Figura 3.5.5 como en la Figura 3.5.6. Y es éste *ServiceConsumer* (*consumer_consumo*), quién toma la solicitud y selecciona el servicio para que sea ejecutado, en este caso es ejecutado el servicio *Notificar_Pedido_Cancelado* (mientras que en la Figura 3.5.5 se había ejecutado el servicio *Notificar_Pedido_Listo*). Ésta selección, que realiza el *ServiceConsumer*, la efectúa mediante la comprobación del *value* del *ContextFeature* que ha cambiado.

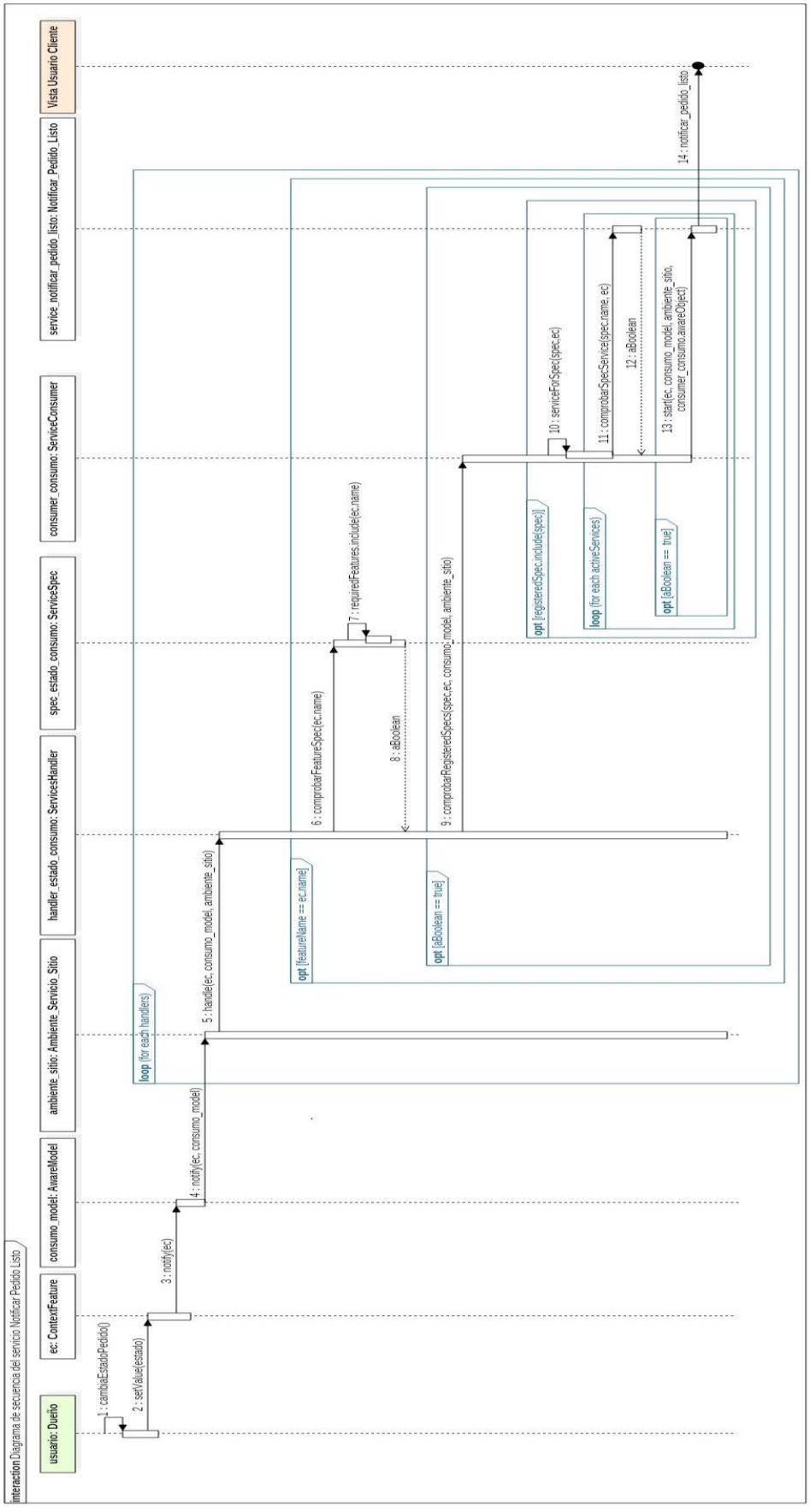


Figura 3.5.5: Diagrama de secuencia del servicio *Notificar Pedido Listo*

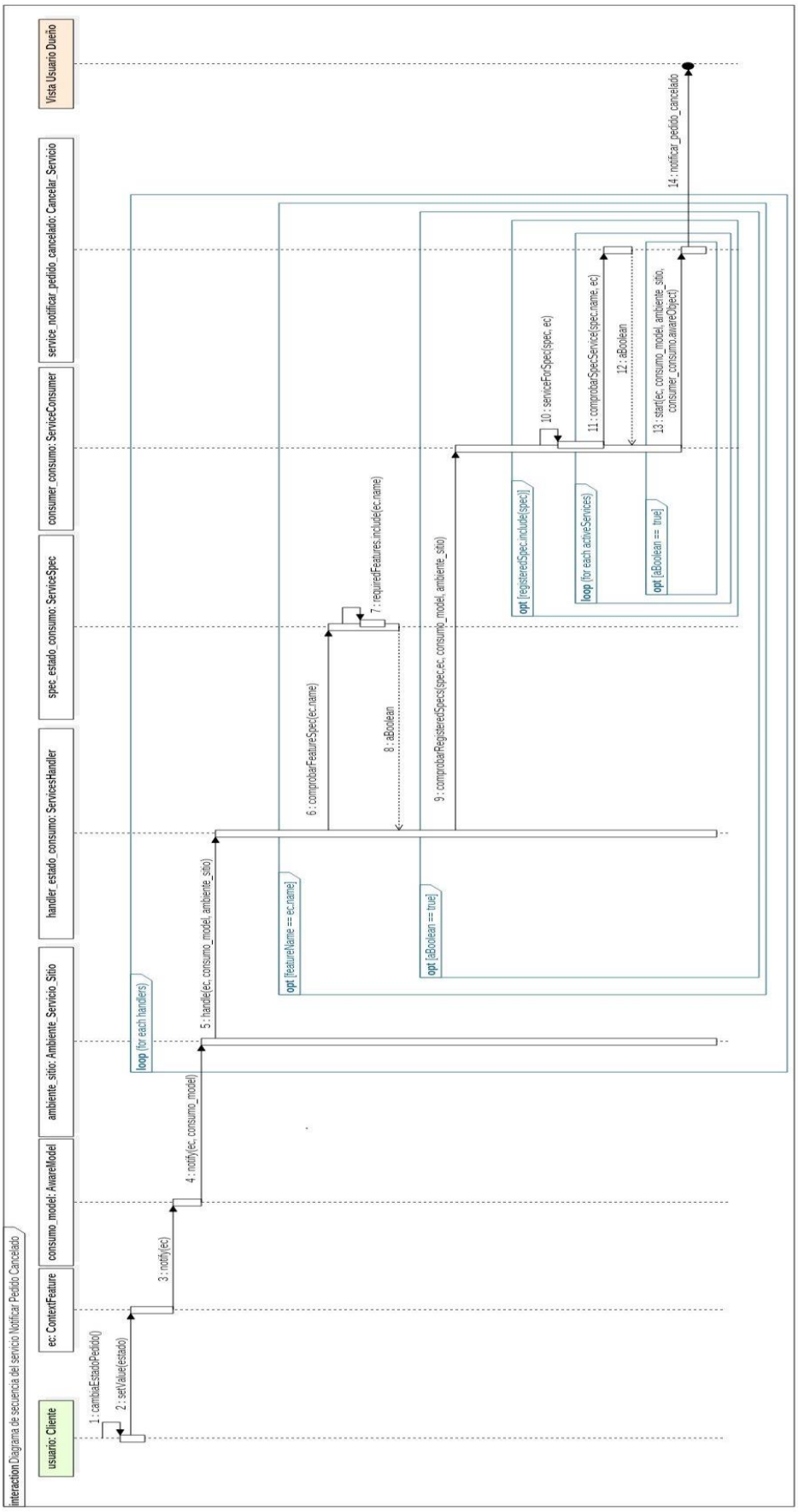


Figura 3.5.6: Diagrama de secuencia del servicio *Notificar Pedido Cancelado*

4. Implementación del Prototipo

En este capítulo se describen las características relacionadas a la implementación del prototipo. Se enuncian las tecnologías utilizadas para el mismo. Además, se especifica el estado inicial y sus principales funcionalidades. Al finalizar el capítulo, se presentan las problemáticas que se encontraron durante el desarrollo, y como se dio solución a las mismas.

El foco del prototipo consiste en mostrar cómo funcionan ciertos servicios contextuales que se presentarán a lo largo de este capítulo. Este prototipo se desarrolló como una aplicación *web responsive* que usa de base el modelo propuesto en la Sección 3.4.

4.1 Características generales del prototipo

En esta sección se presentan las características del prototipo desarrollado, se enuncian las tecnologías utilizadas para su implementación, se describen los dispositivos que fueron utilizados para el desarrollo y pruebas del mismo.

El desarrollo del prototipo consistió en dos etapas, las cuales se describen a continuación destacando las tecnologías utilizadas en cada una de estas.

- En la etapa previa a la codificación, se configuró el entorno de desarrollo usando lo siguiente:
    - *WampServer* 3.0.6 [WampServer, 2004], el cual nos proveyó de:
        - *Apache*[15] 2.4.23 como servidor web.
        - *MySQL*[16] 5.7.14 [MySQL, 1995] como gestor de base de datos.
        - PHP[17] 5.6.25 [PHP, 1995] como lenguaje de programación.
        - *phpMyAdmin* 4.6.4 [phpMyAdmin, 1998] para administrar *MySQL*. Con dicha herramienta se creó la base de datos con sus tablas correspondientes.
    - El repositorio con *Bitbicket*[18] y *Git-Flow*[19] 1.10.1.

  En esta etapa, también se realizaron *mockups* con la herramienta *MockingBot* 3.0 [MockingBot]. Estos se pueden apreciar en el Anexo B.

---

[15] Página de *Apache:* http://tomcat.apache.org (Último acceso 01-02-2018)

[16] Es un sistema de gestión de base de datos relacional (RDBMS) de código abierto, basado en lenguaje de consulta estructurado (SQL). MySQL se ejecuta en prácticamente todas las plataformas, incluyendo Linux, UNIX y Windows.

[17] Es un lenguaje de código abierto muy popular especialmente adecuado para el desarrollo web y que puede ser incrustado en HTML.

[18] Página de *Bitbicket:* https://bitbucket.org (Último acceso 01-02-2018)

[19] Página de *Git-Flow:* https://datasift.github.io/gitflow/IntroducingGitFlow.html (Último acceso 01-02-2018)

- En la etapa de codificación, se utilizaron las siguientes tecnologías:
    - *Bootstrap* 3.3.5 [Bootstrap, 2011] y *CSS3* [CSS, 1996], para realizar el diseño del prototipo.
    - *HTML5* [HTML5, 2014], para presentar el contenido web.
    - *Google Maps JavaScript AP*[20] 3.0, para crear mapas, mostrar los marcadores necesarios y obtener la geolocalización a través del GPS del dispositivo actual.
    - *AJAX*[21].
    - *Web Services RESTful*[22].

Finalmente, se subió el prototipo a un servidor web para que pueda ser accedido desde distintos dispositivos sin la necesidad de instalar un servidor local, la IP de acceso es 107.180.3.49. Dicho servidor cuenta con lo siguiente:

- *MySQL* 5.6.36 [MySQL, 1995] como gestor de base de datos.
- *PHP* 5.6 [PHP, 1995] como lenguaje de programación.
- *phpMyAdmin* 4.0.10.18 [phpMyAdmin, 1998] para administrar MySQL.

El prototipo implementado se basa en el modelo propuesto en el Capítulo 3. Para poder guardar la información del mismo se decidió utilizar una base de datos. En la Figura 4.1.1 se muestra la base de datos creada "*hoy_no_cocino*" con sus tablas y atributos correspondientes:

[20] Página de *Google Maps JavaScript API:* https://developers.google.com/maps/documentation/javascript/?hl=es-419 (Último acceso 01-02-2018)

[21] Página de *w3schools*: https://www.w3schools.com/xml/ajax_intro.asp (Último acceso 19-02-2018)

[22] Página de *dosideas*: https://dosideas.com/noticias/java/314-introduccion-a-los-servicios-web-restful (Último acceso 19-02-2018)

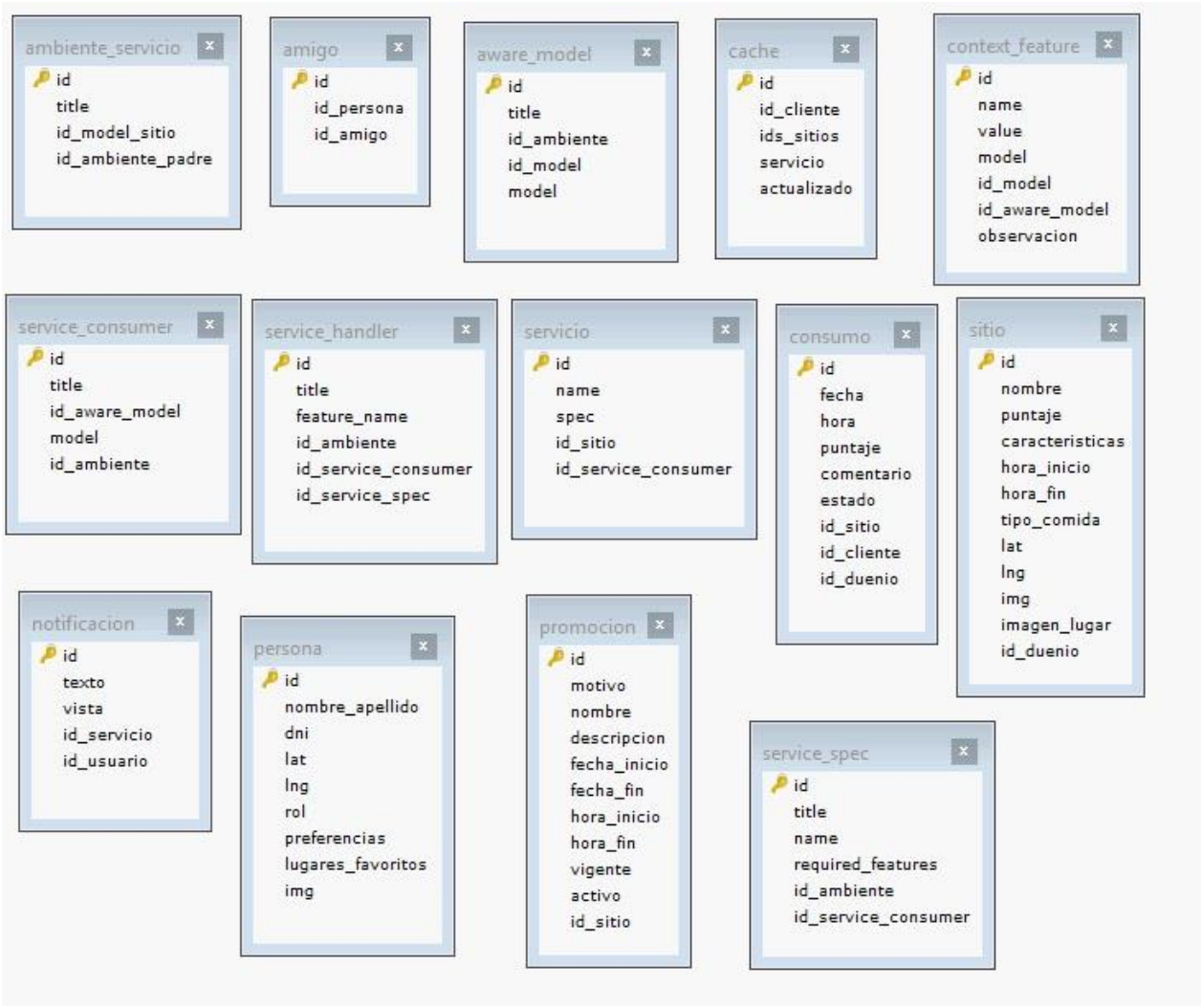


Figura 4.1.1: Estructura de la base de datos *hoy_no_cocino*

Para poder realizar las pruebas funcionales del prototipo se utilizaron los siguientes dispositivos:

- Dos notebook Dell, con las siguientes especificaciones:
    - *Dell Inspiron 5559 Signature Edition* con sistema operativo Windows 10 Home, pantalla de 15” pulgadas y una resolución de 1080x1920 píxeles.
    - *Dell Inspiron M731R* con sistema operativo Windows 10 Home y Linux Mint, pantalla de 17” pulgadas y una resolución de 1600x900 pixeles.
- Dos celulares, con las siguientes especificaciones:
    - *Motorola G4 Plus* con sistema operativo *Android* 7.0, pantalla de 5.5” pulgadas y una resolución de 1080X1920 píxeles.
    - *Samsung Galaxy A5 2016 SM-A510M* con sistema operativo *Android* 7.0, pantalla de 5.2” pulgadas y una resolución de 1080x1920 píxeles.

Las pruebas se hicieron en una primera etapa en un entorno local con las notebooks mencionadas. El entorno de desarrollo local, consistió en montar el prototipo en un servidor

apache local provisto por la herramienta *WampServer* [WampServer, 2004] descripta anteriormente. En una segunda etapa de prueba, además de las notebooks como dispositivos de prueba se incluyeron los dos celulares. Para acceder desde todos estos dispositivos, se montó el prototipo en el servidor web mencionado anteriormente, cuya IP de acceso es 107.180.3.49. Cabe mencionar que en ambos entornos, se requiere una óptima conexión a internet y permitir la utilización del GPS a través del navegador; de no ser así, el prototipo se encuentra limitado y no brinda todas las funcionalidades como se espera.

4.2 Estado Inicial de Prototipo

El estado inicial del prototipo hace referencia no solo a los contextos e información definida, sino también a los valores de contexto con los cuales se usan inicialmente al ejecutar por primera vez el prototipo. En base a estos valores, los distintos servicios contextuales reaccionan. Esto permite que las pruebas funcionales del prototipo sean más agiles y se focalicen en poder apreciar el comportamiento de los servicios contextuales.

A continuación se detallan los valores que toman los distintos usuarios definidos para el prototipo:

- *Usuario*: Gonzalo Rojas (cliente) con los siguiente datos:
    - *Posición*: latitud -34.9290595 y longitud -57.927044.
    - *Rol*: cliente.
    - *Preferencias*: parrilla.
    - *Amigos*:
        - Laura Suarez.
        - María Martínez.
    - *Consumos*:
        - Consumo #1.
        - Consumo #2.
        - Consumo #4.
- *Usuario*: Laura Suarez (cliente) con los siguiente datos:
    - *Posición*: latitud -34.921007 y longitud -57.952857.
    - *Rol*: cliente.
    - *Consumos*:
        - Consumo #3.
        - Consumo #5.
- *Usuario*: María Martínez (cliente) con los siguiente datos:
    - *Posición*: latitud -34.9219055196 y longitud -57.960154.
    - *Rol*: cliente.
- *Usuario*: Marcos Gutiérrez (dueño) con los siguiente datos:
    - *Rol*: dueño.
    - *Sitio*: Artares.

A continuación se lista la información de los consumos que tienen los usuarios, se puede observar como cada uno de estos tiene su estado actual, dependiendo del mismo es como se comportaran los servicios contextuales inicialmente:

- *Consumo*: Consumo #1 con los siguiente datos:
    - *Fecha y hora*: 25/03/2018 a las 20:30hs.
    - *Estado*: Pendiente.
    - *Sitio*: Artares.
    - *Cliente*: Gonzalo Rojas.
- *Consumo*: Consumo #2 con los siguiente datos:
    - *Fecha y hora*: 19/12/2017 a las 22:30hs.
    - *Estado*: Pendiente.
    - *Sitio*: Artares.
    - *Cliente*: Gonzalo Rojas.
- *Consumo*: Consumo #3 con los siguiente datos:
    - *Fecha y hora*: 21/11/2017 a las 21:00hs.
    - *Estado*: Cancelado.
    - *Sitio*: Artares.
    - *Cliente*: Laura Suarez.
- *Consumo*: Consumo #4 con los siguiente datos:
    - *Fecha y hora*: 25/10/2017 a las 14:00hs.
    - *Estado*: Confirmado.
    - Sitio: Artares.
    - *Cliente*: Gonzalo Rojas.
- *Consumo:* Consumo #5 con los siguiente datos:
    - *Fecha y hora*: 20/12/2017 a las 23:00hs.
    - *Estado*: Confirmado.
    - *Sitio*: Artares.
    - *Cliente*: Laura Suarez.

A continuación se lista la información de los sitios gastronómicos precargados en el prototipo. Se puede observar como cada uno de estos tiene su puntaje, horario, tipo de comida, posición, dependiendo de los valores de los mismos es como se comportaran los servicios contextuales para cada cliente.

- *Sitio*: Artares con los siguiente datos:
    - *Puntaje*: 7.6.
    - *Horario*: 19:00 a 02:00 Hs.
    - *Tipo de Comida*: Cervecería, Hamburguesería
    - *Posición*: latitud -34.9016 y longitud -57.9488.
    - *Dueño*: Marcos Gutiérrez.

- *Sitio*: Chichara con los siguiente datos:
    - *Puntaje*: 7.
    - *Horario*: 20:00 a 04:00 Hs.
    - *Tipo de Comida*: Parrilla.
    - *Posición*: latitud -34.9114 y longitud -57.9888.
- *Sitio*: La Tratopia con los siguiente datos:
    - *Puntaje*: 8.6.
    - *Horario*: 20:00 a 01:00 Hs.
    - *Tipo de Comida*: Restaurante, Vegetariano.
    - *Posición*: latitud -34.921007 y longitud -57.952857.
- *Sitio*: Pleta con los siguiente datos:
    - *Puntaje*: 7.5.
    - *Horario*: 19:00 a 02:00 Hs.
    - *Tipo de Comida*: Cervecería, Pizzería.
    - *Posición*: latitud -34.7466 y longitud -57.6555.
- *Sitio*: La Candombersa con los siguiente datos:
    - Puntaje: 6.3.
    - Horario: 20:00 a 01:00 Hs.
    - Tipo de Comida: China.
    - Posición: latitud -34.926968 y longitud -57.954404.
- *Sitio*: Baviesa con los siguiente datos:
    - *Puntaje*: 8.
    - *Horario*: 12:00 a 00:00 Hs.
    - *Tipo de Comida*: Fastfood.
    - *Posición*: latitud -34.926334 y longitud -57.961786.
- *Sitio*: Ocho Sabió con los siguiente datos:
    - *Puntaje*: 9.2.
    - *Horario*: 12:00 a 01:00 Hs.
    - *Tipo de Comida*: Pizzería.
    - *Posición*: latitud -34.926369 y longitud -57.952688.
- *Sitio*: Blothers con los siguiente datos:
    - *Puntaje*: 7.8.
    - *Horario*: 21:00 a 02:00 Hs.
    - *Tipo de Comida*: Gourmet.
    - *Posición*: latitud -34.923273 y longitud -57.955048.
- *Sitio*: Runieses con los siguiente datos:
    - *Puntaje*: 9.
    - *Horario*: 20:30 a 01:00 Hs.
    - *Tipo de Comida*: Pastas.
    - *Posición*: latitud -34.904129 y longitud -57.937624.

- *Sitio*: Grac con los siguiente datos:
    - *Puntaje*: 8.4.
    - *Horario*: 21:00 a 01:00 Hs.
    - *Tipo de Comida*: Gourmet.
    - *Posición*: latitud -34.918558 y longitud -57.947538.

A continuación se lista la información de las promociones precargadas en el prototipo. Se puede observar como cada una de estas tiene un periodo de validez, dependiendo de esto es como se comportaran los servicios contextuales para cada cliente.

- *Promoción*: Promoción #1 con los siguiente datos:
    - *Motivo*: Happy Hour
    - *Nombre*: Promoción 2X1
    - *Descripción*: No te pierdas nuestro 2X1 de lunes a jueves.
    - *Fecha y hora inicio*: 22/11/2017 a las 17:00 Hs.
    - *Fecha y hora fin*: 22/11/2018 a las 23:00 Hs.
    - *Vigente*: Sí.
    - *Sitio a la que pertenece*: La Tratopia
- *Promoción*: Promoción #2 con los siguiente datos:
    - *Motivo*: Navidad
    - *Nombre*: Descuento del 50%.
    - *Descripción*: Accedé a estos descuentos increíbles con tu debito visa.
    - *Fecha y hora inicio*: 20/12/2017 a las 10:00 Hs.
    - *Fecha y hora fin*: 25/12/2018 a las 23:00 Hs.
    - *Vigente*: No.
    - *Sitio a la que pertenece*: Artares
- *Promoción*: Promoción #3 con los siguiente datos:
    - *Motivo*: Año Nuevo.
    - *Nombre*: Descuento del 35% en efectivo.
    - *Descripción*: Nuevo año, nuevos descuentos para vos !.
    - *Fecha y hora inicio*: 01/01/2018 a las 00:00 Hs.
    - *Fecha y hora fin*: 30/03/2018 a las 23:00 Hs.
    - *Vigente*: Sí.
    - *Sitio a la que pertenece*: Artares.
- *Promoción*: Promoción #4 con los siguiente datos:
    - *Motivo*: Combo Otoñal
    - *Nombre*: Combo de Pizza + Pinta.
    - *Descripción*: Aprovecha nuestra promo otoñal, dos combos al precio de uno.
    - *Fecha y hora inicio*: 20/03/2018 a las 21:00 Hs.
    - *Fecha y hora fin*: 21/06/2018 a las 23:00 Hs.
    - *Vigente*: Sí.
    - *Sitio a la que pertenece*: Artares.

4.3 Principales Funcionalidades del Prototipo

A lo largo de ésta sección se podrán apreciar las funcionalidades implementadas según los perfiles disponibles: *cliente* y *dueño* de sitio gastronómico. La funcionalidad del cliente se mostrará accediendo desde un celular mientras que la funcionalidad del dueño se mostrará desde una notebook. Si bien ambos perfiles pueden ser accedidos desde ambos dispositivos, se decidió realizarlo de esta manera para poder apreciar como la información puede ser visualizada en estos dos dispositivos.

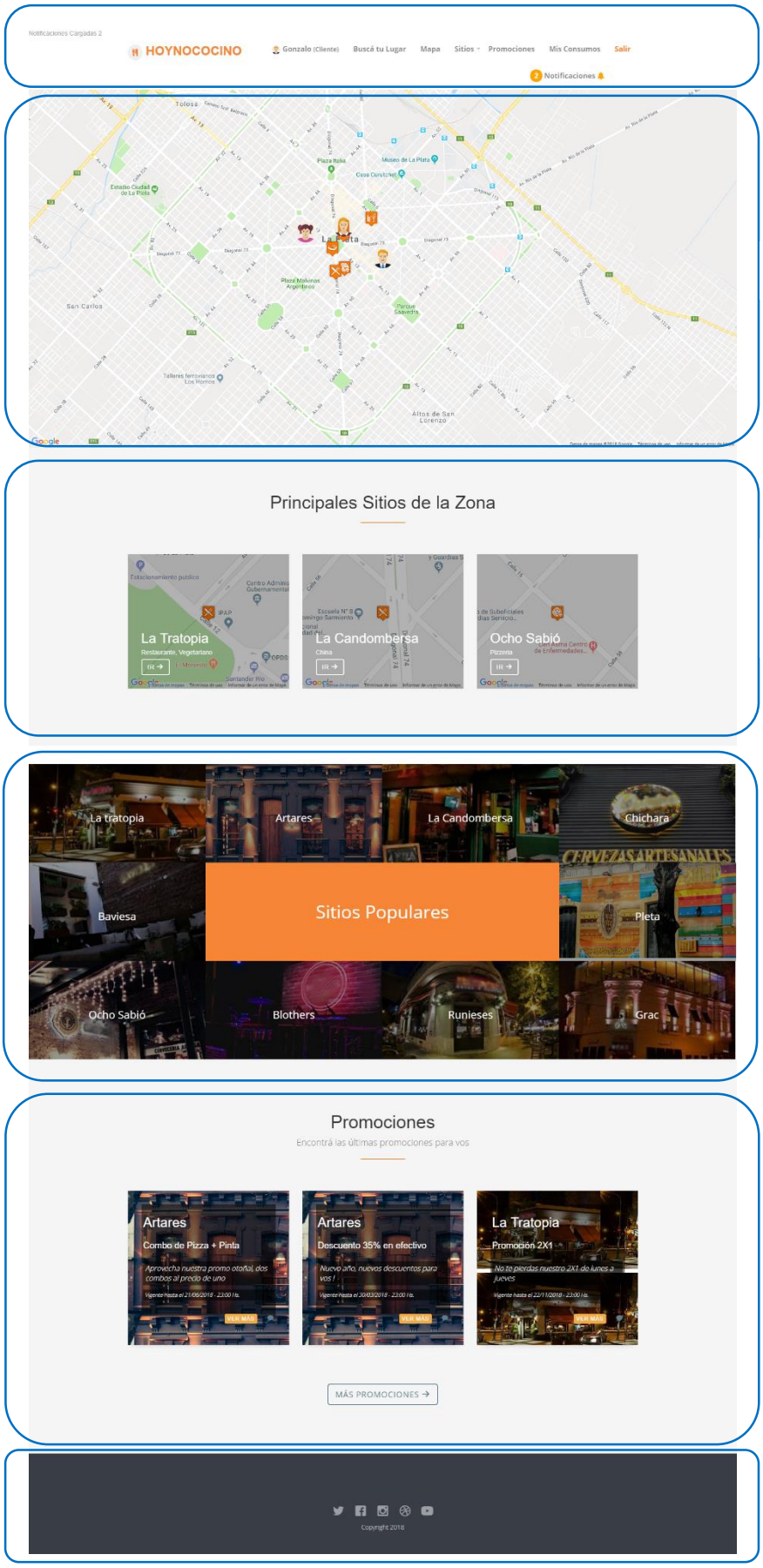
- **Funcionalidades para el perfil *Cliente***

  Al ingresar al prototipo, se presenta la opción para iniciar bajo el perfil de *cliente o dueño* de sitio. Al seleccionar la opción de "*Ingresar como Cliente*" como se puede visualizar en la Figura 4.3.1, se podrá acceder a las funcionalidades habilitadas para el mismo.

Figura 4.3.1: Pantalla Inicio Prototipo

Una vez que el usuario selecciona ingresar con el perfil de *cliente*, como se puede observar en la Figura 4.3.2, se visualiza la pantalla principal con las principales secciones que tiene disponible: *Menú, Mapa, Principales Sitios en la zona, Sitios Populares, Promociones* y *Pie de página*.

Para una mejor apreciación de todas las opciones con las que cuenta el *Cliente*, la pantalla de la Figura 4.3.2 se muestra accedida desde una notebook. En el caso del celular, el usuario tiene las mismas opciones y tiene que moverse con el *scroll* para poder ver las mismas.



Menú

Mapa

Principales Sitios en la zona

Sitios Populares

Promociones

Pie de página

Figura 4.3.2: Pantalla principal del perfil *Cliente*

Para acceder a las diferentes secciones, como se observa en la Figura 4.3.2, el prototipo cuenta con un menú (*superior*) donde se visualizan las opciones que se tiene para navegar en el mismo. El usuario puede seleccionar la opción que requiera haciendo *click* (o tocando) sobre la misma para que el prototipo visualice la pantalla correspondiente.

Cuando se accede desde un dispositivo móvil, el menú se encuentra en la parte superior derecha de la pantalla principal con un icono como se puede observar en la Figura 4.3.3. Al seleccionar dicho menú, se despliegan las opciones disponibles, estas se pueden observar en la Figura 4.3.3. En esta figura, se presenta el menú con las opciones que tiene el cliente *Gonzalo Rojas*:

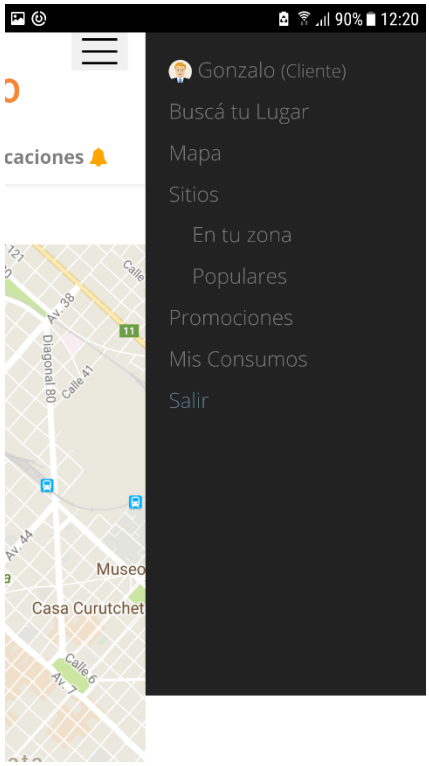


Figura 4.3.3: Menú para el cliente *Gonzalo Rojas*

Se puede observar en la Figura 4.3.3 las siguientes opciones de menú:

- *Perfil del Cliente* (el nombre y foto perfil del usuario dependerá de quien esté logueado)
- *Buscá tu Lugar*
- *Mapa*
- *Sitios*
  - *En tu Zona*
  - *Populares*
- *Promociones*
- *Mis Consumos*
- *Salir*

Antes de explicar en detalle las secciones que pueden ser accedidas a través del menú presentado en la Figura 4.3.3, se aclara que hay ciertas funcionalidades que no están implementadas en el prototipo debido a que escapa a lo que se quiere demostrar en el en el mismo.

El objetivo de que haya botones y ciertas acciones sin funcionalidad, es para brindar la posibilidad visualizar una idea más real y completa de lo que podría realizar el usuario a futuro. En estos casos, se explicará en detalle que es lo que no involucra al prototipo y se informará visualizando el cartel "*Funcionalidad no desarrollada para el prototipo*" como se puede observar en la parte superior de la Figura 4.3.4.

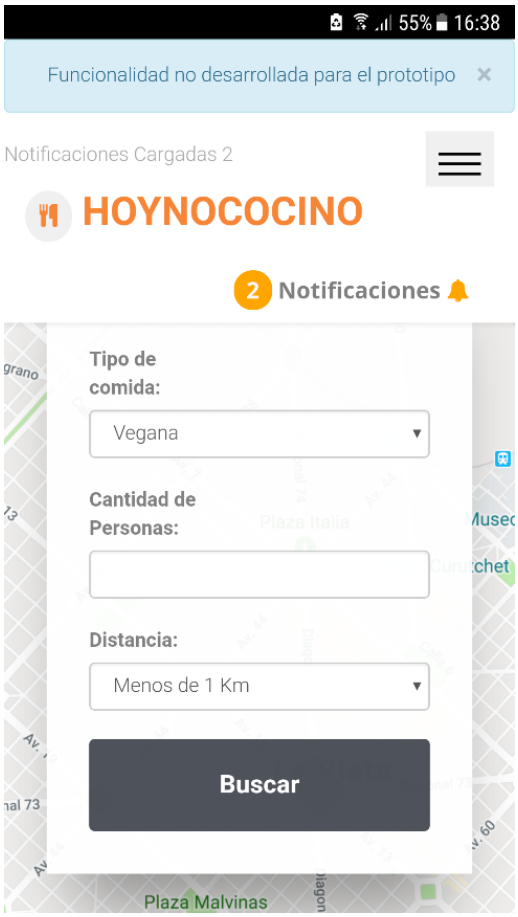


Figura 4.3.4: Funcionalidad no Implementada

A continuación se explicarán en detalle las opciones desplegadas en el menú para el perfil *cliente* que se presentó en la Figura 4.3.3:

- ***Perfil Cliente***

  Al seleccionar la opción "*Perfil Cliente*" (de la Figura 4.3.3) el usuario podrá visualizar su perfil, configurar sus preferencias gastronómicas, sitios preferidos y lista de amigos. En la Figura 4.3.5 se puede apreciar como el usuario visualiza sus configuraciones:

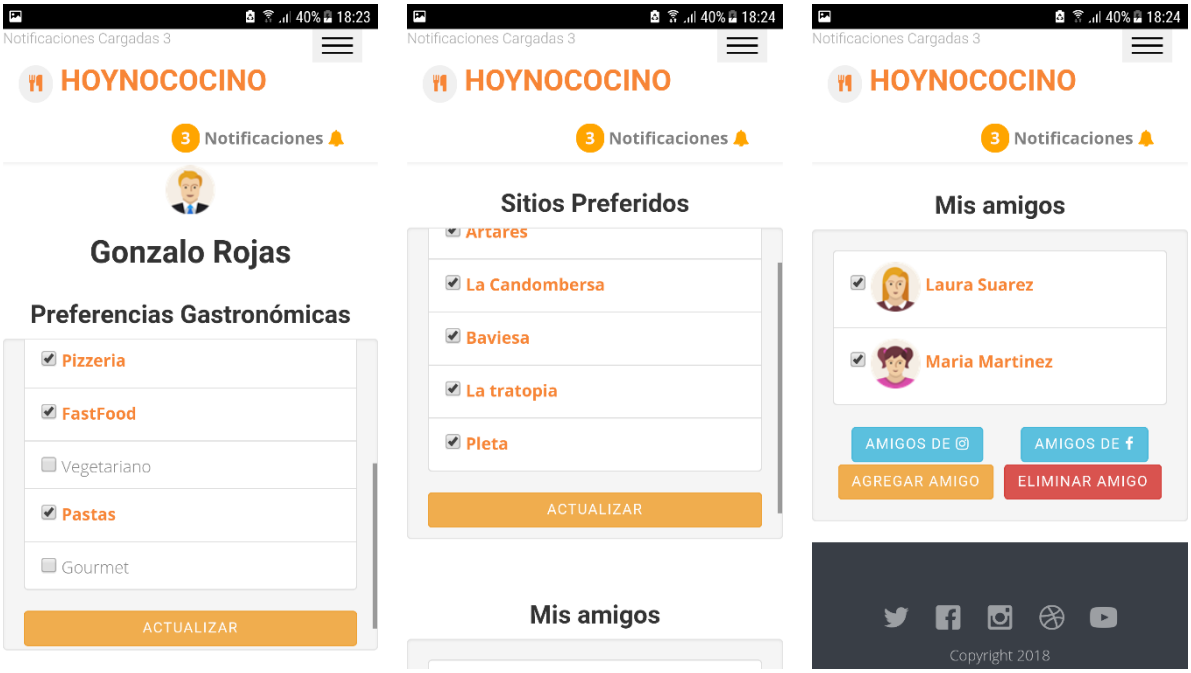


Figura 4.3.5: Preferencias Gastronómicas, Sitios Preferidos y Amigos

Para agregar o quitar preferencias gastronómicas, el usuario debe tildar/destildar las que requiera y luego presionar el botón "*Actualizar*". De manera similar funcionaría la configuración para los sitios preferidos y amigos[23].

- ***Buscá Tu Lugar***

  La segunda opción listada en la Figura 4.3.3, es *"Buscá Tu Lugar"*, esta es una opción que provee la búsqueda personalizada de sitios y promociones aplicando filtros, como por ejemplo el tipo de comida, cantidad de personas que arribarían al lugar, distancia máxima entra la posición actual y la de los sitios, entre otros. En la Figura 4.3.6 se puede observar cómo se visualiza lo mencionado cuando el usuario selecciona la opción del menú "*Buscar Tu Lugar*"[24].

[23] Esta funcionalidad no está implementada porque excede del alcance de lo que se quiere mostrar en la tesina, al seleccionar "*Actualizar*" el usuario recibe el cartel "*Funcionalidad no desarrollada para el prototipo*" como se presentó en la Figura 4.3.4.

[24] La funcionalidad del botón "*Buscar*" no está implementada porque excede del alcance de lo que se quiere mostrar en la tesina, al seleccionar dicha opción el usuario recibe el cartel "*Funcionalidad no desarrollada para el prototipo*" como se presentó en la Figura 4.3.4.

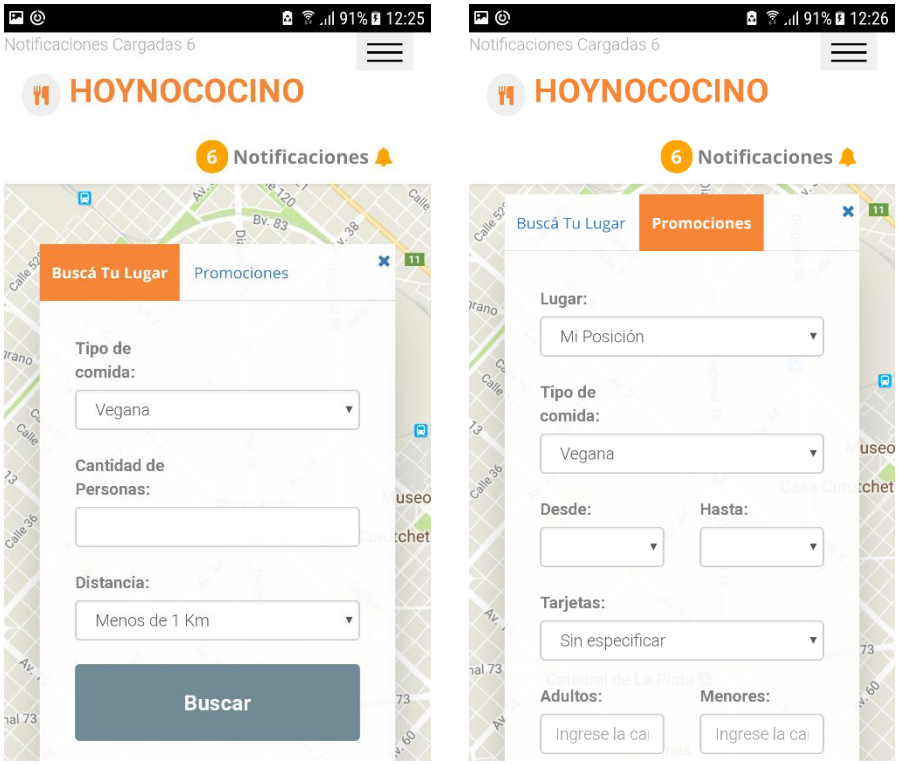


Figura 4.3.6: Buscar Sitios o Promociones

- ***Mapa***

  En la Figura 4.3.3 la tercera opción es "*Mapa*", al seleccionarla el usuario visualiza el mapa, en el cual se muestra a través de marcadores al usuario principal, a sus amigos y sitios gastronómicos cercanos. Dicha cercanía se da si sitios gastronómicos se encuentran en un área que se establece en un radio de 1000 metros a la posición actual del usuario.

  Esto se puede observar en la Figura 4.3.7, en este caso el usuario logeado "*Gonzalo Rojas*", donde se lo representa con un marcador que tiene su imagen de perfil, como así también a sus dos amigas (*Laura Suarez y María Martínez*) y a los sitios gastronómicos cercanos. En la figura no se llega a apreciar pero los marcadores de los amigos, se mueven para poder distinguir si alguno de ellos se encuentra en algún negocio cercano. Se detectó que si los iconos quedaban fijos, éstos podían llegar a ocultar información y/o a superponerse entre ellos. Esta fue una forma inicial para abordar esto en el prototipo.

  Cabe mencionar que a medida que el usuario se mueve y cambia su posición actual, este servicio se va actualizando considerando los parámetros de cercanía mencionados. Es decir, esta opción es contextual al usuario *logueado*, y por lo tanto, los amigos y sitios cercanos se van actualizando constantemente.

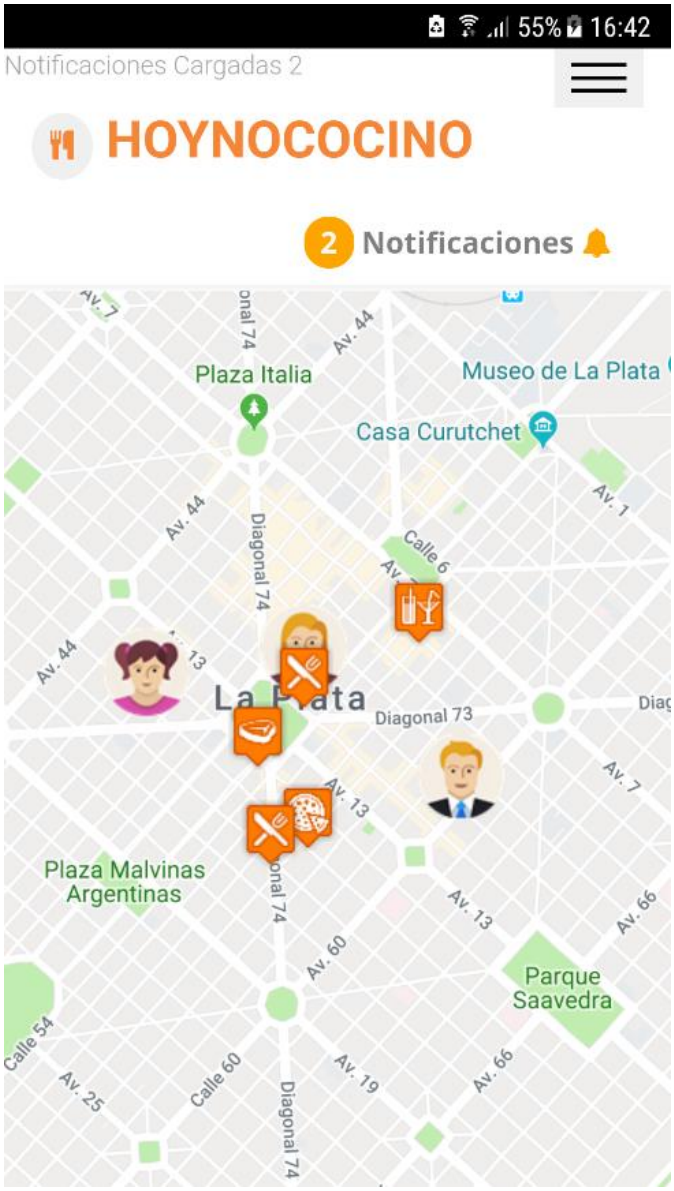


Figura 4.3.7: Mapa y Notificaciones

Como se mencionó anteriormente, tanto al usuario *logueado*, sus amigos y sitios cercanos se los visualiza en el mapa a través de marcadores. Dichos marcadores poseen la funcionalidad que al seleccionarlos muestran la siguiente información:

- *Marcador del usuario logueado*, visualiza la posición actual del usuario *logueado* representada con coordenadas: latitud y longitud.
- *Marcador para amigos*, visualiza el nombre del amigo y su posición representada con coordenadas: latitud y longitud.
- *Marcador para sitios*, visualiza el nombre del sitio, su puntaje expresado a través de estrellas, y su posición representada con coordenadas: latitud y longitud.

En la Figura 4.3.8 se puede observar la información que se visualiza al seleccionar cada uno de los marcadores anteriormente descriptos.

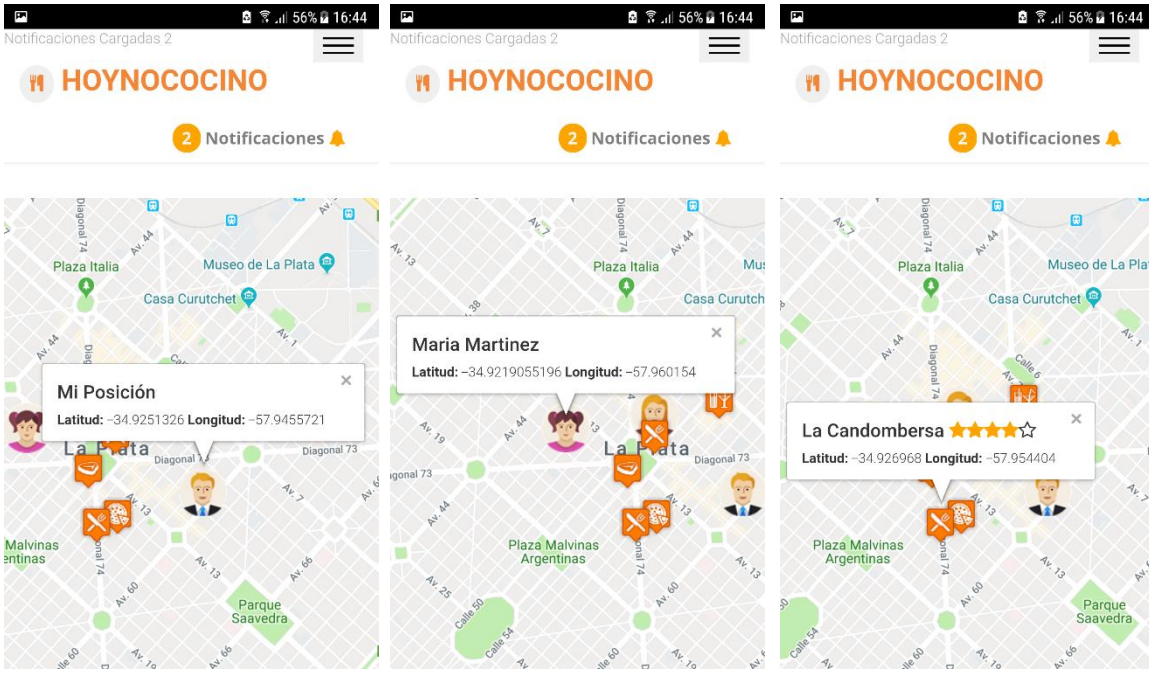


Figura 4.3.8: Detalles al seleccionar los marcadores

- ***Sitios en tu Zona***

  A elegir la opción *Sitios en tu Zona* (de la Figura 4.3.3), el usuario puede visualizar los sitios más cercanos a la posición actual del mismo. Por defecto, se visualizan los tres sitios gastronómicos más cercanos, y en el caso de que el usuario no se encuentre cerca de ningún sitio, dicha sección no se visualiza. En la Figura 4.3.9 se observa un solo sitio cercano a la zona del usuario.

Figura 4.3.9: Principales sitios cercanos

- ***Sitios Populares***

  A elegir la opción *Sitios Populares* (de la Figura 4.3.3), se pueden visualizar los sitios mejores puntuados y más concurridos por los clientes. Dicha sección se visualiza en la Figura 4.3.10. Esto se recibe en forma de lista para que el usuario conozca los sitios más populares.

Figura 4.3.10: Sitios más Populares

- ***Promociones***

  Al seleccionar la opción de Promociones (de la Figura 4.3.3), el usuario tendrá una vista rápida para conocer las últimas tres promociones brindadas por sitios gastronómicos.

  De cada promoción, se visualiza a que sitio pertenece, una descripción que explica de qué se trata, la vigencia de la misma y un botón "*ver más*"[25].

  En la Figura 4.3.11 se visualiza la sección de *Promociones*, hay dos pantallas ya que el usuario tiene que moverse con el *scroll* en la misma para ir viendo la información de cada una.

[25] La funcionalidad del botón "*Ver más*" no está implementada porque excede del alcance de lo que se quiere mostrar en la tesina, al seleccionar "dicha opción el usuario recibe el cartel "*Funcionalidad no desarrollada para el prototipo*" como se presentó en la Figura 4.3.4.

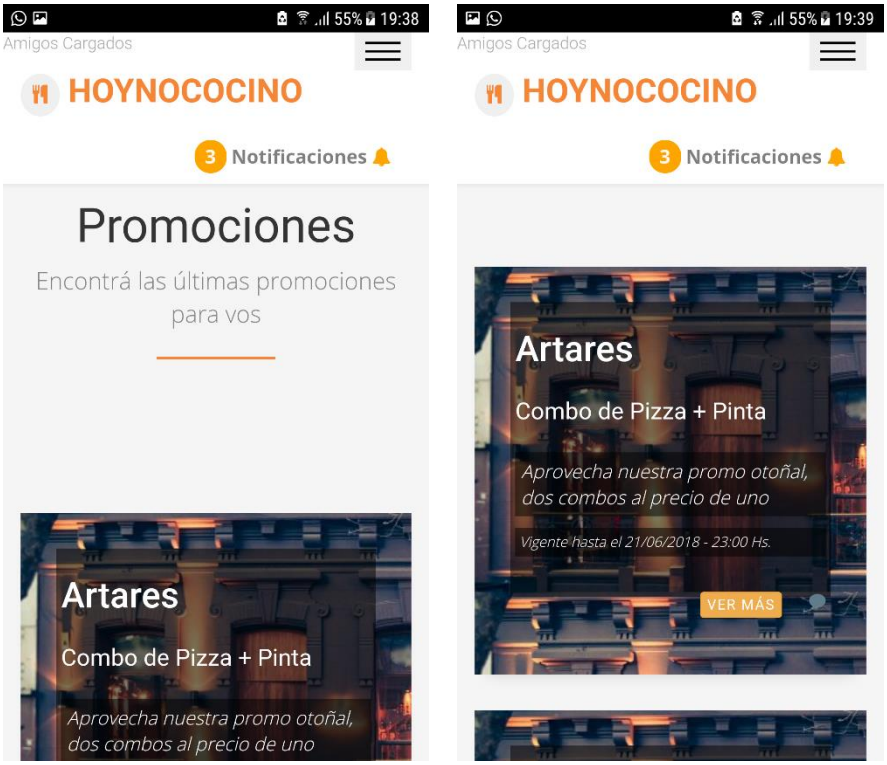


Figura 4.3.11: Últimas promociones

- ***Mis consumos***

  Al seleccionar la opción de *"Mis Consumos"* (de la Figura 4.3.3), se le da la posibilidad al usuario de acceder a sus consumos, visualizando cada uno de ellos con sus principales características. Cabe mencionar que se entiende por consumo a cualquier servicio gastronómico que el usuario haya realizado, como puede ser un pedido, una reserva de mesa, anotarse en una lista de espera u otro servicio que el sitio gastronómico ofrezca. En la Figura 4.3.12 se puede observar los pedidos que tiene el cliente *Gonzalo Rojas.*

Figura 4.3.12: Consumos del cliente *Gonzalo Rojas*

Como se puede apreciar en la Figura 4.3.12, los datos principales que aparecen en el listado de consumos son: *el número de consumo, fecha y hora, puntaje (calificación), sitio en el cual realizó el consumo* y *estado* (el cual puede ser *Pendiente, Listo, Confirmado o Cancelado*).

Por otro lado, también se provee las opciones para conocer más detalles de los consumos y para cancelarlos a través de los botones "*Ver más*" [26] y "*Cancelar*" respectivamente. La opción para cancelar el consumo solo está disponible cuando el mismo está en estado de *Pendiente*.

- ***Salir***

  Al seleccionar la opción para *salir* (de la Figura 4.3.3) se vuelve al inicio del prototipo visualizando en pantalla la Figura 4.3.1.

- **Funcionalidades para el perfil *Dueño***

  Como se mencionó anteriormente al principio de esta sección, al ingresar al prototipo se presenta la opción para iniciar con perfil de *cliente* o *dueño* de sitio como se puedo visualizar en la Figura 4.3.1.

  Al seleccionar la opción para ingresar con el perfil de *dueño*, se visualiza la pantalla principal como se puede observar en la Figura 4.3.13, donde se pueden visualizar las distintas secciones para dicho perfil: *Menú, Características del sitio gastronómico* (incluye posición del mismo en un mapa), *Promociones*, *Servicios que ofrece el sitio gastronómico, Comentarios de los clientes* y *Pie de página*.

[26] La funcionalidad del botón "*Ver más*" no está implementada porque excede del alcance de lo que se quiere mostrar en la tesina, al seleccionar dicha opción el usuario recibe el cartel "*Funcionalidad no desarrollada para el prototipo*" como se presentó en la Figura 4.3.4.

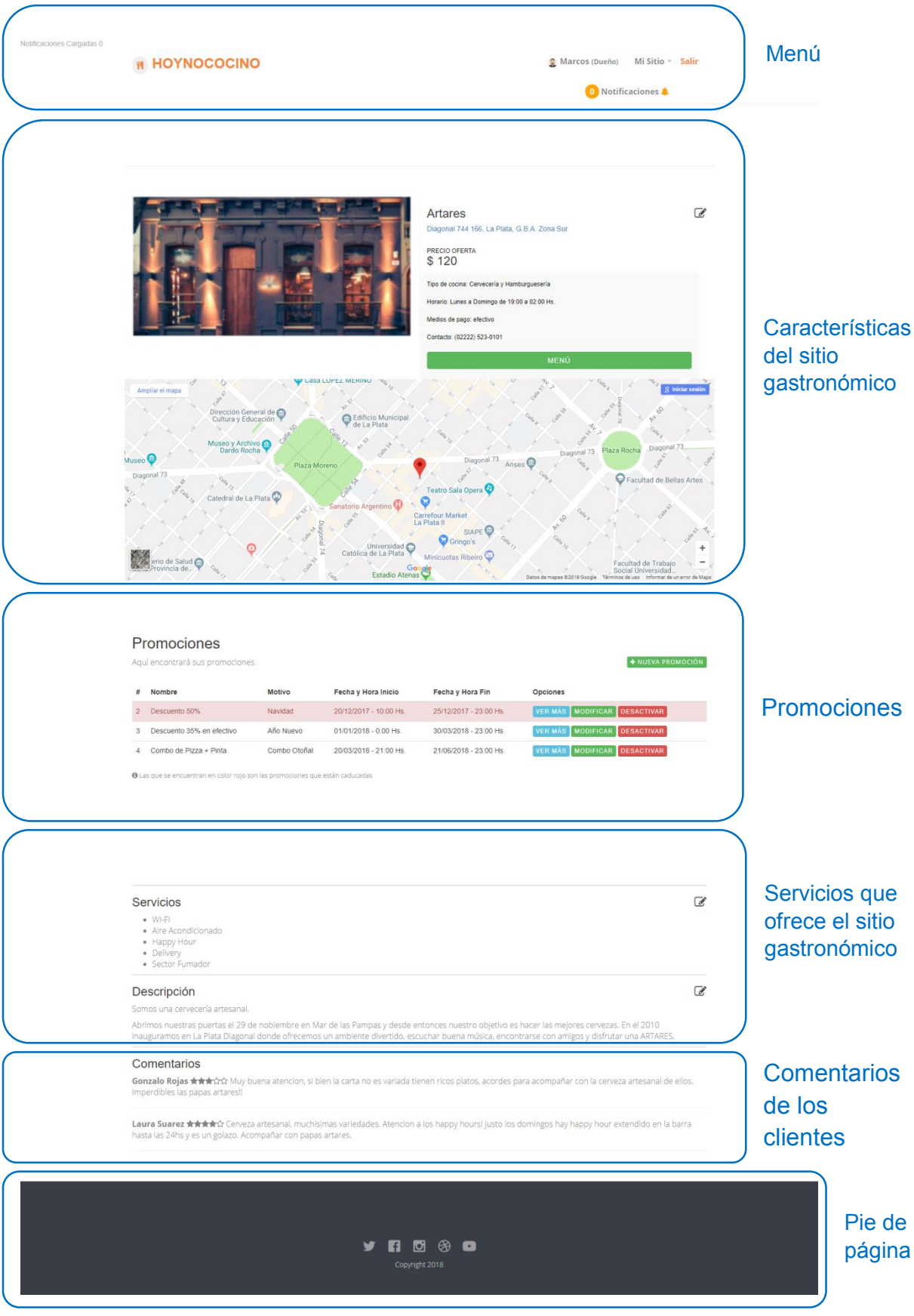


Figura 4.3.13: Pantalla principal del perfil *Dueño*

Para acceder a las diferentes secciones, como se observa en la parte superior de la Figura 4.3.13, el prototipo cuenta con un menú donde se visualizan las opciones que se tiene para navegar en el mismo. El usuario puede seleccionar la opción que requiera haciendo *click* sobre la misma.

En la Figura 4.3.14 se presenta el menú con las opciones que tiene el dueño *Marcos Gutiérrez*, estas son:

- *Perfil del Dueño* (el nombre y foto perfil del usuario dependerá de quien esté logueado)
- *Pedidos*
- *Servicios*
- *Promociones*
- *Salir*

Figura 4.3.14: Menú para el dueño Marcos Gutiérrez

A continuación se explicarán en detalle las opciones desplegadas en el menú para el perfil *dueño* que se presentó en la Figura 4.3.14:

- ***Perfil Dueño***

  Al seleccionar esta opción (de la Figura 4.3.14), el dueño podrá visualizar la pantalla principal de su sitio como se presentó en la Figura 4.3.13, donde podrá ver y editar[27] las características de su sitio gastronómico. Entre los datos más relevantes, se encuentra la imagen del sitio, horarios, tipo de cocina, datos de contacto, menú gastronómico disponible y ubicación del mismo. En la Figura 4.3.15 se puede observar lo recientemente descripto.

[27] La funcionalidad del botón para editar no está implementada porque excede del alcance de lo que se quiere mostrar en la tesina, al seleccionar dicha opción el usuario recibe el cartel "*Funcionalidad no desarrollada para el prototipo*" como se presentó en la Figura 4.3.4.

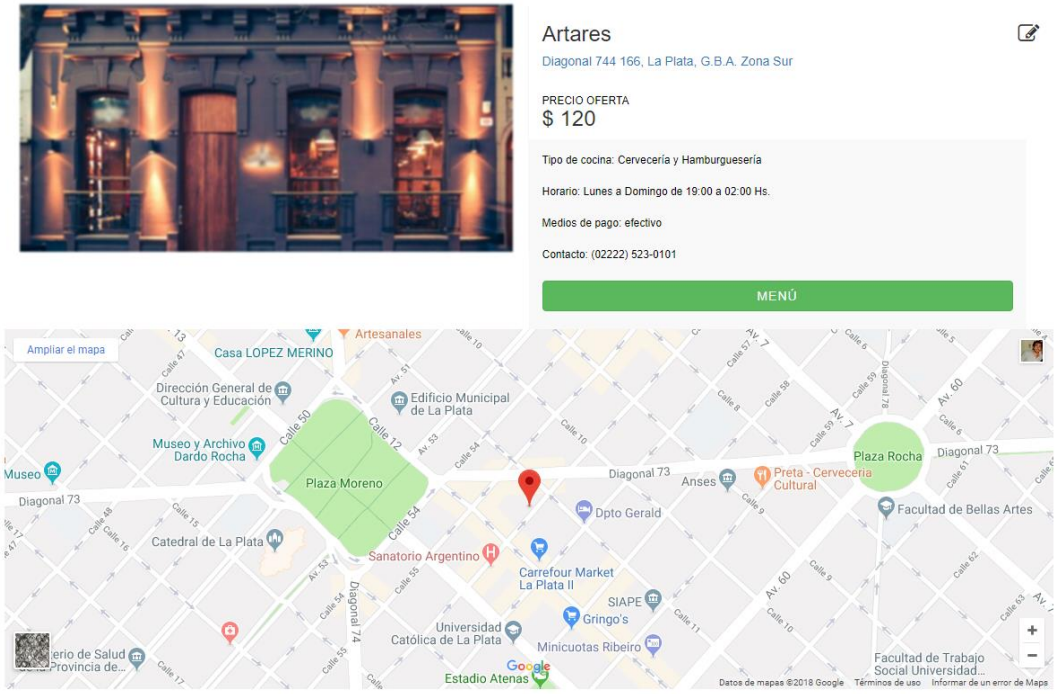


Figura 4.3.15: Características del Sitio – Visualización y Edición

Continuando con las configuraciones del sitio gastronómico, el dueño podrá establecer que servicios brindará su sitio, como así también una descripción del lugar para que luego estos datos estén disponibles para los clientes. También, aquí podrá visualizar los comentarios que los clientes fueron dejando sobre su sitio y servicios brindados. En la Figura 4.3.16 se muestra lo descripto anteriormente.

Figura 4.3.16: Características del Sitio – Visualización y Edición (continuación)

- ***Pedidos***

  Al seleccionar la opción *Pedidos* (de la Figura 4.3.14), se le da la posibilidad al dueño de acceder a los pedidos que tiene su sitio gastronómico, visualizando cada uno de ellos con sus principales características. El dueño también podrá cambiar el estado de los pedidos, como puede ser el caso de que un pedido pase de estar "*pendiente"* a "*listo para retirar"*, y ver más detalles sobre cada uno de ellos[28].

  En la Figura 4.3.17 se puede observar los pedidos que tiene el dueño *Marcos Gutiérrez* para su sitio *Artares*:

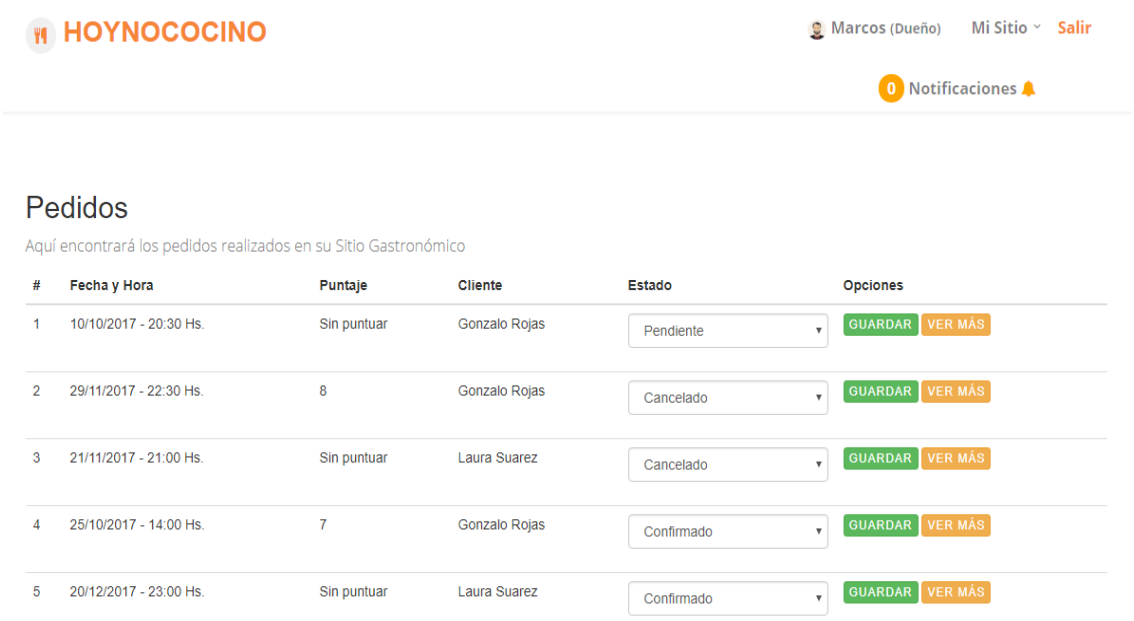


Figura 4.3.17: Pedidos - Sitio *Artares*

- ***Servicios***

  Al seleccionar la opción *Servicios* (de la Figura 4.3.14), se le muestra al usuario una pantalla similar a la mostrada en la Figura 4.3.16 Donde puede ver y editar dicha información.

- ***Promociones***

  Al seleccionar la opción *Promociones* (de la Figura 4.3.14), el dueño del sitio puede visualizar todas las promociones del sitio (caducadas y vigentes), agregar nuevas[29], modificarlas y desactivarlas, como se puede apreciar en las Figuras 4.3.18.

---

[28] La funcionalidad del botón "*Ver más*" no está implementada porque excede del alcance de lo que se quiere mostrar en la tesina, al seleccionar dicha opción el usuario recibe el cartel "*Funcionalidad no desarrollada para el prototipo*" como se presentó en la Figura 4.3.4.

[29] La funcionalidad de los botones "*Ver más*" y "*Nueva Promoción*" no está implementada porque excede del alcance de lo que se quiere mostrar en la tesina, al seleccionar dicha opción el usuario recibe el cartel "*Funcionalidad no desarrollada para el prototipo*" como se presentó en la Figura 4.3.4.

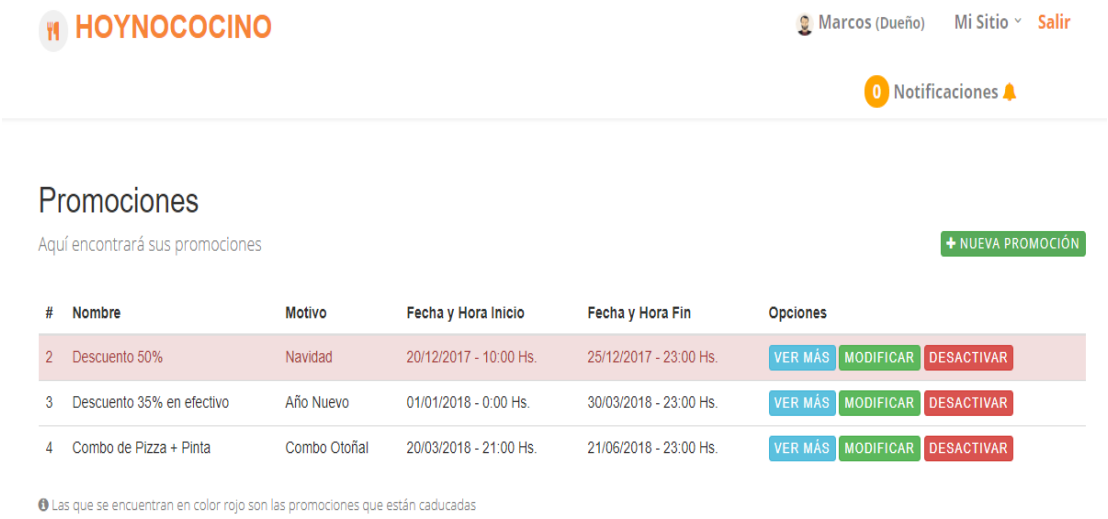


Figura 4.3.18: Listado de promociones

Como se pudo observar en la Figura 4.3.18, los datos de las promociones que se visualizan son: e*l número de promoción, el nombre y motivo de la misma, fecha y hora* (desde su comienzo hasta su final) *y las opciones disponibles*. En dicho listado solo se encuentran las promociones activadas, por lo tanto si el usuario desactiva alguna de ellas con el botón rojo "*Desactivar*", esa promoción no se visualizará más.

Por otro lado, las promociones que se encuentran pintadas de color rojo significan que están vencidas. Al seleccionar la opción para "*Modificar*" una promoción, se visualizara el formulario para editar los datos de la promoción seleccionada, como se observa en la Figura 4.3.19.

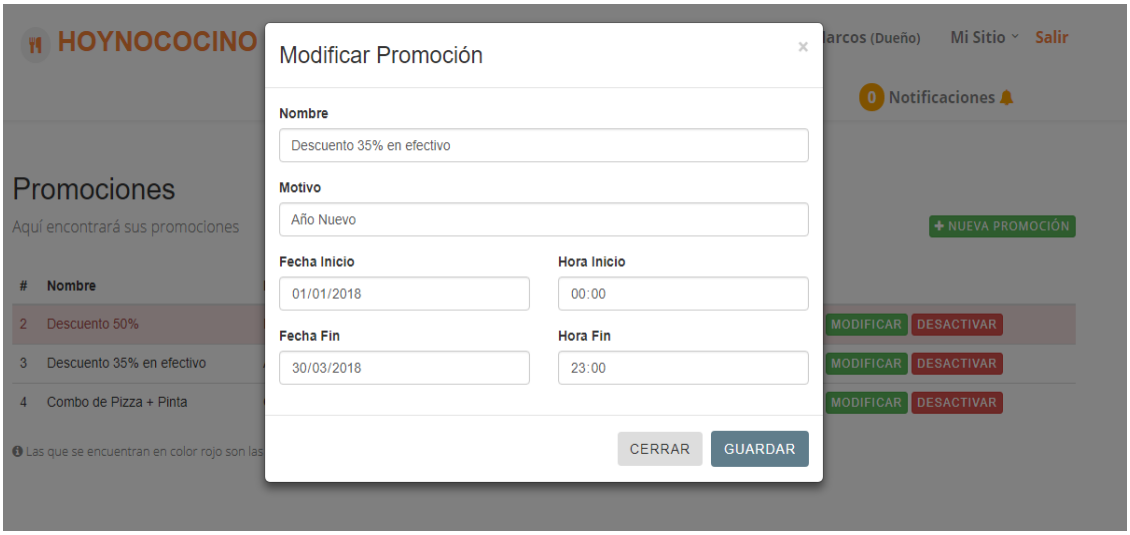


Figura 4.3.19: Modificar Promoción

- ***Salir***

  Al seleccionar la opción *Salir* (de la Figura 4.3.14), se vuelve al inicio del prototipo visualizando en pantalla la Figura 4.3.1.

- **Notificaciones**

Una vez presentada las secciones correspondientes al menú que posee tanto el usuario con perfil de *cliente* como de *dueño*, se explicará a continuación cómo y dónde se visualizan las notificaciones en el prototipo para ambos perfiles.

Como se observa en la Figura 4.3.20 las notificaciones poseen distintas alternativas para visualizarlas.

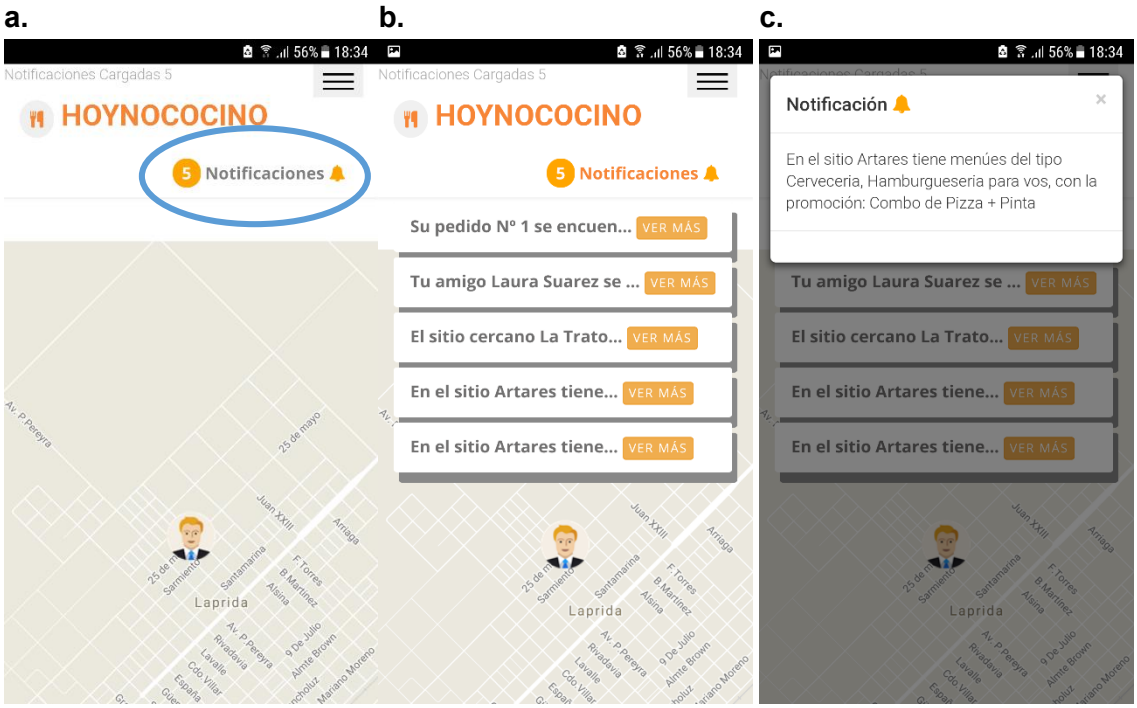


Figura 4.3.20: Notificaciones

Veamos en detalle cada una de las opciones mostradas en la Figura 4.3.20:

a) Se presenta la visualización por defecto, la cual avisa al usuario la cantidad de notificaciones que posee en un determinado momento. Si el usuario hace *click* sobre el botón "*Notificaciones*", se muestra lo que se presenta en la Figura 4.3.20b.

b) Se listan todas las notificaciones que posee el usuario con un indicio de lo que el prototipo quiere notificar, y con la posibilidad de ver más, lo cual permite ampliar la información de la notificación recibida.

c) Si el usuario hace *click* en el botón "*Ver más*" de la Figura 4.3.20b., el prototipo mostrará un cartel con la información completa de dicha notificación. Cuando el usuario abre el cartel de la notificación, el prototipo marcará dicha notificación como que fue vista por el usuario.

- **Posicionamiento**

Como se mencionó a lo largo de ésta sección, el usuario con perfil *cliente* posee en su menú (Figura 4.3.3) la opción para visualizar un mapa (Figura 4.3.7) en donde se representa a través de marcadores al usuario logueado, a sus amigos y sitios

gastronómicos cercanos. Dichos marcadores son ubicados en el mapa de acuerdo a la posición que tenga cada uno de ellos. Para que el *cliente* pueda visualizar lo descripto, se requiera que el prototipo pueda acceder a la posición actual del usuario. Para esto, al ingresar como *cliente*, el prototipo le solicita al mismo la autorización para poder acceder a su posición actual. En la Figura 4.3.21 se presenta como se realiza dicha solicitud.

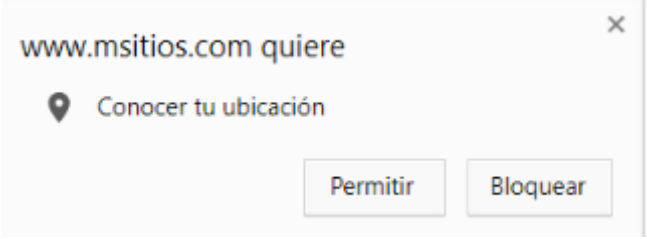


Figura 4.3.21: Solicitud Permiso Ubicación

En caso de que el usuario niegue dicho acceso presionando en el botón "*Bloquear*", o permita la solicitud pero tiene el GPS deshabilitado en el dispositivo, el prototipo mostrará en pantalla un cartel con el aviso "*GPS deshabilitado, no se puede acceder a la geolocalización*". Dicho aviso, se puede observar en la Figura 4.3.22.

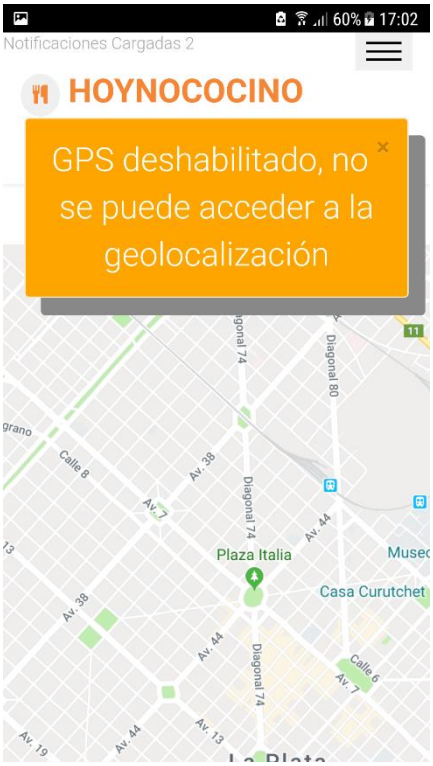


Figura 4.3.22: Aviso GPS deshabilitado

- **Pie de Página**

Para finalizar, se encuentra la sección de *pie de página*, donde se pueden visualizar los accesos directos a las redes sociales que podría tener el prototipo como son *Twitter*, *Facebook*, *Instragram*, canal de *YouTube*. Dicha sección, está disponible tanto para el perfil *cliente* (final de la Figura 4.3.2) como para el *dueño* (final de la Figura 4.3.13).

Debido a que esto escapa a lo que se quiere mostrar en este trabajo, no existen conexión con dichas redes sociales desde el prototipo. En la Figura 4.3.23 se puede observar estos accesos.

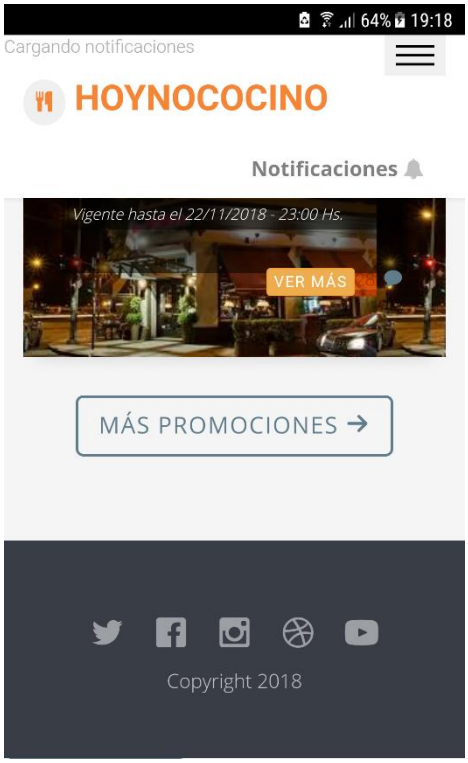


Figura 4.3.23: Pie de página

4.4 Dificultades a la hora de desarrollar el prototipo

En ésta sección se detallan las dificultades más relevantes que fueron surgiendo a lo largo del análisis, desarrollo e implementación del prototipo.

- **Representar los modelos presentados en la base de datos**

  A la hora de generar la base de datos, fue complejo representar todo el comportamiento de los modelos presentados en la Figura 3.4.1 (Modelo de Contexto) y 3.4.5 (Modelo de Dominio) de la Sección 3.4 en una base de datos relacional. En algunas situaciones, no se presentaban serias dificultades pero el resultado de lo descripto era una base de datos compleja y de gran tamaño que no era necesaria para implementar el prototipo. Para disminuir dicha complejidad, se simplificaron algunas clases de los modelos para obtener una base de datos más acorde a lo que el prototipo demandaba.

    - Del *Modelo de Contexto* presentado en la Figura 3.4.1 la principal simplificación que se realizó fue dejar un único *ambiente*, y no disponer de *subambientes* debido a que los servicios contextuales del prototipo se dan en un único ambiente (donde se encuentran los consumos que cambian de estado y las promociones que se notifican). Es decir, en vez de tener un *subambiente* para el único sitio que posee los valores que son utilizados para brindar los servicios contextuales, se deja un solo *ambiente* general.

- Del *Modelo de Dominio* presentado en la Figura 3.4.5 se simplificaron en la base de datos las clases *Posicion* y *Rol* convirtiéndolas en atributos de la clase *Persona*. Lo mismo con las clases *Tipo_Comida* y *Menu* para que sean atributos de la clase *Sitio_Gastronomico*.

- **Complejidad en el seguimiento de la ejecución de los servicios**

  A lo largo del desarrollo del prototipo, fue complejo el seguimiento del código y testeo de las funcionalidades de los distintos servicios. El modelo cuenta con muchos niveles de abstracción y delegación, que si bien representan un diseño desacoplado, generan complejidad en el seguimiento de la propagación de cambios contextuales, sobre todo a la hora de detectar funcionamientos inadecuados en los servicios contextuales.

- **Error en la captación de la geolocalización en el servidor de producción**

  A la hora de poner en producción el prototipo en el servidor, no funcionaba la geolocalización, es decir, dicho prototipo no podía obtener la posición actual del GPS y por lo tanto, no visualizaba los marcadores en el mapa. Dicho problema surgía debido a que el servidor no poseía *Certificado SSL (Secure Socket Layer)* para poder utilizar el protocolo *HTTPS* que es requerido por *Google Maps API*[30], indicando lo siguiente:

  > "*Se requiere HTTPS para todas las solicitudes de servicios web de Maps API que contengan datos del usuario o identificadores de desarrollador. Las solicitudes realizadas a través de HTTP que incluyan datos confidenciales se rechazarán."*

  Este problema se resolvió generando un *Certificado SSL*[31] a través del portal que provee el servicio de hosting (*GoDaddy*[32]) donde se subió a producción el prototipo. Hecho esto, se le permitió al prototipo poder captar la geolocalización en el servidor web y así obtener el correcto funcionamiento del mismo.

- **Problemas con las rutas en producción**

  A la hora de poner en producción el prototipo en el servidor, algunos archivos que se incluían en el proyecto con rutas que comenzaban con "*/*" no eran encontradas, y por lo tanto no se podían cargar dichos archivos. Esto no pasaba en la etapa de desarrollo con *WampServer* 3.0.6 [WampServer, 2004] ya que las rutas funcionaban correctamente.

  Dicho inconveniente se logró descubrir revisando los *logs* del servidor en producción y posteriormente se resolvió, removiendo del inicio de las rutas el carácter que generaba el error ("*/*").

---

[30] Página de *Google Maps JavaScript API: https://developers.google.com/maps/web-services/overview?hl=es-419* (Último acceso 18-02-2018)

[31] Página de *GoDaddy:* https://ar.godaddy.com/help/que-es-un-certificado-ssl-542 (Último acceso 18-02-2018)

[32] Página de *GoDaddy:* https://ar.godaddy.com/help/solicitar-un-certificado-ssl-562 (Último acceso 18-02-2018)

- **El retorno en las funciones del lado del cliente generaba demora excesiva o producía que la ejecución de las mismas retornen valores incorrectos**

  Al momento de generar el mapa y cargar los *marcadores* en el mismo, por ejemplo al usuario principal con sus amigos y sitios cercanos, se producía una demora excesiva en las respuestas de las funciones que lo realizaban, y por ende, en la carga del mapa del prototipo web. En otros casos, el mapa no podía ser inicializado porque los retornos de las funciones eran incorrectos (*undefined*). Es decir, el error se daba porque las funciones que inicializaban y creaban el mapa no retornaban el valor correcto en un determinado momento, porque las funciones anidadas no terminaban la ejecución antes del retorno. Esto se debe a que *Javascript* [Javascript, 1995] no soporta *multi-threading* [33] [34].

  La solución fue ejecutar dichas funciones de manera asincrónica [35] [36], también alterar el orden de los llamados de las funciones, logrando así, obtener los valores de retornos correctamente.

- **El retorno del lado del servidor en la propagación de los cambios contextuales (resultado de una petición) generaba demora excesiva y producía que la ejecución de los servicios sea demasiada lenta**

  Cuando se daba un cambio de contexto, el retorno en la propagación de dicho cambio era considerablemente lento. Es decir, el retorno desde que se producía el cambio de un valor observado por una *ContextFeature*, hasta que ésta propagaba y delegaba dicho cambio a las demás clases del modelo pasando todas las validaciones correspondientes para que finalmente se ejecutara el servicio. Entonces, por ejemplo, en los servicios que hay que notificar cambios contextuales al cliente, perdían efectividad al producirse lo mencionado.

  La solución escogida fue, que al momento de ejecutar un servicio que debería notificar alguna información al cliente, el servicio inserte dicha notificación en una tabla de la base de datos llamada "*notificacion*", y luego, a través de una función de *Javascript* [Javascript, 1995] se consulta cada 5 segundos si hay alguna notificación nueva para mostrarle al usuario. Dicha tabla, posee un funcionamiento similar al de una memoria caché, donde se va almacenando datos temporales (*notificaciones*). Cabe a aclarar que, para que un servicio agregue una notificación en la tabla, se debe propagar el cambio y pasar las validaciones tal cual se describe en los diagramas presentados en las Figura 3.5.4, Figura 3.5.5 y Figura 3.5.6.

---

[33] Página de *StackOverflow*: https://stackoverflow.com/questions/39879/why-doesnt-javascript-support-multithreading (Último acceso 18-02-2018)

[34] Página de *Sitepoint*: https://www.sitepoint.com/multi-threading-javascript/ (Último acceso 18-02-2018)

[35] Página de *Google*: https://developers.google.com/web/fundamentals/primers/async-functions?hl=es

[36] Página de *Mozilla:* https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/funcion_asincrona (Último acceso 18-02-2018)

En el caso del servicio que visualiza los sitios cercanos a la posición actual del usuario, tiene un funcionamiento similar a lo anteriormente descripto, pero el *observer* de la posición, está dado por la función de *Javascript watchPosition()*[37] de la librería de *Geolocation API*.

- **Superposición de marcadores en el mapa entre sitios y amigos**

  Cuando se daba el contexto en el que más de un objeto tenía la misma posición, por ejemplo cuando un amigo se encontraba en un sitio gastronómico, se superponían los marcadores en el mapa dificultando la visualización de los mismos. Esto se resolvió agregando animaciones[38] diferentes a los marcadores (usuarios amigos), y así poder visualizarlos correctamente en el mapa. Más específicamente, a los marcadores de los amigos se les agrego la animación para "*rebotar*" (*bounce*) sobre el lugar en donde se encuentran, de esta manera, queda más claro cuando un amigo se encuentra en un sitio, ya que simula estar saltando sobre el sitio gastronómico.

  Se aclara que dicha solución es para el prototipo, porque en el caso de haber *N* amigos y *N* sitios en una única posición, no se podría identificar y ver con claridad los marcadores en el mapa.

- **Notificaciones duplicadas**

  Como se mencionó anteriormente, en el momento que un servicio contextual se ejecuta y requiere notificar al usuario, agrega dicha notificación en la tabla "*notificacion"* (caché). Por ejemplo, en el servicio donde se notificar si un amigo se encuentra en un sitio gastronómico cercano, pasaba que el GPS captaba una nueva posición (cambio de contexto) cuando en realidad se diferenciaba por decimales con respecto a la posición actual y no representaba un cambio de posición concreto y considerable. Dicho cambio de contexto producía la ejecución del servicio y por ende la inserción de la notificación correspondiente. Cuando se producían estos cambios de contextos insignificantes se insertaba más de una vez la misma notificación y el usuario tenía como resultado notificaciones repetidas.

  La solución que se llevó a cabo fue verificar antes de insertar una notificación: si la misma no había sido vista por el usuario, si no había otra con el mismo contenido para ese usuario y para ese mismo servicio contextual que se ejecutó.

- **Recarga innecesaria de marcadores en el mapa al cambiar la posición**

  Como se mencionó anteriormente muchas veces el GSP captaba una nueva posición (*cambio de contexto*) cuando en realidad no representaba un cambio de posición

---

[37] Página de *Mozilla*: https://developer.mozilla.org/es/docs/Web/API/Geolocation/watchPosition (Último acceso 18-02-2018)

[38] Página de *Google*: https://developers.google.com/maps/documentation/javascript/examples/marker-animations?hl=es-419 (Último acceso 18-02-2018)

significativamente considerable. Esto producía que haya que actualizar los marcadores en el mapa innecesariamente.

Este problema se solucionó evaluando antes de actualizar los marcadores en el mapa si el cambio de posición era notable, considerando un cambio de posición cuando se diferenciaba al menos de cinco metros aproximadamente de la posición actual. Este problema es habitual en aplicaciones sensibles al contexto [Fortier et al., 2010], donde muchas veces se tiene en cuanta cuanto varió el valor de contexto, para propagar o no el cambio.

5. Ejemplo uso del prototipo

En este capítulo se describe el funcionamiento de los servicios contextuales mencionados en la Sección 3.1, estos son:

- *Visualizar sitios cercanos*
- *Notificar que un amigo se encuentra en algún sitio cercano a su posición actual*
- *Notificar promociones de acuerdo a las preferencias gastronómicas del usuario y posición actual*
- *Notificar al cliente que su pedido se encuentra listo*
- *Notificar al dueño que un pedido fue cancelado por el cliente*

Para la demostración de los servicios se partirá de un escenario inicial, como se mencionó en la Sección 4.2.

Para las demostraciones de los servicios, la posición inicial del usuario *Gonzalo Rojas* (*cliente*) al iniciar será la siguiente: latitud -34.9290595 y longitud -57.927044, mencionada de aquí en más como el *origen*. En la Figura 5.1 se puede observar donde se encuentra el usuario con las coordenadas mencionadas mostrando dicho posicionamiento con *GoogleMaps* [GoogleMaps, 2005]. Esta posición se usara como posición inicial de todos los servicios que se muestran en este capítulo.

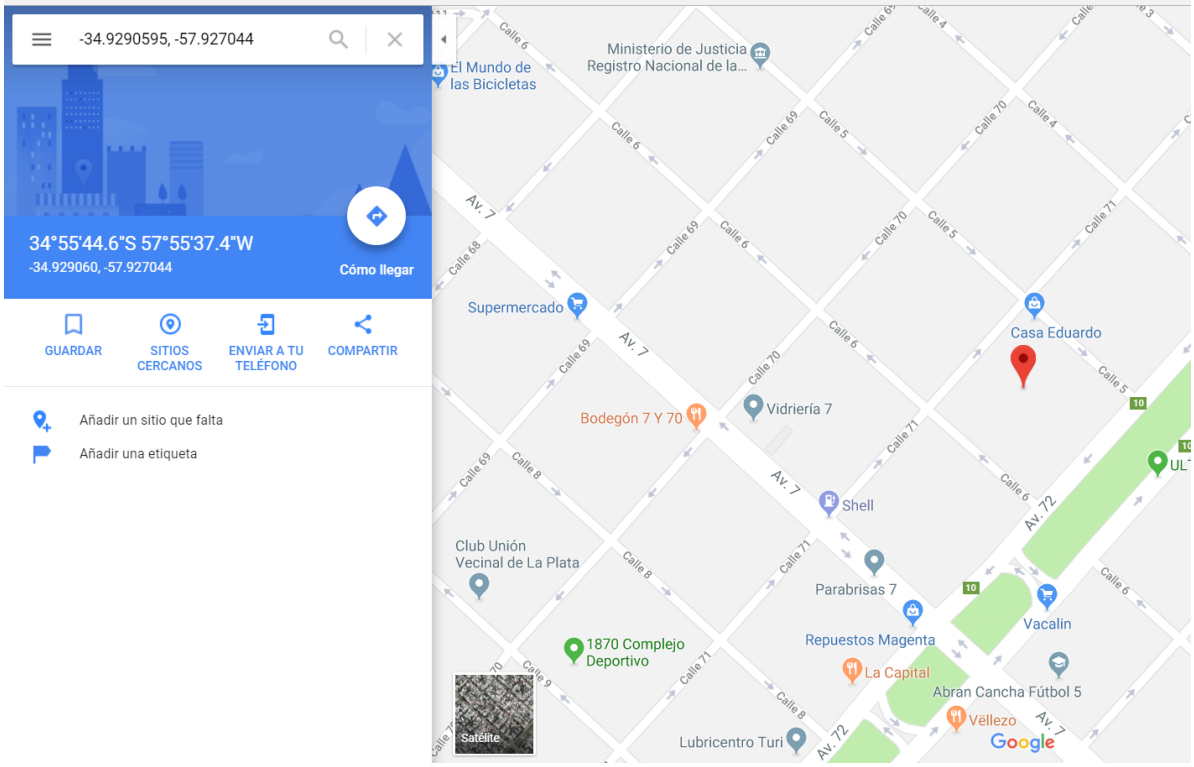


Figura 5.1: Posición inicial del usuario

- ***Servicio Visualizar Sitios Cercanos***

  Al ingresar al prototipo, el usuario Gonzalo con perfil *cliente*, accede al mapa que muestra los sitios gastronómicos cercanos a la posición actual del mismo. Como se mencionó en el capítulo anterior, se consideró cercano a un radio menor a 1 kilómetro entre el usuario y los sitios.

  Recordando, los sitios gastronómicos precargados en el prototipo como se presentaron en la Sección 4.2, poseen las siguientes posiciones:

  - *Artares* con Posición: latitud -34.9016 y longitud -57.9488.
  - *Chichara* con Posición: latitud -34.9114 y longitud -57.9888.
  - *La Tratopia* con Posición: latitud -34.921007 y longitud -57.952857.
  - *Pleta* con Posición: latitud -34.7466 y longitud -57.6555.
  - *La Candombersa* con Posición: latitud -34.926968 y longitud -57.954404.
  - *Baviesa* con Posición: latitud -34.926334 y longitud -57.961786.
  - *Ocho Sabio* con Posición: latitud -34.926369 y longitud -57.952688.
  - *Blothers* con Posición: latitud -34.923273 y longitud -57.955048.
  - *Runieses* con Posición: latitud -34.904129 y longitud -57.937624.
  - *Grac* con Posición: latitud -34.918558 y longitud -57.947538.

  En este caso, tal como informa el aviso en el prototipo que se muestra en la Figura 5.1, el usuario *Gonzalo Rojas* no posee sitios cercanos su posición de origen, que como se mencionó, la misma está conformada por una latitud de -34.9290595 y una longitud de -57.927044.

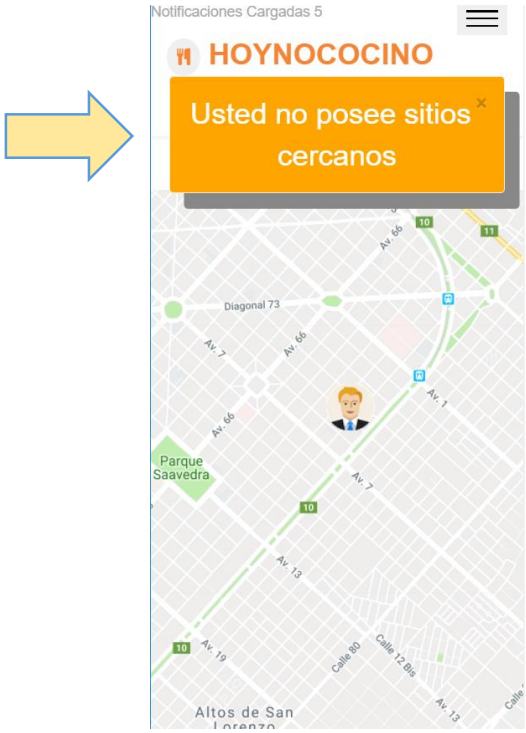


Figura 5.1: Posición inicial del usuario en el prototipo

Si se calcula la distancia a través de *GoogleMaps* entre la posición origen (-34.9290595, -57.927044) de *Gonzalo Rojas* y por ejemplo la del sitio *La Tratopia* (-34.921007,-57.952857), se puede verificar que entre ellos hay una distancia de 2.52 kilómetros, siendo superior a la distancia de 1 kilómetro que se considera como cercana. En la Figura 5.2 se puede demostrar lo expuesto.

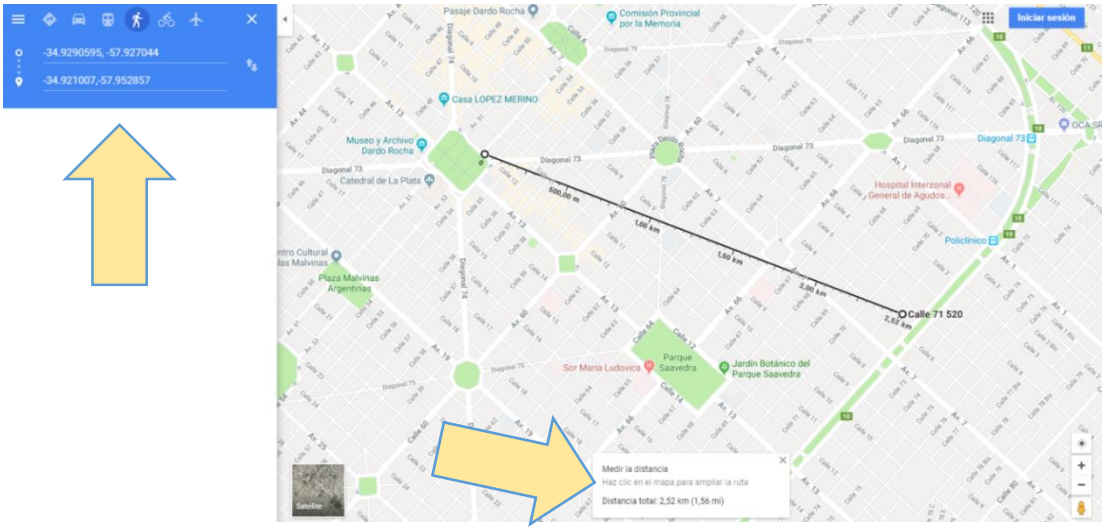


Figura 5.2: Distancia entre la posición de Gonzalo Rojas (-34.9290595, -57.927044) y el sitio *La Tratopia* (-34.921007, -57.952857)

A partir del escenario presentado anteriormente, supongamos que el usuario cambia la posición actual a las siguientes coordenadas: latitud -34.9252207 y longitud -57.945648. Esta nueva posición se puede visualizar en la Figura 5.3. Este desplazamiento también será usado en los siguientes servicios que se muestran en este capítulo para poder mostrar la movilidad del usuario.

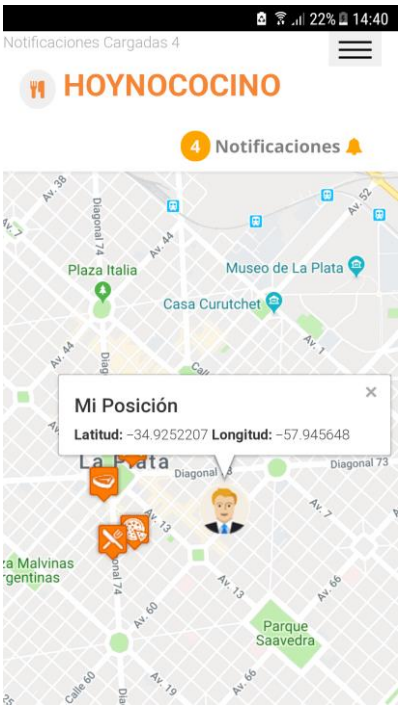


Figura 5.3: Posición luego del desplazamiento

Dado este cambio de posición (*contexto*), el prototipo lo detecta y comprueba si hay sitios cercanos a la nueva posición del usuario. Para recordar cómo funciona y se propaga el cambio de contexto hasta que se produce la ejecución del servicio *Visualizar Sitios Cercanos*, se recuerda lo mostrado en el *diagrama de secuencia* de la Figura 3.5.3, dónde se identifica que el valor a observar es la posición del usuario, y que al cambiar, provoca una reacción en el prototipo interactuando entre las distintas clases, para así brindar el servicio. Como resultado de dicha comprobación, se obtiene que los sitios *Ocho Sabió*, *La Candombersa*, *Grac*, *La Tratopia* y *Blothers* están cercanos a la nueva posición de *Gonzalo*, y como consecuencia se activa el servicio *Visualizar Sitios Cercanos*, el cual muestra estos sitios como se puede apreciar en las Figuras 5.4 y Figura 5.5. En dichas figuras se muestra la información de cada uno de estos lugares.

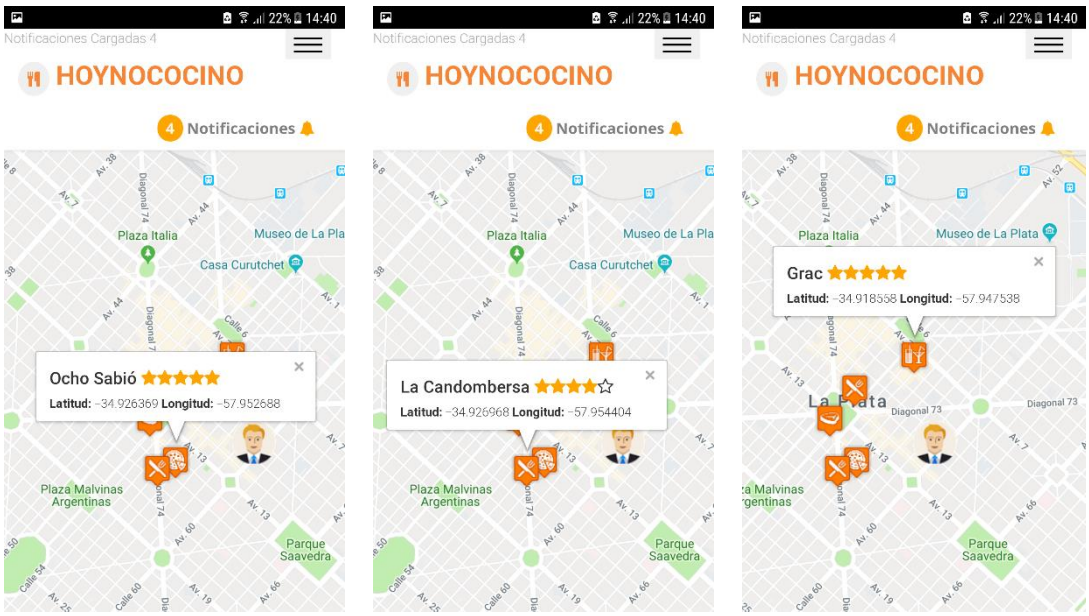


Figura 5.4: Sitios cercanos

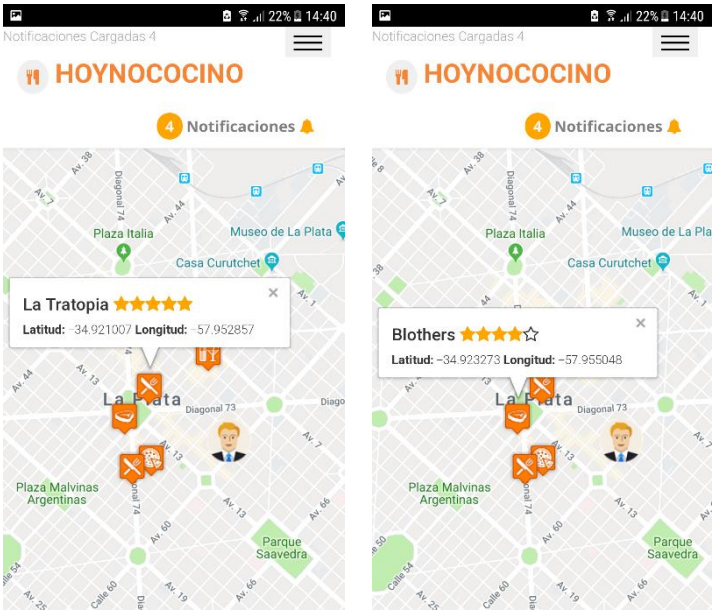


Figura 5.5: Sitios cercanos (continuación)

De esta manera, se pudo apreciar como a medida que el usuario va cambiando de posición los sitios cercamos varían.

Cabe destacar que el funcionamiento interno de este servicio se puede apreciar en el diagrama de secuencia de la Sección 3.5 - Figura 3.5.3.

- **Servicio notificar presencia de amigos en sitio cercano a su posición actual**

En este servicio se le notifica al usuario *cliente* si alguno de sus amigos se encuentra en un sitio cercano a su posición actual. Para profundizar sobre este servicio, se toma de ejemplo nuevamente al usuario *Gonzalo Rojas*, recordando que el mismo posee dos amigos. Como se mencionó anteriormente, dichos amigos están precargados en el prototipo en su estado inicial y se pueden ver dentro del perfil del cliente como se muestra la Figura 5.6.

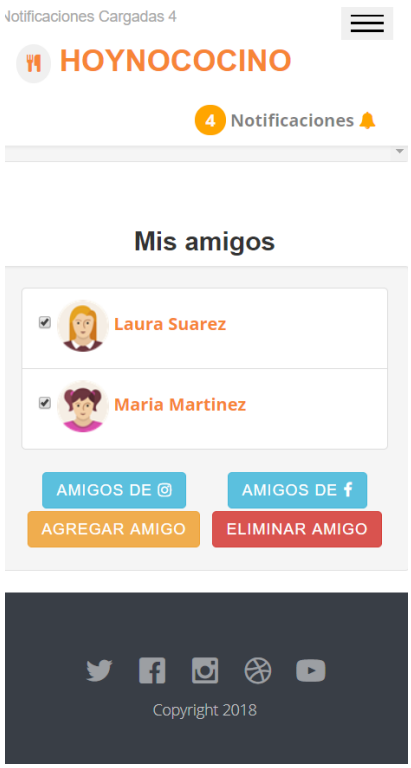


Figura 5.6: Amigos del usuario Gonzalo

Supongamos que estos amigos mostrados en la Figura 5.6, están en las siguientes posiciones:

- *Laura Suarez*: latitud -34.921007 y longitud -57.952857.
- *María Martínez*: latitud -34.9219055196 y longitud -57.960154.

Supongamos que volvemos a posicionar a *Gonzalo* Rojas en la posición mostrada en la Figura 5.1. Es decir, no hay sitios cercanos, con lo cual tampoco se muestran amigos en sitios cercanos. Luego, el usuario se desplaza a la posición mostrada en la Figura 5.3. Esto provoca un cambio de contexto detectado por el prototipo. Ante este cambio de

posición, al igual que en el servicio anterior (*sitios cercanos*) el prototipo comprueba si hay sitios cercanos a la nueva posición del usuario (como se mostró en las Figuras 5.4 y 5.5), pero a su vez, en este servicio se comprueba si algunos de sus amigos se encuentran en dichos sitios. Acorde a la posición actual de *Laura Suarez* (-34.921007, -57.952857) está en *La Tratopia*, por lo que provoca la ejecución del servicio *Notificar presencia de amigos en sitio cercano*. La ejecución de dicho servicio, provoca que se le notifique a Gonzalo lo detectado como se puede observar en la Figura 5.7.

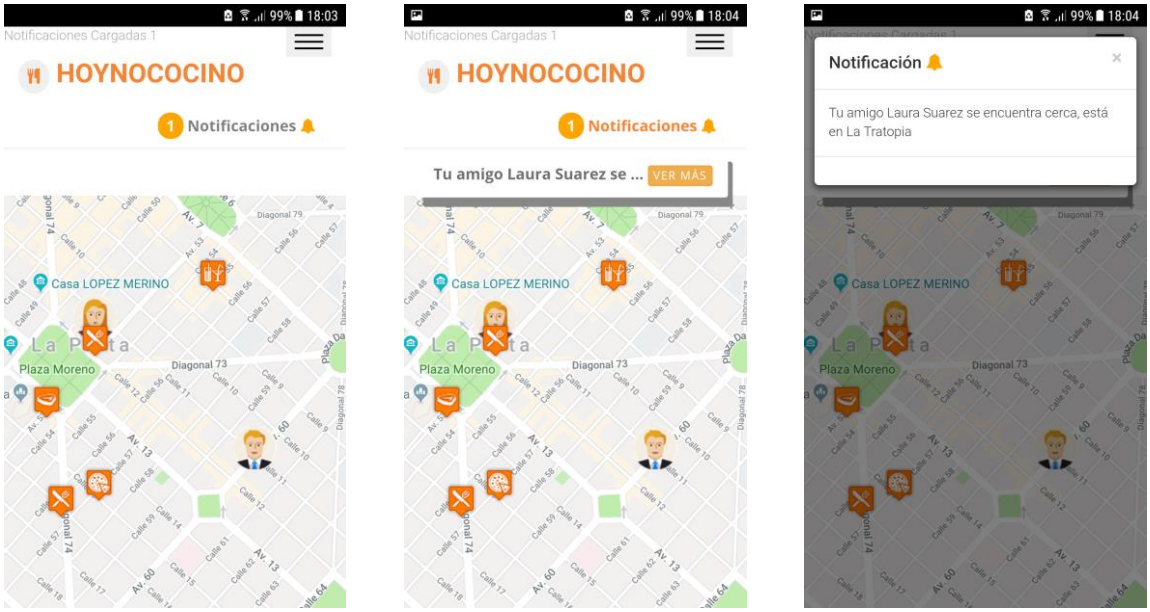


Figura 5.7: Notificación amigo en sitio cercano

Cabe mencionar que para el funcionamiento de este servicio, se toma como contextos: la posición actual de Gonzalo, a su lista actual de amigos y sus respectivas posiciones como así también los sitios cercanos. A medida que el usuario se va moviendo y se van actualizando los sitios cercanos, la visualización de sus amigos en estos sitios, también se actualiza.

▪ Servicio notificar promociones

En este servicio se le notificará al cliente sobre las promociones que ofrecen los diferentes sitios, según sus gustos y su posición actual. Cabe recordar que las promociones ya precargadas en el prototipo se presentaron en la Sección 4.2: A modo de resumen y para facilitar al lector se vuelven a listar simplificadamente las promociones vigentes.

- Promoción #1, pertenece al sitio *La Tratopia*, con estado de vigente y nombre "*Promoción 2X1*".
- Promoción #2, pertenece al sitio *Artares*, con estado de no vigente y nombre "*Descuento del 50%*".
- Promoción #3, pertenece al sitio *Artares*, con estado de vigente y nombre "*Descuento del 35% en efectivo*".
- Promoción #4, pertenece al sitio *Artares*, con estado de vigente y nombre "*Combo de Pizza + Pinta*".

Las notificaciones pueden darse porque están relacionadas al contexto de las preferencias del usuario o porque están cercanas a su posición actual. Es decir, estas notificaciones se activan acorde a estos dos criterios. A continuación se describe cómo funcionan cada uno de estos.

- **Notificación promociones por posición**

  Supongamos que volvemos a posicionar a *Gonzalo* Rojas en la posición mostrada en la Figura 5.1. Es decir, no hay sitios cercanos, con lo cual tampoco se muestran amigos en sitios cercanos. Luego, el usuario se desplaza a la posición mostrada en la Figura 5.3. Esto provoca un cambio de contexto detectado por el prototipo.

  Ante este cambio de posición, el prototipo verifica si existen sitios cercanos a la posición nueva del usuario. Al igual que en el servicio "*Visualizar sitios Cercanos*", se obtienen los sitios: *Ocho Sabió*, *La Candombersa*, *Grac*, La *Tratopia* y *Blothers*.

  Luego, si algunos de estos sitios cercanos tienen promociones vigentes, se notifican de las mismas al usuario. Dado que el sitio "*La Tratopia*" tiene la promoción (#1) vigente, se le notifica de la misma al usuario como se presenta en la Figura 5.8.

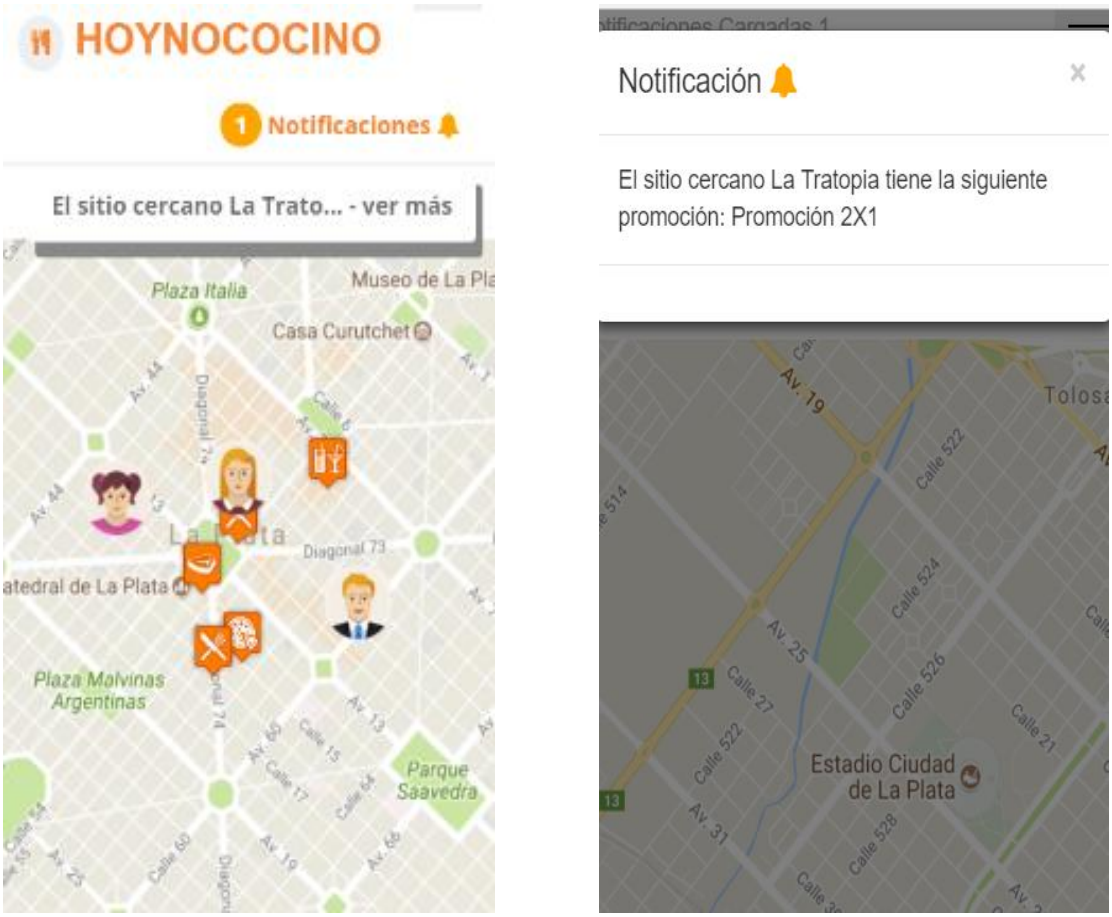


Figura 5.8: Notificación promociones por posición

- **Notificación promociones por preferencias gastronómicas**

Como se puede observar en la Figura 5.9.a*, Gonzalo Rojas* tiene en su estado inicial como preferencias gastronómicas a "*parrilla*", supongamos que añade una nueva preferencia: "*hamburguesería*" como se muestra en la Figura 5.9.b.

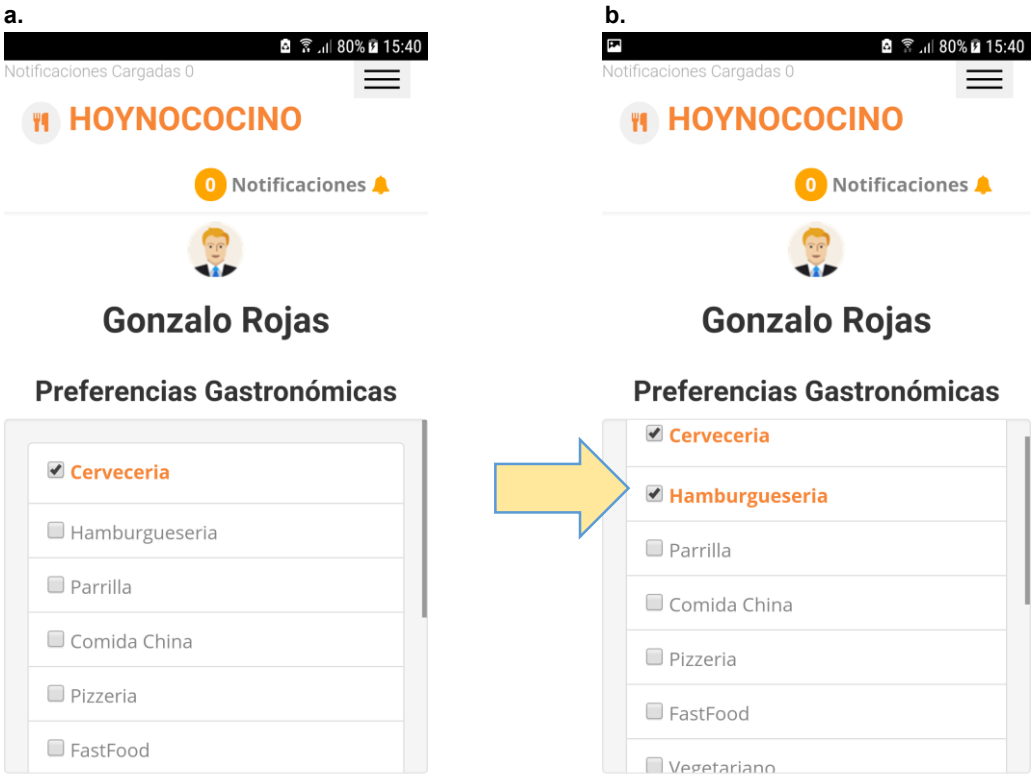


Figura 5.9: Cambio de preferencias gastronómicas

El prototipo detecta este cambio de preferencia gastronómica (contexto), y verifica si los sitios gastronómicos que tienen como tipo de comida la nueva preferencia del cliente (*hamburguesería*) poseen promociones vigentes. En el caso que se cumplan las condiciones mencionadas, se le notifica al cliente de las promociones correspondientes.

Acorde a este cambio en el contexto de las preferencias, se identifica que el sitio *Artares* que tiene como tipo de comida a: *Cervecería* y *Hamburguesería* (como se detalló en la Sesión 4.2), coincidiendo con la nueva preferencia añadida por el usuario en la Figura 5.9.b. Además, este sitio tiene dos promociones vigentes. Es decir, como resultado del cambio de contexto de preferencias, se le notifica al cliente de las promociones correspondientes como se puede observar en la Figura 5.10.

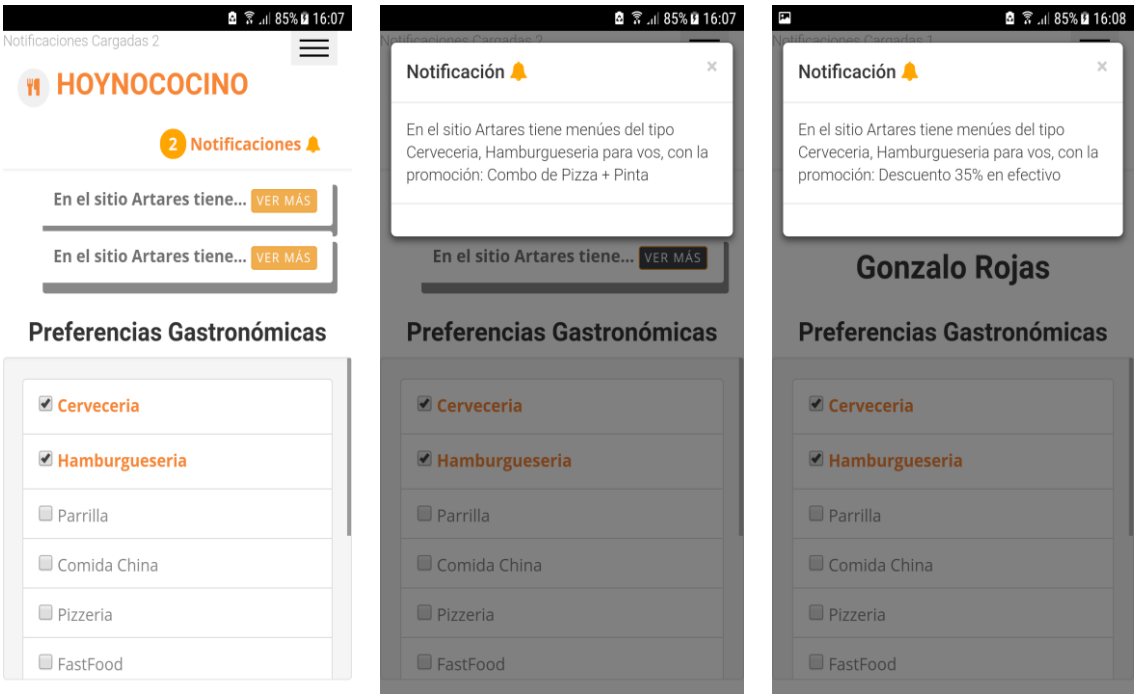


Figura 5.10: Notificación promoción por preferencia gastronómica

- ## Servicio notificar al cliente que su pedido se encuentra listo para ser retirado

Para describir este servicio y su funcionamiento en el prototipo, se toman como referencias los consumos que posee el sitio *Artares*. En la Figura 5.11 se puede observar la lista de consumos que puede visualizar el dueño de dicho sitio (*Marcos Gutiérrez*). Este listado guarda relación con los consumos presentados en la Sección 4.2.

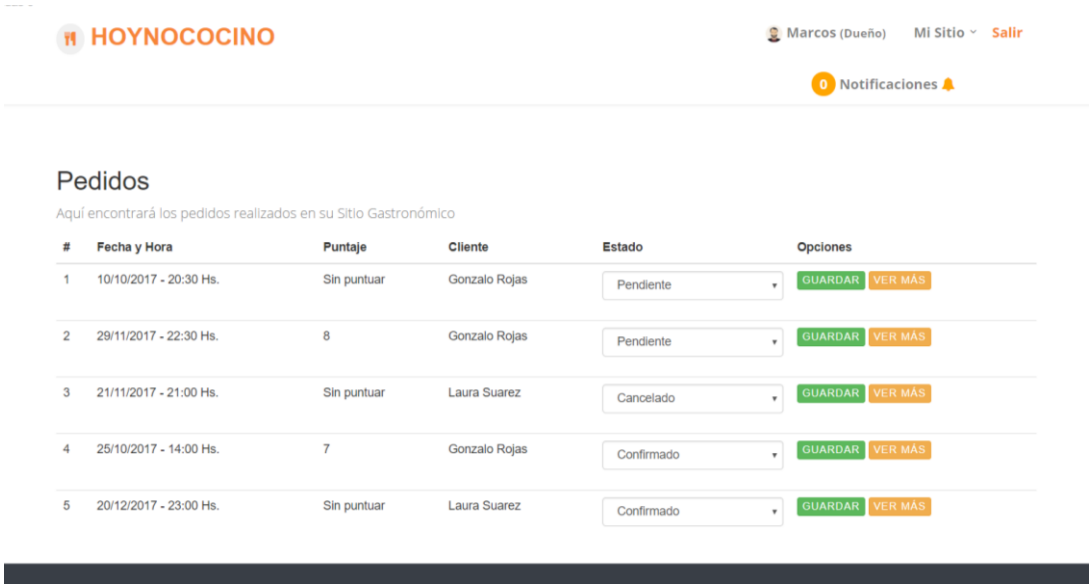


Figura 5.11: Listado de Pedidos del sitio Artares

Como se mencionó en la Sección 4.3 (Figura 4.3.18) el dueño del sitio gastronómico puede cambiar los estados de sus pedidos. Supongamos que el dueño cambia el estado de algunos de sus pedidos a "*Listo*". En la Figura 5.12 se puede observar cómo se puede cambiar el pedido #1 de "*Pendiente*" a "*Listo*".

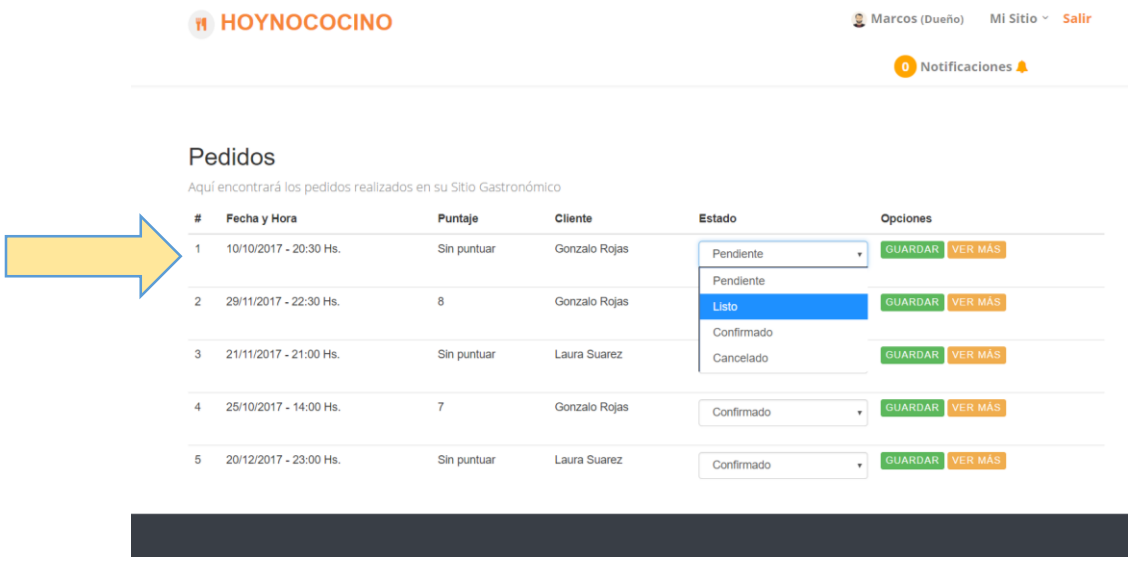


Figura 5.12: Cambio de estado de un pedido a *Listo*

Al producirse este cambio de contexto (*estado del consumo*) en el pedido #1, se notifica dicho cambio al cliente (*Gonzalo Rojas*) cómo se muestra en la Figura 5.13.

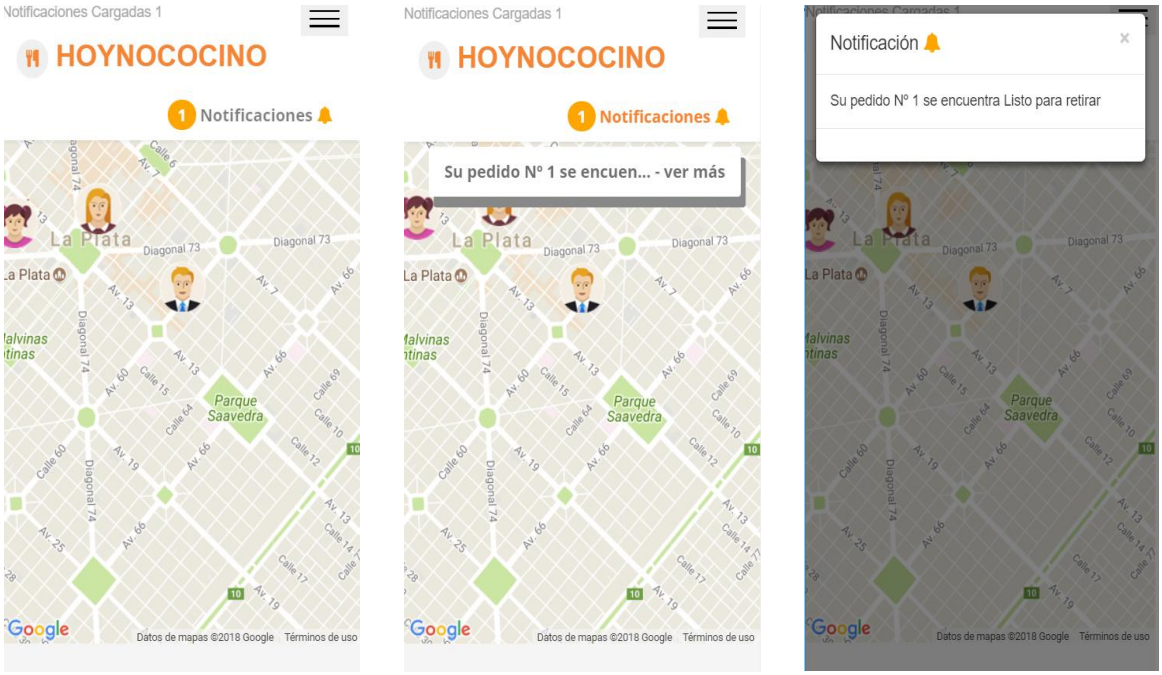


Figura 5.13: Notificación al cliente que uno de sus pedidos está listo

Cabe destacar que el funcionamiento interno de este servicio se puede apreciar en el diagrama de secuencia de la Sección 3.5 - Figura 3.5.5.

- **Servicio notificar al dueño del sitio que un pedido fue cancelado**

Este servicio es similar al anterior (cambio de pedido a *listo*), pero aquí es el cliente quién cambia el estado del consumo. Para este caso, como se presenta en la Figura 5.14, se toman como referencia los consumos que posee el cliente *Gonzalo Rojas*.

Figura 5.14: Listado de consumos del cliente

Dentro de esos pedidos ya generados, al presionar en el botón "*Cancelar*" en algún pedido, el usuario puede proceder a la cancelación de ese consumo. Para este ejemplo, como se puede observar en la Figura 5.15, dicho usuario cancela el pedido #2 perteneciente al sitos *Artares*.

Figura 5.15: Cancelación de pedido por parte del cliente

Como resultado de dicha cancelación (cambio de contexto), como se puede observar en la Figura 5.16, se genera un aviso para el dueño del sitio *Artares*.

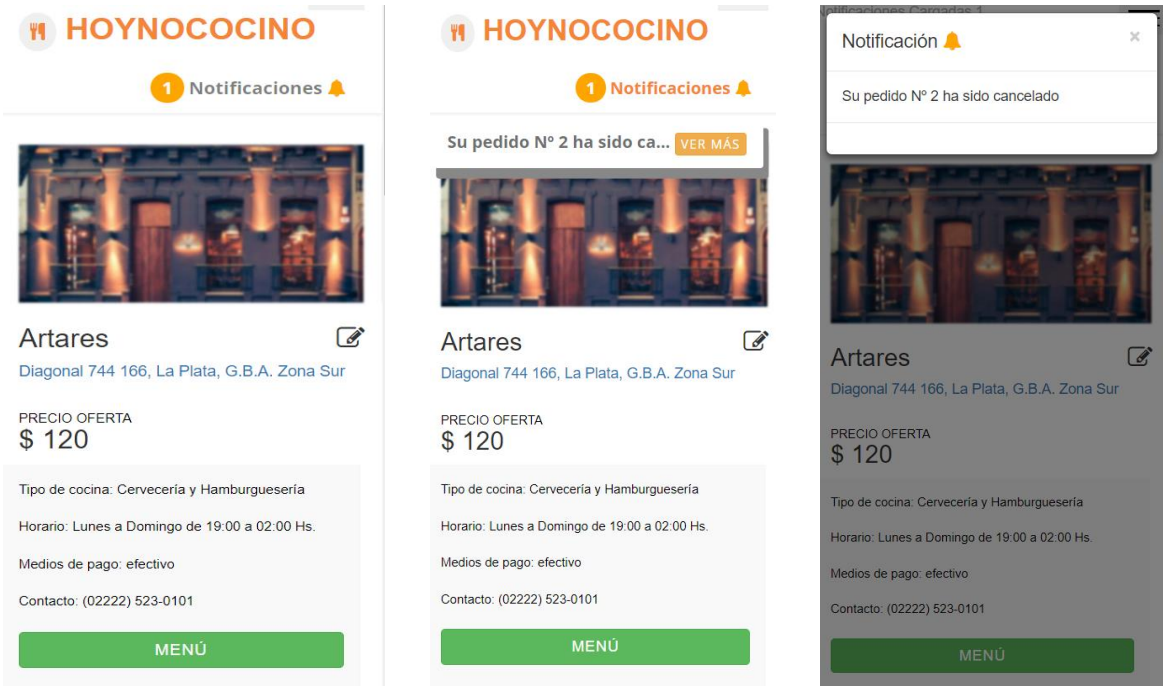


Figura 5.16: Notificación de pedido cancelado

Además de la notificación, en el listado de consumos del dueño se modifica el estado del pedido #2 cancelado, como se puede observar en la Figura 5.17.

Figura 5.17: Estado del pedido actualizado

Cabe mencionar que al cancelar el pedido, se genera un cambio de contexto y comienza a procesarse de la misma manera que los anteriores servicios. Los detalles

internos que suceden a nivel del modelo se pueden visualizar en el diagrama de secuencia de la Sección 3.5 - Figura 3.5.6.

De esta manera, se pudo apreciar cómo funcionan los servicios contextuales del prototipo. Más específicamente, como ciertos cambios de contexto se dan por la movilidad del usuario, como es el cambio de posición, o porque los usuarios (clientes o dueños) cambian alguna preferencia o cambian el estado de un pedido. Ante cualquier cambio de estos contextos, el prototipo reacciona y verifica qué servicio contextual debe ejecutarse y acorde a esto actualizar ya sea la visualización que recibe el usuario y/o enviar notificaciones.

6. Conclusiones y Trabajos Futuros

En este capítulo se presenta las conclusiones que se pueden mencionar a partir del trabajo realizado en esta tesina. Luego, se proponen trabajos a realizar en un futuro que, por tener un tiempo limitado o porque dichos trabajos excedían el alcance de lo que se quiere mostrar en ésta tesina, no fueron abordados en la misma.

A lo largo de esta tesina se investigaron diversos modelos contextuales (Secciones 2.3 y 2.4), los cuales se tomaron como base para diseñar el modelo final presentado. En el transcurso hasta obtener dicho modelo, se implementaron otros modelos (*Dominio* y *Contextual*) que luego fueron extendidos y rediseñados.

En la Sección 3.2, se presentó una solución de modelado que fue pensado para el ámbito de *Sistemas Gastronómicos*, el cual contaba con información del dominio y del contexto de manera acoplada. Esto generaba que el mecanismo de observación de los cambios del contexto esté a cargo de clases que no tenían esas responsabilidades, y además producía que las mismas estuvieran sobrecargadas de comportamiento. Por otro lado, dicho modelo, cumplía con uno de los objetivos buscados que era obtener un modelo de dominio con información en el ámbito gastronómico. Pero, como en este trabajo es de interés las características de contexto, para brindar servicios contextuales, dicho modelo se presentaba poco flexible y era limitada su escalabilidad.

Posteriormente, teniendo en cuenta el problema que surgía del modelo presentado Sección 3.2, se comienza a pensar en un rediseño tomándose como base el modelo de [Fortier et al., 2007], el cual tenía como objetivo separar el dominio de aplicación del dominio de adaptación. Este modelo de base provee una capa de contexto desacoplada de la del dominio, pero no era orientado a servicios como se pretendía para esta tesina. Una vez seleccionado el modelo base [Fortier et al., 2010], se realizó una integración junto a una simplificación presentada en [Challiol et al., 2007] y un rediseño obtenido de [Challiol et al., 2012], obteniendo como resultado el modelo de contexto para el dominio gastronómico presentado en la Sección 3.4 (Figura 3.4.1).

Luego de obtener la integración para el modelo de contexto, se retorna al primer modelo presentado en la Sección 3.2 (Figura 3.2.8), el cual contaba con información del dominio y del contexto de manera acoplada, para realizar una refactorización que tenía como objetivo quitar todas aquellas características contextuales. De esta manera, como se presenta en la Sección 3.4 (Figura 3.4.5) se logra desacoplar el contexto obteniendo un modelo solamente con información de dominio.

Como resultado final de las integraciones mencionadas, se obtuvo un modelo con dos capas desacopladas:

- una de ***contexto*** (Figura 3.4.1) que permite observar los cambios en un modelo de dominio y accionar acorde a estos (brindando servicios contextuales), sin que dicho dominio conozca este comportamiento explícitamente. De esta manera, queda representada una capa de contexto independiente del dominio con la cual se la integre.

- una de ***dominio*** (Figura 3.4.5), la cual se encarga del comportamiento de la aplicación, en este caso información gastronómica.

Con estos dos modelos anteriores se obtiene un modelo gastronómico general independiente de los mecanismos de observación de cambios y servicios contextuales que se brindan. Por otro lado, ofrece grandes ventajas a los desarrolladores que vayan a hacer uso del mismo, como la posibilidad de definir las características que van a observarse y los servicios contextuales que se accionarán al detectar cambios en las mismas, siendo sencilla la adaptación a cualquier otro dominio.

A su vez, los desarrolladores pueden implementar sub-clases de *ServiceHandler* para poder reaccionar a ciertos eventos que ocurran dentro de su aplicación y así brindar nuevos servicios de manera sencilla sin que afecte el comportamiento de los demás módulos. Como conclusión, se puede decir que el modelo propuesto en esta tesina es genérico y flexible para abarcar gran parte de los requerimientos que puedan tener las aplicaciones que utilizan servicios gastronómicos contextuales.

Sin embargo, dicho modelo también posee ciertas desventajas. La principal es que para utilizarlo es necesario entender su funcionamiento, el cual conlleva tiempo y dificultades que muchas veces terminan desenfocando al desarrollador del objetivo principal de su aplicación. Por otro lado, como sucede a menudo en los *frameworks*, el desacoplamiento y tratar cuestiones generales de aplicaciones produce una sobrecarga en las mismas, debido a la cantidad extra de clases, instancias y mensajes que se deben realizar para lograr y respetar el funcionamiento esperado. Si bien esto puede verse como una desventaja, una vez interpretado el modelo, es de utilidad debido a que ahorra trabajo y tiempo para el desarrollo de aplicaciones que brindan servicios contextuales.

En base al modelo propuesto, se presenta en el Capítulo 4 el prototipo implementado, el cual cuenta con un estado inicial para agilizar las pruebas de los servicios contextuales. En el Capítulo 5, se presenta como se comportan los servidos contextuales gastronómicos implementados en el prototipo. Cada uno de estos servicios se comportan acorde al modelo propuesto en esta tesina (Figuras 3.4.1 y 3.4.5).

A continuación se describirán algunos trabajos que podrían realizarse a futuro, estos trabajos, pueden ser tanto a nivel de modelado como de implementación.

A nivel de modelado, como se ya se mencionó, las clases del modelo propuesto pueden ser sub-clasificadas fácilmente para tener funcionalidades más específicas. Gracias a ello, se podría pensar en crear módulos que agrupen comportamientos específicos para un tipo de aplicación. Algunos ejemplos podrían ser:

- Extender el modelo, con un conjunto de servicios contextuales particulares que se adapten a cierta información relevante del usuario, como puede ser datos sobre su salud. Por ejemplo, en el caso de que alguna persona posea alguna enfermedad que le impida comer cierto tipo de comida, estos servicios particulares podrían recomendarle solo gastronomía acorde a su enfermedad o avisarle antes de confirmar un pedido que cierta comida no puede consumir.

- Extender la clase *ServiceEnvironments* para lograr incluir otros *subambientes* dentro del ambiente general. Por ejemplo, el ambiente general (clase *Ambiente_Servicio_Gastronomico*) puede tener como *subamientes* zonas, y a su vez, estas tener como *subambientes* a los sitios (*Ambiente_Servicio_Sitio*). Una zona podría ser un país, provincia, ciudad, barrio, entre otros. Las zonas se representarían agregando una clase (clase *Ambiente_Servicio_Zona*) que tenga como *subamientes* a los ambientes de los sitios. De esta manera, la aplicación podría brindar servicios contextuales y dar información sobre los sitios según la zona en la que se encuentra el usuario.

- Extender el modelo de dominio agregando las clases necesarias para contar con la posibilidad de pagar los servicios consumidos. Este impacto requiere analizar qué controles de seguridad son necesarios diseñar para poder realizar este tipo de operaciones online.

- Realizar pruebas con usuarios para detectar nuevos servicios contextuales específicos aún no soportadas por el modelo, y así enriquecer el mismo. Es decir, muchas veces los usuarios son los que indican los reales requerimientos útiles de este tipo de aplicaciones.

En cuanto al prototipo se pueden contemplar el agregado a futuro de, por ejemplo, las siguientes funcionalidades:

- Dar la opción al usuario de elegir el radio en metros/kilómetros para establecer la distancia de la cual está interesado y considera cerca. De esta configuración por parte del usuario van a depender los servicios que el prototipo le brinde, como las notificaciones de promociones y amigos cercanos.

- Generar una vista alternativa para los marcadores del mapa, para el caso en donde se presenta la situación de que varios amigos están en algún sitio cercano, y para evitar superposiciones de marcadores y lograr visualizarlos correctamente. El prototipo los mostraría como un grupo en el mapa dando la opción para poder acceder al mismo y conocer los amigos que se encuentran en el sitio.

- Contemplar cuestiones de privacidad del usuario, por ejemplo: darle la posibilidad al usuario para que acepte o no que su ubicación (marcador) se muestre en el mapa de sus amigos.

- Proveer un sistema de chat como parte del prototipo que permita que un grupo de amigos pueda coordinar el encuentro en un determinado sitio. Podría ser que el sistema les sugiera lugares acordes a las preferencias de comidas que tienen todos los integrantes del grupo.

- Las notificaciones podrían implementarse con una política push[39], es decir, contar

[39] Página con información políticas push: https://developers.google.com/web/fundamentals/codelabs/push-notifications/?hl=es (Último acceso 18-02-2018)

con un servicio que esté "*escuchando*" si hay alguna notificación encolada para mostrarla tanto en desktop como en celulares. Este tipo de notificación permitiría, por ejemplo, que se puedan mostrar notificaciones en el celular sin que el usuario tenga que tener abierto el navegador. Algo similar ocurriría desde el navegador en desktop, solicitando previamente los permisos al usuario para recibir notificaciones.

- Implementación de *Realidad Aumentada*[40], por ejemplo, el prototipo podría visualizar las promociones y menúes que tiene un sitio gastronómico cuando el usuario pasa cerca del mismo.

- Contar con la posibilidad de proveer una versión "*off-line*" que contenga la información y el contenido del momento actual del usuario. Esa información puede contener lo referido a su perfil (preferencias, posición, etc) y los sitios cercanos. Esta versión se utilizaría en los casos de no contar con conectividad, entonces se lograría seguir operando con el prototipo. El usuario podría disponer de servicios aunque no se cuente con toda la funcionalidad dado que para ciertos contextos se necesita conectividad.

- Analizar cómo se debería comportar un prototipo similar al implementado cuando los sitios gastronómicos se encuentra, por ejemplo, dentro de un shopping. En este caso, hay que determinar la mejor manera de mostrar estas opciones sobre el mapa. Para esto, se podría explorar el uso de mapas *indoor*, por ejemplo, usando mapas de *Google Indoor*[41]. Pero además, se debe analizar la incorporación de mecanismos de sensado *indoor* (por ejemplo, códigos QR o *beacons*[42]), los cuales podrían ser soportado por el modelo propuesto. Sin embargo esto tiene un impacto en el prototipo, ya que la posición actual del usuario pasa a ser identificada con varios mecanismos de sensado.

- Agregar al prototipo la posibilidad de que se permita realizar pagos online de los servicios que ofrecen los sitios gastronómicos. Para esto, el dueño de cada sitio podrá configurar y decidir qué medios de pago acepta y del otro lado, los clientes al momento de realizar algún consumo, poder seleccionar el medio de pago y efectuarlo. También el dueño podría realizar promociones dependiendo el medio de pago que selecciona el cliente para abonar sus servicios.

En las pruebas realizadas en esta tesina se simulo el uso del GPS. A futuro sería fundamental realizar pruebas con usuarios en ambientes reales, para detectar cómo se comporta tanto el GPS como la conectividad con datos móviles.

---

[40] Página con la definición de realidad aumentada https://definicion.de/realidad-aumentada (Último acceso 22-02-2018)

[41] Página de mapas de *Google Indoor*: https://www.google.com/maps/about/partners/indoormaps (Último acceso 01-03-2018)

[42] Página de *Estimote* (uno de los proveedores de *beacons* más populares actualmente): https://estimote.com (Último acceso 01-03-2018)

7. Referencias Bibliográficas

[Abowd et al., 1999] Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M., & Steggles, P. (1999, September). Towards a better understanding of context and context-awareness. In *International Symposium on Handheld and Ubiquitous Computing* (pp. 304-307). Springer Berlin Heidelberg.

[Bauer and Dey, 2016] Bauer, C., & Dey, A. K. (2016). Considering context in the design of intelligent systems: Current practices and suggestions for improvement. Journal of Systems and Software, 112, 26-47.

[Bäumer et al., 1998] Bäumer, D., Riehle, D., Siberski, W., & Wulf, M. (1998). The role object pattern. In *Washington University Dept. of Computer Science*.

[Bazzocco, 2005] Bazzocco, J. (2005). *Arquitectura orientada a objetos para aplicaciones sensibles al contexto* (Tesis Maestría, Facultad de Informática, UNLP).

[Becker & Nicklas, 2004] Becker, C., & Nicklas, D. (2004). Where do spatial context-models end and where do ontologies start? A proposal of a combined approach.

[Brown, 1997] Brown, P. J., Bovey, J. D., & Chen, X. (1997). Context-aware applications: from the laboratory to the marketplace. *IEEE personal communications*, *4*(5), 58-64.

[Brown, 1995] Brown, P. J. (1995). The stick-e document: a framework for creating context-aware applications. *Electronic Publishing-Chichester-*, *8*, 259-272.

[Bootstrap, 2011] Página de *Bootstrap*: http://getbootstrap.com (Último acceso 01-02-2018).

[Challiol et al., 2007] Challiol, C., Fortier, A., Gordillo, S., & Rossi, G. (2007). A flexible architecture for context-aware physical hypermedia. In *Database and Expert Systems Applications, 2007. DEXA'07. 18th International Workshop on* (pp. 590-594). IEEE.

[Challiol et al., 2012] Challiol, C., Rossi, G., Gordillo, S. E., & Fortier, A. (2012). Separation of concerns in mobile hypermedia: architectural and modeling issues. *Handbook of Research on Mobile Software Engineering: Design, Implementation and Emergent Applications*, *1*, 211-233

[CSS3, 1996] Página del estándar *W3C*: https://www.w3.org/standards/techs/css#w3c_all (Último acceso 01-02-2018)

[Dey, 1998] Dey, A. K. (1998). Context-aware computing: The CyberDesk project. In *Proceedings of the AAAI 1998 Spring Symposium on Intelligent Environments* (pp. 51-54).

[Dey, 2000] Dey, A. K. (2000). *Providing architectural support for building context-aware applications* (Doctoral dissertation, Georgia Institute of Technology).

[Dey & Abowd, 2000] Dey, A. K., & Abowd, G. D. (2000). The context toolkit: Aiding the development of context-aware applications. In *Workshop on Software Engineering for wearable and pervasive computing* (pp. 431-441).

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

[Duran-Limon et al., 2003] Duran-Limon, H. A., Blair, G. S., Friday, A., Grace, P., Samartzidis, G., Sivaharan, T., & Wu, M. (2003). Context-aware middleware for pervasive and ad hoc environments. *Context, Tech. Rep*.

[ElTenedor, 2011] Página de *ElTenedor*: https://www.thefork.com (Último acceso 23-9-2017).

[Emmanouilidis et al., 2013] Emmanouilidis, C., Koutsiamanis, R. A., & Tasidou, A. (2013). Mobile guides: Taxonomy of architectures, context awareness, technologies and applications. *Journal of Network and Computer Applications*, *36*(1), 103-125.

[Fortier et al., 2007] Fortier, A., Challiol, C., Rossi, G., & Gordillo, S. (2007). Physical Hypermedia: a Context-Aware approach. In *Proceedings of the CAiSE* (Vol. 7, pp. 499-513).

[Fortier et al., 2010] Fortier, A., Rossi, G., Gordillo, S. E., & Challiol, C. (2010). Dealing with variability in context-aware mobile software. *Journal of Systems and Software*, *83*(6), 915-936.

[Gamma et al., 1995] Vlissides, J., Helm, R., Johnson, R., & Gamma, E. (1995). Design patterns: Elements of reusable object-oriented software. *Reading: Addison-Wesley*, *49*(120), 11.

[GoChef, 2016] Página de *GoChef*: https://www.gochef.com (Último acceso 23-9-2017).

[GoogleMaps, 2005] Página de *GoogleMaps:* http://www.wampserver.com (Último acceso 01-02-2018).

[HTML5, 2014] Página de *HTML5*: https://www.w3.org/TR/html52 (Último acceso 01-02-2018)

[Javascript, 1995] Página oficial de *Javascript*: https://www.javascript.com (Último acceso 01-02-2018)

[jQuery, 2006] Página de *jQuery*: https://jquery.com/ (Último acceso 23-9-2017).

[Korpipää et al., 2003] Korpipaa, P., Mantyjarvi, J., Kela, J., Keranen, H., & Malm, E. J. (2003). Managing context information in mobile devices. *IEEE pervasive computing*, *2*(3), 42-51.

[Leonhardt, 1998] Leonhardt, U. (1998). *Supporting location-awareness in open distributed systems* (Doctoral dissertation, University of London).

[Loke, 2006] Loke, S. (2006). *Context-aware pervasive systems: architectures for a new breed of applications*. CRC Press.

[MySQL, 1995] Página de *MySQL:* https://www.mysql.com (Último acceso 01-02-2018).

[Musi Gentile, 2015] Musi Gentile, F. A. (2015). *Modelo de gestos considerando contexto* Tesina de Grado. Facultad de Informática, UNLP.

[MockingBot] Página de *MockingBot:* https://mockingbot.com (Último acceso 01-02-2018)

[Nielsen, 2016] Nielsen, Tops Of 2016: Digital, The Nielsen Company, 2016.

[Pascoe, 1998] Pascoe, J. (1998, October). Adding generic contextual capabilities to wearable computers. In *Wearable Computers, 1998. Digest of Papers. Second International Symposium on* (pp. 92-99). IEEE.

[PedidosYa, 2009] Página de *PedidosYa*: https://www.pedidosya.com (Último acceso 23-9-2017).

[PHP, 1995] Página de *PHP*: http://php.net (Último acceso 01-02-2018).

[phpMyAdmin, 1998] Página de *phpMyAdmin*: https://www.phpmyadmin.net (Último acceso 01-02-2018).

[Universidad ORT Uruguay, 2015] Galico, D., Natanzon, K., Vega, C., Matalonga, S., & Solari, M. (2015). Software Sensible al Contexto: definiciones y desarrollo de un estudio de caso en Google Glass. Documento de Investigación, Nro. 13, mayo 2015. Universidad ORT Uruguay. Facultad de Ingeniería. ISSN 1688-6372

[REST, 2001] Página del estándar *W3C*: https://www.w3.org/2001/sw/wiki/REST (Último acceso 23-9-2017).

[Restaurantes.com, 2016] Página de *Restaurantes.com*: https://www.restaurantes.com (Último acceso 23-9-2017).

[Ryan et al., 1999] Ryan, N., Pascoe, J., & Morse, D. (1999). Enhanced reality fieldwork: the context aware archaeological assistant. *Bar International Series*, *750*, 269-274.

[Salber et al., 1999] Salber, D., Dey, A. K., & Abowd, G. D. (1999, May). The context toolkit: aiding the development of context-enabled applications. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems* (pp. 434-441). ACM.

[Schilit, 1994] Schilit, B. N., & Theimer, M. M. (1994). Disseminating active map information to mobile hosts. *IEEE network*, *8*(5), 22-32.

[Schilit et al., 1994] Schilit, B., Adams, N., & Want, R. (1994, December). Context-aware computing applications. In *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on* (pp. 85-90). IEEE.

[Yelp, 2004] Página de *Yelp*: https://www.yelp.com (Último acceso 23-9-2017).

[WampServer, 2004] Página de *WampServer:* http://www.wampserver.com (Último acceso 01-02-2018).

[Ward, 1997] Ward, A., Jones, A., & Hopper, A. (1997). A new location technique for the active office. *IEEE Personal communications*, *4*(5), 42-47.

[Weyns et al., 2015] Weyns, D., Caporuscio, M., Vogel, B., & Kurti, A. (2015, September). Design for sustainability= Runtime adaptation∪ evolution. In Proceedings of the 2015 European Conference on Software Architecture Workshops (p. 62). ACM.

[Woolf and Johnson, 1996] Woolf, B., & Johnson, R. (1996). The type object pattern. *Pattern Languages of Program Design*, *3*, 132.

Anexo A: Otras Aplicaciones Móviles en al Ámbito de Sitos Gastronómicos

En este anexo se presentan otras aplicaciones relevadas para el dominio gastronómico, más allá de las mencionadas y analizas en la Sección 2.2.

Las aplicaciones relevadas en este anexo son: *LocalEats* [43], *cookapp* [44], *Waitry* [45], *Restorando*[46], *My Table*[47], *Go-Chef Madrid*[48] y *FourSquare*[49]. En la Tabla A.1 se presenta un resumen de los servicios provistos por las aplicaciones móviles mencionadas anteriormente. Se puede apreciar que los servicios analizados para cada una de estas aplicaciones son los mismos que los previamente analizados en la Tabla 2.2.1 para las aplicaciones presentadas Sección 2.2. Las aplicaciones de este anexo permiten apreciar que existen más aplicaciones más allá de las analizadas en la Sección 2.2.

Tabla A.1: Resumen de los servicios provistos por las aplicaciones móviles analizadas

		APLICACIONES						
		LocalEats	cookapp	Waitry	Restorando	My Table	Go-Chef Madrid	FourSquare
SERVICIOS	Visualización de lugares según la ubicación	✓	✓	✓	✓	✓	✓	✓
	Búsqueda con filtros de distancia, precio y horarios de apertura	✓	✓	✓	✓	✓	✓	✓
	Notificar por promociones	✓	-	-	✓	✓	-	-
	Lista de Favoritos	✓	-	-	✓	-	-	✓
	Añadir reseñas, sugerencias, fotos y calificaciones	-	✓	✓	✓	✓	-	✓
	Compartir en redes sociales	-	✓	✓	✓	✓	✓	✓
	Realizar reservas	✓	✓	✓	✓	✓	-	✓
	Seguir y ver actividad de amigos	-	-	-	-	-	-	✓
	Descripción menú y especialidades	✓	✓	✓	✓	✓	✓	✓
	Sumar puntos para obtener descuentos	-	-	-	✓	-	✓	-
	Suscripción Newsletter	-	✓	-	✓	-	-	-

[43] Página de LocalEats: https://www.localeats.com (Último acceso 01-03-2018)
[44] Página de *cookapp*: https://www.cookapp.com (Último acceso 01-03-2018)
[45] Página de *Waitry*: https://www.waitry.net (Último acceso 01-03-2018)
[46] Página de *Restorando*: https://www.restorando.com (Último acceso 01-03-2018)
[47] Página de *Google Play*: https://play.google.com/store/apps/details?id=com.mytable.restaurantguide&hl=es (Último acceso 01-03-2018)
[48] Página de *App Store*: http://www.app-store.es/gochef-restaurantes-madrid (Último acceso 01-03-2018)
[49] Página de *FourSquare*: https://es.foursquare.com (Último acceso 01-03-2018)

		LocalEats	cookapp	Waitry	Restorando	My Table	Go-Chef Madrid	FourSquare
	Notificación de pedido listo	-	✓	-	-	-	-	-
	Solicitar Delivery de pedidos	-	-	-	-	-	✓	-
	Otros servicios no gastronómicos	-	✓	✓	-	-	-	✓
	Personalizar menú y realizar pedido	-	✓	-	-	-	-	-
	Como llegar al lugar	-	-	-	-	-	-	✓
	Pago virtual	-	✓	✓	-	-	-	-

Anexo B: Mockups del Prototipo

En este anexo se muestra la elaboración de *Mockups* realizados previamente al desarrollo del prototipo. El objetivo de ellos fue facilitar la posterior construcción del prototipo logrando un maquetado que sirva como guía para el desarrollo. Estos *mockups* fueron realizados usando la herramienta *MockingBot 3.0* [MockingBot].

A continuación se muestran algunas pantallas que fueron creadas. Cabe destacar que, no se diseñaron todas las pantallas que posee el prototipo ya que varias son muy similares.

En la Figura B1 se muestra un bosquejo de la información que sería útil que recibiera el usuario cliente cuando está cerca de un sitio cercano.

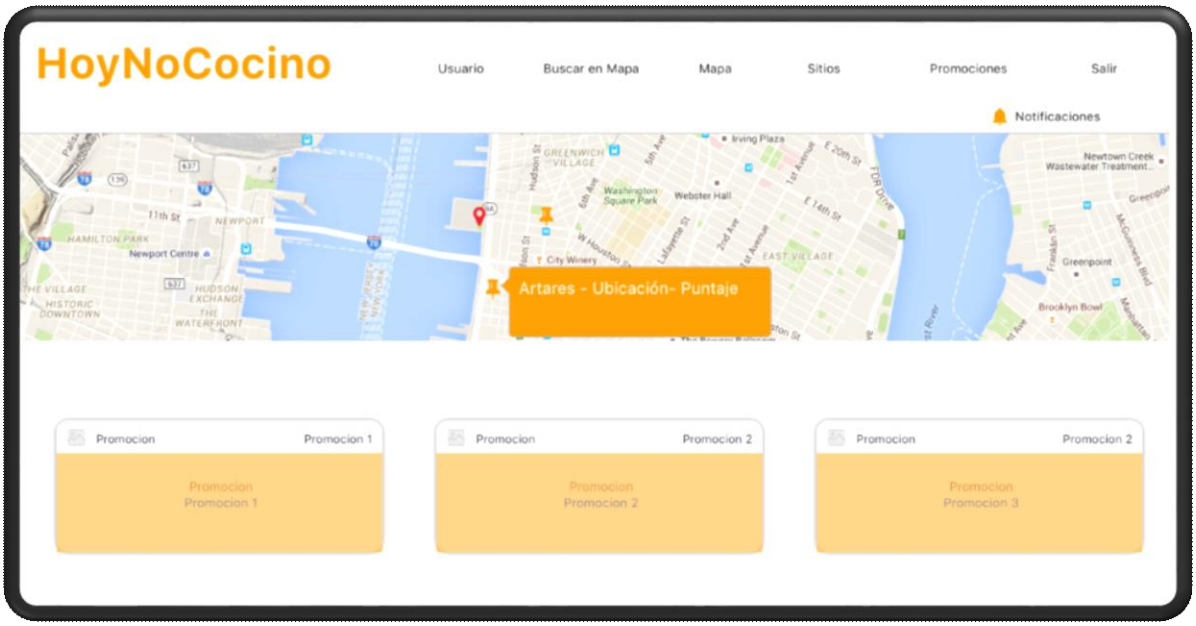


Figura B1: Mockup inicio del usuario *cliente* con información de un sitio cercano

En la Figura B2 se muestra un bosquejo de la información que podría ver un usuario cliente cuando está cerca de varios sitios cercanos y además tiene amigos en estos. Se puede observar que se le notifica que tiene un amigo cerca. Ya en este bosquejo se empieza a detectar la superposición de *markets*, una de las problemáticas mencionadas en la Sección 4.4.

En la Figura B3 se muestra un bosquejo de la información que podría ver un usuario cliente cuando uno de sus pedidos está listo. Este aviso se identifica un área de notificaciones en la parte superior de la pantalla.

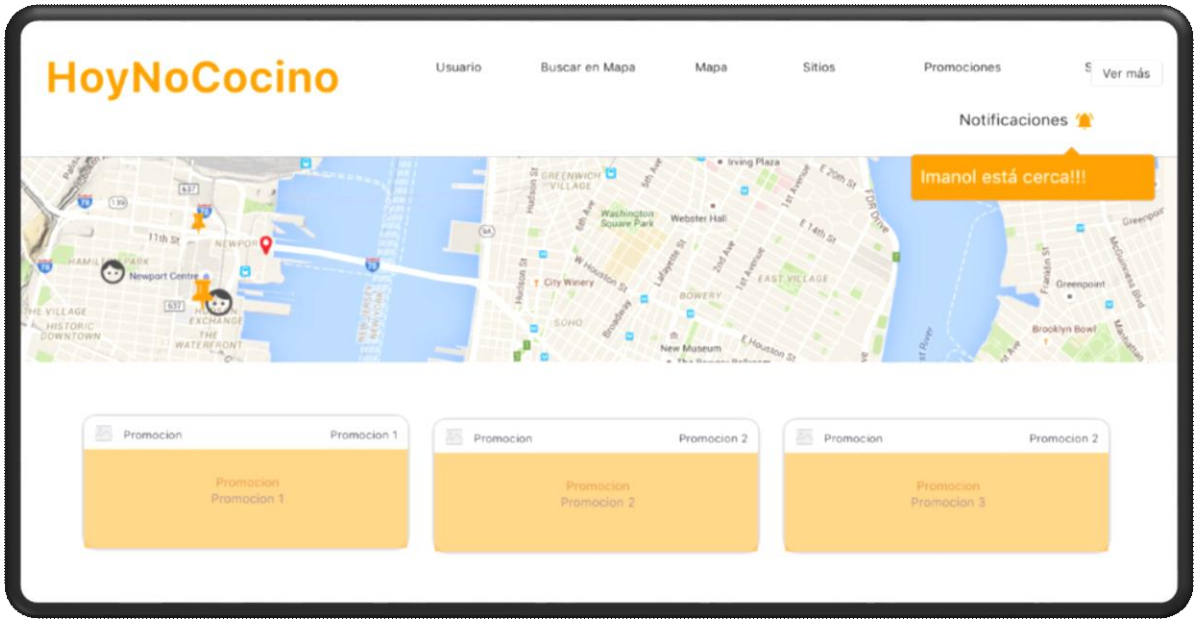


Figura B2: Mockup incio del usuario cliente con amigos y sitios cercanos

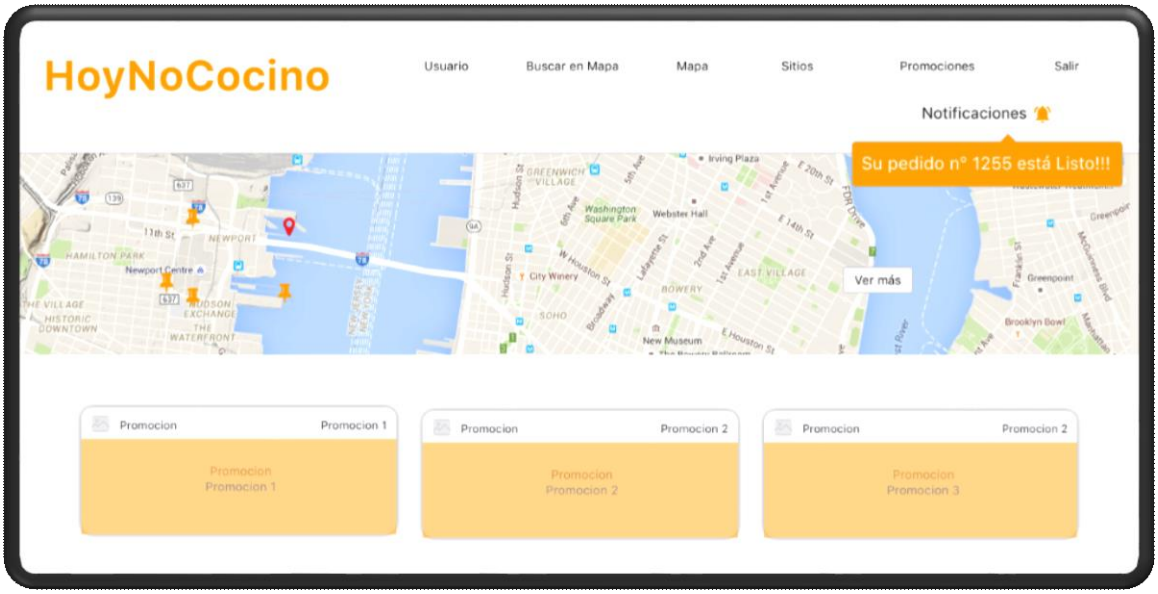


Figura B3: Mockup inicio del usuario cliente con notificación de pedido en estado listo

En la Figura B4 se muestra un bosquejo del listado de pedidos para el sitio *Artares*. Se puede observar el estado de los mismos así como también el ver más información de los mismos.

Figura B4: Mockup listado de pedidos que posee el sitio Artares

En la Figura B5 se muestran dos del sitio *Artares*. En la Figura B5.a se puede observar dicho sitio sin notificaciones, mientras que en la Figura B5.b se muestra como visualizar una notificación relacionada al sitio.

a. **b.**

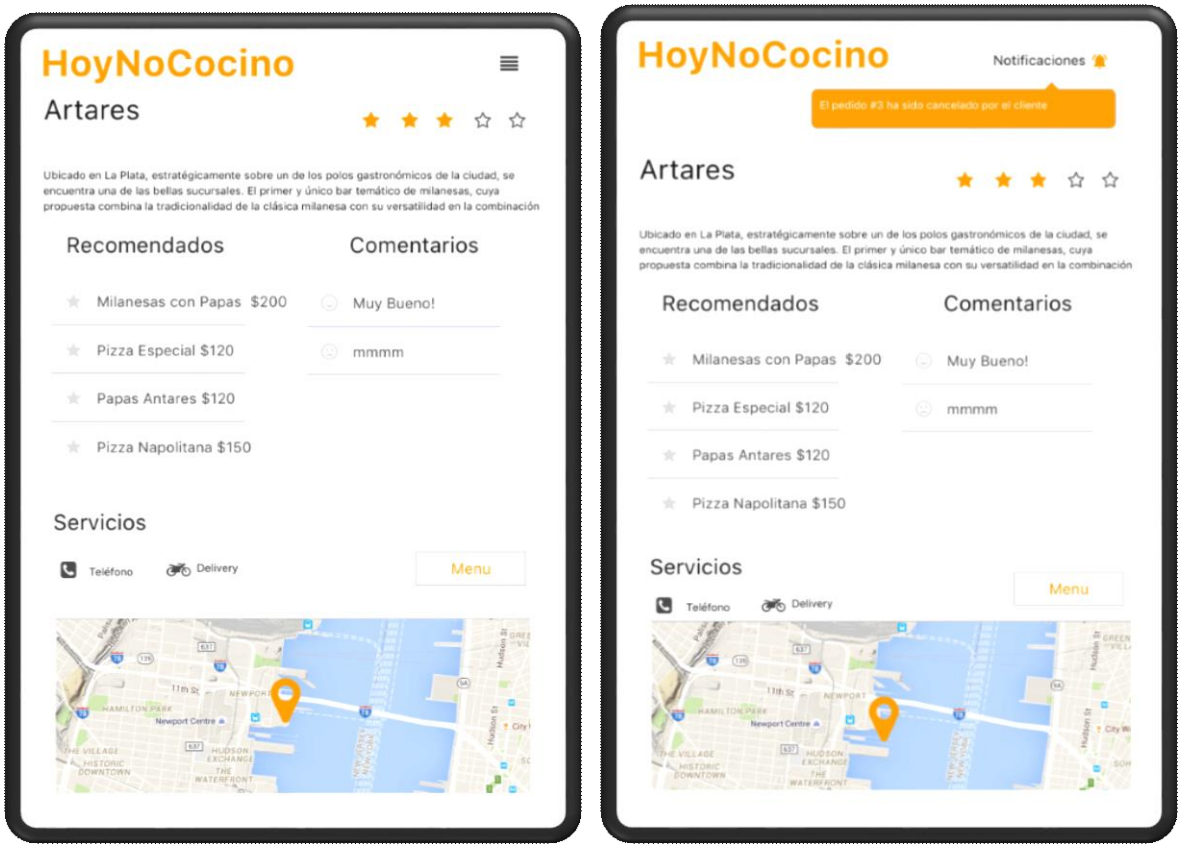


Figura B5: Mockup vista del sitio *Artares* sin notificación (a) y con notificación (b) respetivamente