UNIVERSIDAD DE MURCIA

FACULTAD DE INFORMÁTICA

A Model-Driven Approach for Context Modeling in Context-Aware Systems

Ingeniería Dirigida por Modelos para el Modelado de Contexto en Sistemas Context-Aware

D. José Ramón Hoyos Barceló

2015

A Model-Driven Approach for Context Modeling in Context-Aware Systems

A dissertation presented by
José Ramón Hoyos Barceló
and supervised by
Jesús García Molina & Juan A. Botía Blaya

In partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the subject of Computer Science

University of Murcia
December 2015

A mi mujer Gema
y a mi hija Inés

Resumen extendido

El término *Computación Ubicua* (también conocido como *Computación Pervasiva*) hace referencia a ordenadores y dispositivos que pueden encontrarse a nuestro alrededor por todas partes, pero que están diseñados de forma que no molesten a las personas y faciliten su interacción con los usuarios de una forma natural. Esto hace que podamos usarlos sin ser conscientes de ello, ya que se encontrarían integrados en nuestra vida diaria.

El desarrollo de estos sistemas involucra diferentes áreas de la *Informática*, entre las que destacan la *Ingeniería del Software*, las *Redes de Comunicaciones*, la *Interacción Persona-Ordenador* y la *Inteligencia Artificial*. La *Computación Ubicua* también está relacionada con otros paradigmas como la *Computación Sensible al Contexto* (*Context-Aware Computing*), la *Computación Móvil* y también con tecnologías relacionadas con sensores y redes de comunicación.

Para poder proporcionar a los usuarios unos servicios adecuados, estos sistemas deben ser conscientes de su entorno, es decir, la información del contexto que los rodea. La *Computación Context-Aware* (utilizaremos el término "context-aware" para referirnos a los sistemas *Sensibles al Contexto*) se caracteriza por ser un modelo de computación en el cual los usuarios interaccionan con diferentes ordenadores o sistemas software que son capaces

de percibir su entorno y adaptar automáticamente su comportamiento a los cambios que se produzcan en él. El entorno de los sistemas se define por su información de contexto que incluye datos tales como la localización de un usuario, el día y la hora, las actividades que está realizando el usuario o los parámetros de configuración de algún dispositivo. Por lo tanto, uno de los componentes clave de los sistemas context-aware es el componente encargado de manejar la información de contexto, que es el responsable de mantener una representación de dicha información y manipularla.

La mayoría de los sistemas context-aware suelen considerar que la información de contexto que manejan y de la cual depende su funcionamiento es siempre completa y fiable. Sin embargo, esta información a menudo es imperfecta. Los sensores encargados de obtener la información del entorno adolecen de limitaciones técnicas en el sentido de que los valores de los datos que suministran son sólo una aproximación a los valores del mundo real. Otros factores tales como interferencias en las redes de comunicaciones o un fallo físico en un sensor pueden afectar también a la información de contexto.

El hecho de que un sistema context-aware confíe plenamente en que la información que está manejando es correcta puede conducir a un comportamiento inesperado del sistema. Por tanto dichos sistemas deberían de conocer no solo los datos relativos a la información de contexto sino también la calidad de dicha información.

Todo lo expuesto con anterioridad, nos sugiere que la construcción de aplicaciones software context-aware es una tarea compleja que implica considerar los problemas inherentes a la adquisición de información de contexto fiable, a la representación adecuada del contexto y a los mecanismos de adaptación necesarios para llevar a cabo un comportamiento correcto del sistema ante cambios en el entorno. Por lo tanto, es necesario utilizar un enfoque sistemático basado en los sólidos principios de la *Ingeniería del Software* para poder tratar dicha complejidad.

La *Ingeniería Dirigida por Modelos* (en inglés Model-Driven Software Engineering, MDE) promueve un uso sistemático de modelos para elevar el nivel de abstracción de las especificaciones del software e incrementar el nivel de automatización en su desarrollo. Los modelos ayudan a los desarrolladores a razonar y a predecir sobre los problemas y soluciones que puedan aparecer en la creación de un sistema software.

Existen varios enfoques para el modelado del contexto cuyo objetivo es representar la información de contexto, pero todos presentan alguna limitación. Por ejemplo, son específicos para un framework determinado, tienen un nivel de abstracción bajo o carecen de una representación adecuada para algunos de los elementos de contexto que hay que modelar. MDE hace uso de modelos para el desarrollo de los sistemas, y de transformaciones de modelos para generar automáticamente artefactos software que son parte de la aplicación final.

El trabajo que presentamos en esta tesis propone un enfoque MDE que define una arquitectura generativa para sistemas context-aware. Dicha arquitectura está basada en un *Lenguaje Específico de Dominio* (en inglés *Domain-Specific Language*, DSL) llamado MLContext que permite representar la información de contexto teniendo en cuenta aspectos de calidad de la misma. Utilizando MLContext se pueden crear modelos de contexto a partir de los cuales es posible generar artefactos software relacionados con el manejo de la información de contexto para distintos middlewares (por ejemplo, ontologías o código Java).

1. Objetivos

Dado el objetivo general de presentar una solución MDE para automatizar el manejo de contexto en middlewares context-aware, se han planteado cuatro objetivos específicos en esta tesis.

i) Desarrollar un Lenguaje Específico del Dominio para el modelado del contexto con un nivel alto de abstracción.

Un DSL con un nivel alto de abstracción y que además sea independiente de la plataforma proporciona varios beneficios a los desarrolladores que se enfrentan al problema de modelar el contexto. Por un lado, no es necesario que traten con detalles específicos de implementación o relativos a algún middleware o framework. Por otro lado, un DSL es más sencillo de aprender y de usar que otros lenguajes de programación de propósito general porque los conceptos que se manejan son cercanos a los conceptos del dominio y esto hace que el proceso de desarrollo sea más simple y claro que si se realizara a un nivel de abstracción más bajo. Este DSL debería estar orientado también a reducir el tamaño de las especificaciones de contexto para conseguir un aumento en productividad. Para finalizar, el uso de un DSL favorece la interacción entre el desarrollador y el usuario, especialmente cuando se está validando el modelo de contexto.

El diseño del DSL debería estar precedido de un análisis del dominio, para capturar los aspectos más importantes relacionados con el modelado del contexto mediante la definición de su vocabulario y conceptos básicos.

ii) Definir un conjunto de constructores para el DSL de manera que se pueda especificar la calidad de la información de contexto y los requisitos de calidad.

Como se comentó anteriormente, la falta de información sobre la calidad del contexto (QoC) puede originar que se degrade el rendimiento de las aplicaciones context-aware y puede conducir a un comportamiento incorrecto. La información sobre QoC puede ayudar a este tipo de aplicaciones a resolver situaciones de conflicto o de incertidumbre. Un DSL para el modelado de contexto debería tener algunos constructores que le permitieran representar la QoC de la información y también permitir a los desarrolladores definir niveles de calidad y requisitos de calidad para la información de contexto.

iii) Generar automáticamente artefactos software para diferentes context-aware middlewares.

El DSL para modelar la información será el elemento central de una arquitectura basada en MDE capaz de generar automáticamente artefactos software relacionados con el manejo del contexto para middlewares context-aware (por ejemplo, ontologías, código Java o ficheros XML). Un motor de transformación será el encargado de transformar los modelos que han sido especificados mediante el DSL en los artefactos software.

Este enfoque debe ser independiente de la plataforma (es decir, el modelo de contexto debería poder reutilizarse en diferentes middlewares) y también debe ofrecer una separación de aspectos para favorecer la adaptación ante cambios en los requisitos. Esta separación de aspectos favorecería la independencia de aplicaciones, dado que un modelo de contexto podría ser utilizado por diferentes aplicaciones basadas en un mismo framework y que han sido desarrolladas para el mismo dominio.

iv) Definir un método para el modelado del contexto y la QoC.

El enfoque MDE definido en esta tesis deberá ir acompañado de la propuesta de un método simple para modelar el contexto y la QoC en las primeras etapas del proceso de desarrollo de una aplicación context-aware. El método estará basado en las características particulares de nuestro enfoque y recomendará los pasos a seguir en el modelado del contexto, de manera que los desarrolladores puedan aprovechar todo el potencial de nuestra propuesta.

2. Metodología

Para alcanzar los objetivos de la tesis, hemos seguido la metodología "*Design Science Research Methodology*" (DSRM) (metodología de investigación en ciencias de diseño). Esta metodología comprende un proceso de diseño que consta de seis actividades: (1) Identificación del problema y motivación, (2) Definir los objetivos de una solución, (3) Diseñar el desarrollo, (4) Demostración, (5) Evaluación y (6) Conclusiones y comunicación de los resultados. Se trata de un proceso iterativo ya que la construcción y evaluación de nuevos artefactos sirve como retroalimentación para obtener un diseño e implementación mejores de la solución final.

Siguiendo las actividades del DSRM, comenzamos con la identificación del problema y la motivación: crear un enfoque basado en MDE para solventar las limitaciones de las propuestas existentes para modelar contexto y QoC durante el desarrollo de sistemas context-aware. Para esta actividad se realizó un estudio del estado del arte del problema.

A continuación, realizamos la segunda actividad que implicaba idear una solución al problema e identificar los objetivos que se debían alcanzar para conseguir construir la solución. Para ello se estudiaron en detalle los enfoques actuales para el modelado de contexto y su calidad. Nuestra solución se basa en MDE porque nos proporciona conceptos, técnicas, métodos y herramientas para representar explícitamente la información de contexto en forma de modelos y para automatizar la generación de artefactos software por medio de transformaciones de modelos. En este sentido la información de contexto junto con su calidad se representan por medio de modelos descritos por metamodelos, y se establece una correspondencia entre un metamodelo y un framework concreto context-aware para el que se va a generar el artefacto. Esta correspondencia se define por medio de una transformación de modelos.

La tercera actividad se organizó en dos etapas. Primero se abordó el diseño y la implementación de un lenguaje específico de dominio (DSL) llamado MLContext para modelar la información de contexto que necesitan los sistemas context-aware. Después, extendimos el DSL con constructores para la QoC. Durante esta segunda etapa definimos la estrategia (método) que recomienda a los desarrolladores cómo modelar el contexto. La construcción de una solución MDE completa envuelve la creación de diversas herramientas tales como: (1) un editor para crear las especificaciones del DSL, (2) un inyector, para convertir las especificaciones realizadas con el DSL en un modelo (que conforma con un metamodelo que representa la sintaxis abstracta del DSL) y (3) un motor de transformación que implementa la correspondencia entre la sintaxis abstracta del DSL y un framework context-aware concreto, por medio de transformaciones modelo-a-texto. Estas herramientas las implementamos utilizando otras herramientas basadas en el framework para modelado de Eclipse (EMF) y se integraron dentro de la plataforma Eclipse.

En la cuarta actividad, demostramos que la solución funcionaba resolviendo varias instancias del problema para diferentes middlewares context-aware. Para ello utilizamos dos casos de estudio para mostrar la capacidad de nuestro enfoque a la hora de modelar el contexto y generar el código para varios middlewares con y sin soporte para QoC. Primero utilizamos un caso de estudio basado en el entorno de un hospital cuando la calidad del contexto no estaba aún definida, y luego utilizamos un segundo escenario basado en ventas relámpago (flash-sales) para demostrar cómo modelar los aspectos de calidad.

En la quinta actividad, evaluamos nuestra solución comparando los objetivos con los resultados obtenidos. La evaluación se realizó en términos de facilidad de uso, expresividad y productividad.

Para finalizar, durante las conclusiones identificamos algunas mejoras y líneas de investigación futuras que se podrían seguir. Los resultados de nuestro trabajo se han difundido mediante la publicación de varios artículos de investigación en distintas revistas y conferencias. En esos artículos mostramos la novedad de las contribuciones de nuestro trabajo.

3. Conclusiones y Contribuciones

Nuestro trabajo propone un enfoque MDE para el modelado de contexto en el desarrollo de sistemas context-aware. En este sentido, esta tesis ha proporcionado las siguientes contribuciones: (1) un análisis de dominio sobre el contexto y la calidad del contexto, (2) un lenguaje específico del dominio, llamado MLContext, con un nivel alto de abstracción, (3) una extensión para el modelado de la calidad de contexto y (4) un enfoque MDE para el modelado del contexto.

Los requisitos identificados durante el análisis de dominio se tomaron en consideración durante el diseño de MLContext. El lenguaje MLContext se diseñó para ser independiente de la plataforma y para que fuera sencillo de aprender y de usar. La extensión para el modelado de la calidad de contexto permite a los desarrolladores expresar no sólo los parámetros de calidad de los productores de la información de contexto sino también los requisitos de calidad para una aplicación context-aware. Finalmente nuestro enfoque MDE propone una separación de aspectos y el uso de modelos de contexto genéricos que se pueden reutilizar en distintas aplicaciones y frameworks. Nuestra propuesta va acompañada de un método para guiar a los desarrolladores en la tarea de modelar el contexto y la calidad de contexto para aplicaciones context-aware.

Agradecimientos

Casi no puedo creerme que haya llegado este momento con el que tantas veces he soñado. Son muchos años de trabajo y de esfuerzo, condensados en un sólo documento y sin duda no habría podido seguir adelante sin el apoyo de varias personas a las que quiero mostrar mi agradecimiento.

En primer lugar quiero agradecer a mis padres Ramón y Conchita por haber velado para que recibiera una formación adecuada y por educarme en valores que me han sido muy útiles a lo largo de mi vida y que, sin duda, me seguirán siendo útiles en el futuro. Ellos me animaron a estudiar el doctorado, especialmente mi padre que, por desgracia, no podrá ver cómo he cumplido este sueño pero que seguro que se sentiría muy orgulloso.

También quiero darle las gracias a mi mujer Gema y a mi hija Inés que pronto cumplirá tres años. A la primera por iluminarme la vida y hacer que este mundo parezca maravilloso, y a la segunda por darle sentido a mi existencia. Ambas me han apoyado mucho durante todo este tiempo. Gema con su paciencia y comprensión y dándome muestras de ánimo llegando al punto incluso de leerse parte de este tesis. Inés, con su sonrisa y sus muestras de cariño y también durmiéndose a veces durante largos periodos de tiempo para permitir que me concentrara en el trabajo. Tengo que reconocer que me encanta cuando me achucha y me da besitos. Muchas gracias a ambas por quererme tanto.

Además quiero agradecer el apoyo de mis hermanos Javier, Carlos, Chiti y Maika, de todos mis sobrinos y de mi tita Carmen, catedrática de Estudios Árabes e Islámicos de la Universidad de Valencia a quien siempre he tenido como modelo.

A mis directores de tesis Jesús García Molina y Juan Antonio Botía. Juan Antonio me ha estado apoyando a pesar de la distancia geográfica que nos separaba. Jesús me ha hecho comprender que las cosas siempre se pueden mejorar. Ambos me han ayudado mucho con su guía y sus consejos para mejorar la calidad de este trabajo.

No puedo olvidarme de las personas que hicieron de mi estancia en Bélgica un recuerdo inolvidable, Davy Preuveneers y Yolande Berbers. Davy me apoyó en lo profesional y en lo personal, y Yolande me recibió en su grupo de investigación y me ayudó para convertir mi estancia en una experiencia acogedora mientras me encontraba muy lejos de casa y de mi familia.

Finalmente a algunos compañeros de trabajo que también me han dado ánimos y hacen que se lleve mejor el día a día: a Mercedes Galán, a José Luis Fernández y a Javi Bermúdez que ha sufrido como yo durante todo este tiempo y a quien le deseo buena suerte con su tesis.

Abstract

The term *Ubiquitous Computing* refers to computers which can be everywhere but are designed so as not disturb users and facilitate interaction in a natural way, thus allowing the integration of computing devices into the user's everyday life.

In order to provide users with an adequate servic, these systems should be aware of their context. *Context-aware Computing* characterizes a computing model in which users interact with many different computers and the software systems are able to sense their environment and automatically adapt their behavior to the changes occurring. The environment is characterized by context information that includes data such as the current location and time, user activities and device parameters. A key element in a context-aware architecture is therefore the component that manages the context information, which is in charge of representing and manipulating such information.

Most context-aware systems typically assume that the context information upon which they rely is complete and accurate. However, the information used by them is often imperfect. Sensors suffer from technical limitations, signifying that their sensed data will only be an approximation to real values and other factors such as network disruptions may also affect the context information. A wrong assumption about context information can lead to incorrect behavior by context-aware systems.

Context-aware applications must clearly be developed by bearing in mind the problems inherent to the acquisition of reliable context information, the representation of context and the adaptation mechanisms used to obtain a correct system behavior. A systematic approach based on the use of sound software engineering principles, techniques and methods is therefore be needed to deal with the complexity of context-aware systems.

Models help developers to reason and make predictions about problems and their solutions, and can be used as system specifications. There are several context modeling approaches with which to represent context information but they have some limitations (e.g. they were developed for a specific framework, have a low level of abstraction or lack an adequate representation in the case of some context elements). *Model-Driven Engineering* (MDE) promotes the systematic use of models in order to raise the level of abstraction at which software is specified, and to increase the level of automation in the development of software. MDE uses models to develop systems and models transformations to obtain target system's software artifacts.

This thesis concerns the development of an MDE approach that defines a generative architecture for context-aware middlewares. This architecture provides a *Domain-Specific Language* (DSL) named MLContext which can be used to represent the context information, takes into account the quality of context, and automatically generates software artifacts related to context management from the models created by means of the DSL.

The MLContext abstract syntax has been defined as a metamodel, and model-to text transformations have been written in order to generate the desired software artifacts. The concrete syntax has been defined with the EMFText tool, which generates an editor and a model injector. MLContext has been designed so as to provide a high-level abstraction, to be easy to learn, and to promote the reuse of context models. A domain analysis has been applied in order to elicit the requirements and design choices to be taken into account when creating the DSL. As a proof of concept of the proposal, the generative approach has been applied to several middleware platforms for the purpose of context management.

Acknowledgements

I can hardly believe I've reached this point of which I have dreamed so often. It has been many years of work and effort, condensed into one document and I would certainly not have been able to continue without the support of several people to whom I wish to express my gratitude.

First I want to thank my parents Ramon and Conchita for having ensured I received an adequate training and for educating me in moral values and ethics. These values have been very useful to me throughout my life and will without any doubt remain useful in the future. My parents encouraged me to study the PhD, especially my father who, unfortunately, can't see how I have fulfilled this dream, but I'm sure he would be very proud of me.

I also want to thank my wife Gema, and my daughter Inés whose birthday is coming up soon and she will be a three years old. The first, for enlightening my life and making this world seem wonderful, and the second, for bringing sense to my life. Both have been very supportive during this time. Gema with her patience and understanding, giving me encouragement and even reading part of this thesis. Inés with her smile and her expressions of affection and sometimes falling asleep for long periods of time, to allow me concentrate on my work. I have to admit I love it when she cuddles me and gives me kisses. Thank you very much to both for loving me so much.

I also want to thank the following people for their support: my brothers and sisters Javier, Carlos, Chiti and Maika, all my nephews and nieces, and my aunt Carmen, professor of Arabic and Islamic studies at the University of Valencia. She has always been a role model for me.

To my thesis advisors Jesús Garcia-Molina and Juan Antonio Botía. Juan Antonio has supported me despite the geographical distance between us. Jesús has made me realize that things can always be improved. Both helped me a lot with their guidance and advice to improve the quality of this work.

I must not forget to mention the people who made my stay at Belgium a welcoming experience, Davy Preuveneers and Yolande Berbers. Davy supported me professionally and personally, and Yolande took me in her research group and helped me to make my stay a cozy experience when I was far away from my home and family.

Finally I would like to thank some colleagues who have given me encouragement and make me feel better during the everyday routine: Mercedes Galán, José Luis Fernández and Javi Bermúdez, who has "suffered" like me all this time and I wish him "good luck" with his thesis work.

Contents

List of figures

List of Tables

1 Introduction

A preliminary part

The tragedy of life doesn't lie in not reaching your goal.
The tragedy lies in having no goals to reach.

Benjamin Mays

The term *Ubiquitous Computing* was coined by Mark Weiser in 1988 (Weiser, 1988) at the Computer Science Lab at Xerox PARC to refer to "computers so imbedded, so fitting, so natural, that we use them without even thinking about them". He presented this new computing paradigm in the seminal paper "The computer for the twenty-first century" (Weiser, 1991). In his vision, the continuous increase in computer power and the advances in Internet would allow the integration of computing devices into the everyday life of people (Weisser et al., 1997). Computers would be everywhere but they would be designed so as not to disturb the users and facilitate interaction in a natural way. Ubiquitous Computing has now become widespread and its presence is currently increasing in everyday life. The term *Pervasive Computing* is also used to refer to Ubiquitous Computing. Moreover, some particular manifestations of this paradigm have emerged over the years such as *Ambient Intelligence* (Aarts et al. 2002) or *Everyware* (Greenfield, 2006).

The development of *pervasive systems*[1] involves several areas of Computer Science, mainly related to *Software Engineering*, *Computer Networks*, *Human-Computer Interaction* and *Artificial Intelligence*. *Pervasive computing* is also based on other computing paradigms such as *Context-aware Computing*, *Mobile Computing* and technologies such as sensors and networks. The work presented in this thesis is focused on the aspects of software engineering and the modeling of context information in particular.

In order to provide users with an adequate service, *pervasive systems* should be aware of their context. *Context-aware Computing* was firstly introduced by Schilit and Theimer (Schilit et al., 1994) to characterize a computing model in which users interact with many different mobile and stationary computers and the software systems are able to sense their environment and automatically adapt their behavior to the changes occurring. The environment is characterized by context information that includes data such as the current location and time, user activities and profiles, and device parameters (Baldauf et al., 2007). A key element in a context-aware architecture is therefore the component that manages the context information, which is in charge of representing and manipulating such information.

According to (Lee et al., 2011), context-aware systems have evolved in three generations (see Figure 1.1):

- **First Generation**. The first generation of context-aware systems focused on applications from domains that mainly used location information as context information (Want et al., 1992), although other context information such as time, temperature and season were also considered over the years (Ryan et al., 1997; Brown et al., 1997). These systems do not use context models, in the strict sense, but rather tables in which to store the context values or a compact internal format in which to store the context information.
- **Second Generation**. These systems manage more types of context information, which are acquired from not only sensors but also other

[1] This term will be used to refer to those systems based on ubiquitous computing

external sources. Some frameworks, such as the Context Toolkit (Salber et al, 1999) are created to facilitate the development of context-aware applications. Simple context models started to be used in this generation such as the attribute-value model (Salber et al, 1999), mark-up schemes (Ryan, 1999) or graphical models (McFadden et al., 2004).

- **Third Generation**. In addition to the data obtained from sensors and other sources, higher level context information can also be inferred by reasoning about them. Many of the context-aware systems adopt ontologies as their context-model. These systems focus on achieving scalability, performance (Ejigu et al., 2007; Gu et al., 2005) and the protection of privacy (Hong et al., 2004).

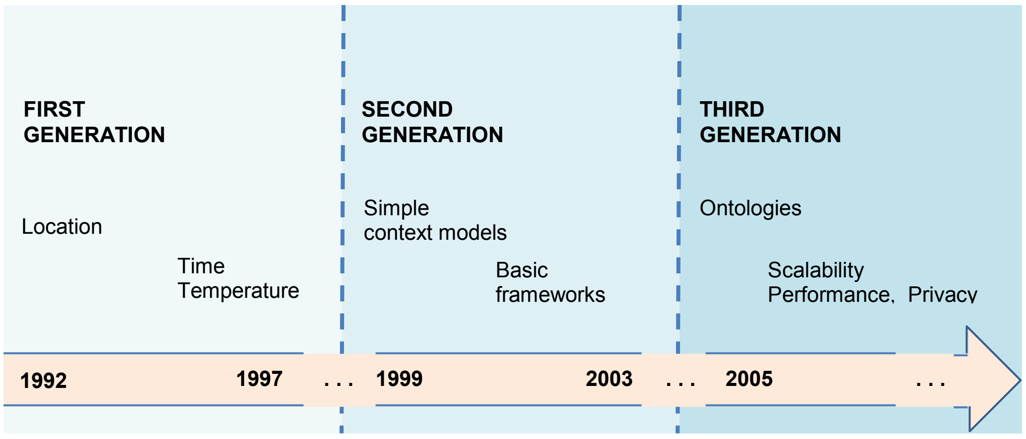


Figure 1.1 Generations of context-aware systems.

Regardless of the generation to which they belong, most context-aware systems typically assume that the context information upon which they rely is complete and accurate. However, the information that they use to recognize different contexts is often imperfect. Sensors have technical limitations, signifying that their sensed data will only be an approximation to real values and other factors such as network disruptions or sensors failures may also affect the context information. Users' actions can contribute to the degradation of information quality, such as the failure of users to carry their locator tags. A wrong assumption about context information may therefore lead to incorrect behavior by context-aware systems (Henricksen et al., 2004), and the lack of information about the quality of context (QoC) can result in the degraded performance of context-aware applications in pervasive environments without the actual problem being known because it is impossible to resolve uncertain and conflicting situations regardingt context information (Manzoor et al., 2011).

The QoC notion was proposed in (Buchholz et al., 2003) as an essential aspect to be taken into account in context-aware software. QoC was there defined as the information that describes the quality of context information. A great deal of attention has since been paid to QoC and several works have presented proposals with which to represent, measure and evaluate it (Wang et al., 1996; Knight et al., 2005; Helfert et al., 2013), but there are few context managers with QoC support, as is noted in (Chabridon et al., 2013). Some research attempts have been made to model and measure the quality of raw context information sensed from the environment. However, to date little attention has been paid to the quality evaluation of derived and inferred context information (Filho et al.,2010) and this will be a critical issue for the next generation of context-aware systems.

All of the above suggests that the construction of context-aware systems is a difficult task. Context-aware applications must clearly be developed by bearing in mind the problems inherent to the acquisition of reliable context information, the representation of context and the adaptation mechanisms used to obtain a correct system behavior. A systematic approach based on the use of sound software engineering principles, techniques and methods is therefore needed to deal with this complexity.

Because context management (i.e. storing and processing) plays a key role in context-aware systems, the creation of context models assists developers as regards understanding and reasoning about context. A number of approaches for context modeling have been proposed in the last decade, all of which differ in the data structures used to represent the context information (Schilit et al., 1994b; Held et al., 2002; Preuveneers et al., 2005). Moreover, several context managers have been built (Bardram, 2005; Nieto et al., 2006; Riva, 2006]. However, the mastering of complexity in managing context information is still a great challenge and there are few tools and models that actually help developers as is noted in (Chabridon et al., 2013).

Model-driven engineering (MDE) is increasingly gaining acceptance, principally owing to its ability to tackle software complexity and improve software productivity (Selic, 2012; Whittle et al., 2014). MDE promotes the systematic use of models in order to raise the level of abstraction at which software is specified, and to increase the level of automation in the development of software. As in other application domains, the activity of modeling in context-aware systems is currently directed towards MDE. Some approaches have recently been proposed to take advantage of MDE techniques in the construction of context-aware pervasive applications (Ayed et al., 2007; Sindico et al., 2009; Chabridon et al., 2013), but a great research effort is still needed to provide appropriate languages, methods and tools that will efficiently support the model-based development of this kind of applications. This thesis therefore concerns the development of an MDE approach that defines a generative architecture for context-aware middlewares. This architecture provides a domain-specific language (DSL) with which to represent the context information, which takes into account the quality of context, and automatically generates software artifacts related to the context management from the models created by means of the DSL.

The remainder of this chapter is organized as follows: Section 1.1 explains the motivation for this work. Section 1.2 introduces the problem that the present thesis resolves and details the thesis goals. Section 1.3 describes the research methodology followed in this thesis. Finally, Section 1.4 provides an overview of the overall structure of this document.

1.1 Motivation

The design and implementation of context-aware systems is a complex task (Chen et al., 2000; Gu et al., 2005) and is still challenging. Some of the reasons for this are the lack of consensus as regards a conceptual framework for context-aware systems and the fact that there has been no scientific approach to context as an object of study (Gu et al., 2005). It is currently difficult to design, develop and maintain this kind of systems because of the aspects related to context management.

Wei and Chan (Wei et al., 2007) identified four aspects to be considered by context-aware system developers when tackling the complexity inherent to such systems: what is context, how to capture it, how it can be represented and how applications can adapt their behavior according to it. In the case of the last two tasks, an adequate representation of context can be determinant for the system performance, but most of the current approaches used to develop context-aware systems employ ad-hoc solutions in spite of modeling context and do not usually take into account any software engineering methods, focusing rather on solving technological problems. Moreover, manual development is error prone and the resulting application is hard to maintain.

Models help developers to reason and make predictions about problems and their solutions, and can be used as system specifications (Selic, 2012). They also improve communications between different stakeholders, and several industrial experiences (Bone et al., 2010; Hutchinson et al., 2011) have proved that models can improve the quality of the software and productivity during the development of systems. Several formalisms with which to represent context models has been proposed to date, such as key-value models like that in (Schilit et al., 1994b) or the Context Toolkit (Dey, 2000), graphical models based on a graphical notation (e.g. UML) like ContextUML (Sheng et al., 2005) or ontology- based models like the Context Broker Architecture (CoBrA) (Chen et al., 2004).

In what follows we shall describe some of the main limitations of existing context modeling approaches. These limitations will be discussed in more detail in Chapter 3, in which we present a review of the state of the art in context modeling. Most of the proposed approaches were developed for a specific framework, and their abstraction is too low for them to be processed in an efficient way. This has some limitations for developers, making the process of developing more complex, because it is more difficult to create, use and maintain context models, relative to a high-level of abstraction. Managed concepts are far from the domain concepts, and developers must deal with the particular aspects of context representation for specific frameworks. In addition, the low-level of abstraction of the models makes the interaction between the developer and the user difficult, especially when validating the context model. Moreover, in a heterogeneous environment, in which different context-aware frameworks share the same context, developers may need to create different context models representing the same context information, one for each specific framework, rather than creating only one.

The Unified Modeling Language (UML) is a modeling standard that has been approved by ISO in 2005, and which is widely used in academic and industrial communities. It is a general purpose modeling language, and several context modeling approaches have been based on it. These approaches (based on UML or UML profiles), are restricted to the definition of context modeling languages whose abstract syntax and semantics is close to those provided by UML (Selic, 2007). Since UML profiles are not a first-class extension mechanism and do not allow the UML metamodel to be modified, the new elements which specialize existing elements must not violate the abstract syntax rules and semantics of UML. DSLs created from scratch are currently considered to be the most appropriate in most situations (Kelly et al., 2009; Völter, 2009).

The third generation of context-aware systems started to use ontologies as context models. Models based on ontologies are not easy to handle, because the building and management of ontologies is a complicated task which often involves the design and implementation of complex algorithms (Khattak et al., 2009). Moreover, the creation of ontology management systems is still at an early stage (Maedche et al, 2003; Gómez-Romero et al., 2011).

Some context modeling approaches are incomplete and lack adequate representation as regards some elements such as sources of context or entities (Sheng et al., 2005; Nieto et al., 2006). Others are more oriented toward representing services than context (Serral et al., 2008), or their context information is scattered throughout the model (Sheng et al., 2005; Conan et al., 2007), thus making processing difficult. Most of the approaches do not distinguish between different types of context and few approaches make use of MDE techniques. Most proposals do not therefore allow developers to generate code from the models. Finally, it is worth noting that current approaches do not take into account the separation of concerns and context models normally include information that is specific to a particular application and these models cannot consequently be reused in other applications scenarios.

With regard to the quality of context, existing approaches do not normally consider QoC as a crucial issue in the performance of context-aware systems (Manzoor et al., 2011) and very few frameworks support QoC. Most of the proposals do not therefore solve the problem of modeling QoC or expressing quality requirements. We have included QoC representation in our work because it is an intrinsic aspect of context information. Other aspects such as context reasoning or the response of the system to certain context situations (adaptation) are out of the scope of this thesis.

One example of the extent to which QoC can be a critical factor for context-aware systems is the Iranian Airbus disaster (Fisher et al., 2001). On July 3rd 1988, the U.S. Navy Cruiser USS Vincennes accidentally shot down an Iranian commercial aircraft with 290 passengers onboard, and data-quality problems were a major factor in the decision made to attack a civilian aircraft. Some of the flaws resulted from accuracy, completeness and trustworthiness issues, made manifest in the use of wrong target identifiers, conflicting information, and the detection of an inaccurate altitude of the aircraft. An adequate model of context and, in particular, the modeling of QoC requirements could help prevent such failures. There are currently a lack of solutions for representing QoC and QoC requirements. In addition, existing definitions of QoC overlook its multifaceted nature (in the sense that it is necessary to evaluate several parameters, some of which are dependent on others) and consider it to be an objective term (Manzoor et al., 2011).

As indicated above, MDE is a software paradigm in which models play a key role. The main idea of MDE is to use models at different levels of abstraction to develop systems and models transformations to obtain the target system's software artifacts (e.g. source code or XMl files) from models. An MDE approach would appear to be a good choice as regards dealing with the aforementioned limitations of current context modeling approaches. Context information, including QoC could thus be represented at a high level of abstraction, and context models could easily be understood by the user. These context models should be platform-independent, i.e. they should not contain details related to any specific framework, which favors portability. They would be specified by using a DSL, which should capture the essential concepts of the context domain. This would make context specifications shorter and more legible and would allow developers to ignore technology-specific details (e.g. programming languages or context-aware frameworks).

Transformation engines could then be used to automatically obtain software artifacts for different context-aware middlewares, which favors interoperability.

Another benefit is an increased adaptability to changes in the specification (e.g. to change one type of sensor for other). As changes are located at a certain abstraction level their code can be generated again and, by applying a separation of concerns, those parts that are not affected can be reused. The automatic generation of code also makes the development process of context-aware systems less error prone.

1.2 Problem statement and thesis goals

As evidenced in the above discussion, the development of context-aware systems is a complex task in which context management plays a key role. One of the main problems that the developers of these systems must confront is the creation and use of an adequate context representation which allows the specification of both, the environment and all other aspects of interest needed for a correct system behavior adaptation. How the context information is currently modeled could be improved to achieve a larger quality and productivity in the development of context-aware applications. Moreover, a little attention has been paid to the problem of modeling QoC, which is a critical issue if context-aware applications are to perform correctly in pervasive environments. In this thesis, we have attempted to overcome the limitations identified above by defining an MDE approach in order to create a generative architecture for context-aware systems, which focuses on the context modeling aspects and takes into account the quality of context. We also propose a method for context and QoC modeling.

The statement of the problem therefore allows us to infer the following objectives of this thesis:

- **To create a Domain-Specific Language for context modeling with a high-level of abstraction**. A DSL with a high-level of abstraction (and that is platform-independent) has several benefits for developers when modeling context. On the one hand, they do not need to deal with implementation or specific framework details. On the other hand, a DSL is easier to learn and also to use because managed concepts are near to the domain concepts, which makes the process of developing simpler and more understandable that in the case of a low-level of abstraction. This DSL should promote the writing of shorter context specifications, which will increase productivity. Finally, it favors the interaction between the developer and the user particularly when validating the context model. The design of the DSL should be preceded by a domain analysis in order to capture the most significant aspects related to the context modeling from the domain - through the definition of its vocabulary and key concepts -. It is worth noting that, to the best of our knowledge, no domain analysis of this type has ever been carried out for any context modeling language.

- **To define a set of constructors for the DSL in order to specify the quality of the context information and the quality requirements**. The lack of information about the QoC may result in the degraded performance of context-aware applications. QoC information can help resolve uncertain and conflicting situations in context-aware applications. A DSL for context modeling must have some constructors for QoC modeling at the sensor level and some mechanisms with which to represent the QoC of derived context information. It should also allow the developer to specify quality levels and quality requirements for the context information.
- **To automatically generate software artifacts for context-aware middlewares**. The DSL created to model context information will be the core element of an MDE generative architecture that is able to automatically generate software artifacts related to the context management in context-aware middlewares (e.g. ontologies, Java code, or XML files). A DSL engine (i.e. a model transformation chain) would be in charge of transforming the models expressed with the DSL into software artifacts. The approach should be platform-independent (i.e. the context model could be reused in different frameworks), and should promote the separation of concerns in order to favor adaptability to changes in the requirements. This separation of concerns could also favor an application independence (i.e. the context model could be reused for different applications within the same framework).
- **To define a method for context and QoC modeling**. The MDE approach defined in this thesis will be accompanied by a proposal for a simple method with which to model context and QoC in the earlier stages of a context-aware application development process. It is based on the particular characteristics of our approach and recommends the steps to be followed in context modeling, thus

allowing developers to take full advantage of the specificities (models and tooling involved) of our proposal.

1.3 Research methodology

In order to achieve the objectives of this thesis that were introduced in the section above, we have followed the design science research methodology (DSRM) described in (Peffers et al., 2008; Vaishnavi et al, 2013). The design process consists of six activities (see Figure 1.2): (1) Problem identification and motivation, (2) Define the objectives of a solution, (3) Design and development, (4) Demonstration, (5) Evaluation, and (6) Conclusions and communication.

This is an iterative process in which the knowledge produced throughout the process by constructing and evaluating new artifacts served as feedback for a better design and implementation of the final solution.

Following the activities defined in DSRM, we began by identifying the problem and our motivation. In the previous section, we have identified the problem to be resolved and we have stated it clearly: to create a MDE approach to overcome some of the limitations of current approaches for context and QoC modeling in the development of context-aware systems. This activity requires knowledge of the state of the problem and the consequences (i.e. benefits and drawbacks) of its solution, which will be discussed in Chapter 3.

We then performed the second activity which involved devising a solution to the problem and identifying the objectives to achieve in order to build the solution. This was done by studying the most relevant approaches from similar application domains in detail (see Chapter 3). Our solution will be based on MDE because it provides the foundations needed to explicitly represent the context information by means of models and to automate the generation of software artifacts from these models by means of model transformations. The context and the QoC information will be represented by models which are described by metamodels, and the mapping between a metamodel and a concrete context-aware framework will be defined by a model transformation.

The third activity was organized into two main steps. Firstly, we tackled the design and implementation of a DSL with which to model context information for context-aware systems. Secondly, we extended the DSL with QoC constructs. In this second step, we defined the strategy to be recommended to developers in order to model context. The construction of a full-fledged MDE solution involved the creation of several tools such: (1) an editor with which to create DSL specifications, (2) an injector with which to convert DSL specifications into a model (which conforms to the metamodel that represents the abstract syntax of the DSL), and (3) a transformation engine that implements the mapping between the abstract syntax of the DSL and a particular context-aware framework by means of a model-to-text transformation. These tools have been implemented by means of tools based on the Eclipse Modeling Framework (EMF) and have been integrated into the Eclipse platform.

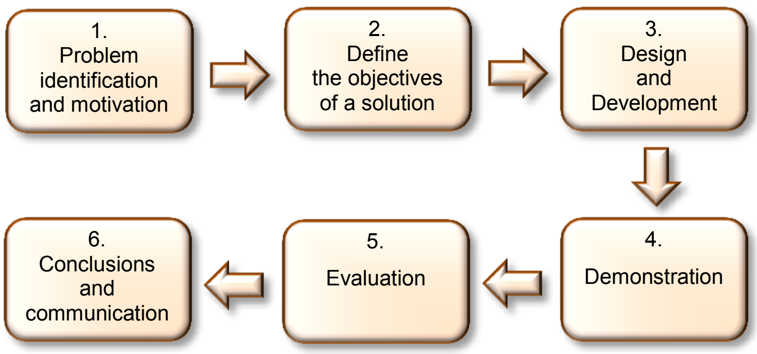

1. Problem identification and motivation
2. Define the objectives of a solution
3. Design and Development
4. Demonstration
5. Evaluation
6. Conclusions and communication

Figure 1.2 Research metodology.

In the fourth activity, we demonstrated that the solution works by solving several instances of the problem for different context-aware middlewares. We have used two different case studies to show the capabilities of our approach as regards modeling context and generating code for several middlewares with and without QoC support. A case study based on a Hospital environment was used when the QoC support had not yet been defined, and a flash-sales scenario was later used to show how QoC could be managed. In the fifth activity we then evaluated the solution by comparing the objectives with the results obtained. We have validated our approach in terms of easiness of use, expressiveness and productivity.

Finally, some improvements and future research lines were identified in the conclusions. We disseminated the results obtained from our work in several research articles published in journals and conferences. In these articles, we show the novelty and contributions of our work. The most relevant publications and presentations at conferences, along with a discussion regarding the dissemination of the contents and the results presented in this thesis, will be shown in Chapter 7.

1.4 Outline of this thesis

This thesis is presented in six chapters, including this one, and one appendix. A guide to the organization of this thesis is provided as follows:

Chapter 1. This chapter motivates the thesis work by introducing the problem being confronted. It also briefly describes the contents of the thesis and the main objectives.

Chapter 2 introduces the main fields that are related to the work presented in this thesis in order to provide the reader with a basic background in order to understand the overall thesis work.

Chapter 3 presents projects and initiatives that confront similar problems or which in some way provide solutions to some aspects of this work. The projects are organized in several sections owing to their scope. These initiatives are analyzed and compared to the work that is presented in this thesis.

Chapter 4 provides an overview of the thesis work and the implementation. The chapter also explains how our approach has been evaluated from several dimensions.

Chapter 5 explains the development of the MLContext DSL in detail, from the design phase to the validation of the proposal.

Chapter 6 presents an extension of MLContext whose purpose is to model the quality of context aspects. This extension is also validated by means of a case study.

Chapter 7 summarizes the main contributions and publications of this thesis and cites to the contributions. This chapter also provides some insights into further work.

2 Background

A must know to understand this work

There are 10 types of people in the world:
those who understand binary, and those who don′t.

Unknown Author

Computer-aware computing is a very active research field whose challenges and research questions are evolving continuously. Building context-aware systems involves aspects from different computer science areas, and researchers in these areas are making efforts to understand and improve concepts, technologies and applications for this field.

The objective of this chapter is to provide the knowledge needed to better understand this thesis. We shall introduce some basic concepts and techniques of context-aware systems and Model-Driven Engineering, in addition to the MDE tools used in the implementation of the approach devised in our work.

With regard to context representation, we shall explain some basic notions which are essential in our work as regards context (Section 2.1) and quality of context (Section 2.3). We shall also show the different types of context representation (Section 2.2). We shall additionally introduce the foundations of Model-Driven Engineering (Section 2.4), in particular metamodeling, domain-specific languages and model transformations. Finally, we shall show the tooling used to implement our approach: EMF and Ecore (Section 2.5), EMFText (Section 2.6) and MOFScript (Section 2.7).

2.1 What is context?

As indicated in Chapter 1, context-aware systems should be aware of their context to provide adequate services. These systems are able to automatically adapt their behavior based on the current context without user intervention. Therefore, the storage and processing of context are critical issues for context-aware systems which need an internal representation of context (i.e. a context model) to work properly.

Previously to present the most widely used formats to express context models, we shall precisely define the concept of context. Context has often a significant impact on the way humans or machines act and on how they interpret things (Bolchini et al., 2007). The word "context" derive from the Latin con (with or together) and texere (to weave) which describe a context as "an active process dealing with the way humans weave their experience within their whole environment to give it meaning".

The term context is poorly understood by the computer science community which has overloaded it with a wide variety of meanings (Henricksen et al., 2002). There are many definitions of context in the literature but all of them offer few clues of the properties that are of interest in the field of context-aware systems. Table 2.1 shows some of these definitions ordered by authors.

Research on context-aware systems in the nineties focused on the theoretical and conceptual foundations of these systems and the work carried out was so relevant that the main context definitions belong to this period (Schilit et al., 1994; Dey et al., 2000).

Table 2.1 Definitions of context.

Author	Definition of context
Brezillon, 1999	"A set of knowledge pieces related to a particular activity or situation."
Brezillon et al., 2004	“Whatever does not intervene explicitly in a problem solving but constrains it.”
Brown, 1996	"The elements of the user’s environment which the computer knows about."
Chen et al., 2004	"By context, we mean the situational conditions that are associated with a user: location, room temperature, lighting conditions, noise level, social activities, user intentions, user beliefs, user roles, personal information, etc."
Dey, 2000	"Context is any information that can be used to characterize the situation of an entity (i.e. a person, computing device, or non-computational physical object)."
Dey, 2001	"Context is the location, identity and state of people, groups and computational and physical objects."
Dey et al., 2000	"Any information that can be used to characterize the situation of any entity, where an entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and the application themselves.
Fischer, 2012	"Context is the 'right' information, at the 'right' time, in the 'right' place, in the 'right' way to the 'right' person"
Göker et al, 2002	"A context can be defined as a description of aspects of a situation. In this way, context can seem similar to cases in case-based reasoning. A context as an internal representation in the computer should be a structure for information units and data."
Kofod-Petersen et al., 2005	“Context is the set of suitable environmental states and settings concerning a user, which are relevant for a situation sensitive application in the process of adapting the services and information offered to the user.”
Lenat, 1995	"A context or ‘micro-theory' is viewed as a set of assertions, representing a particular set of surrounding circumstances, relevant facts, if-then rules, and background assumptions."
Moran et al., 2001	"Context refers to the physical and social situation in which computational devices are embedded"
Rittenbruch, 2002	"A complex description of shared knowledge within which an action or event occurs”
Schilit et al., 1994	"Location, identities of nearby people, objects and changes to those objects"

However there is a lack of consensus on the delimitation of the field of study and on the use of a common model to represent the context information. While the computer science community initially perceived the context as a matter of user and objects location (Schilit et al., 1994), over the last years this notion has been evolved and context is considered as part of a process in which the users are involved (Coutaz et al., 2005).

Some definitions of context are limited to certain types of information (Schilit et al., 1994; Moran et al., 2001). Others relate context to the user and his/her environment (Brown, 1996; Chen et al., 2004; Kofod-Petersen et al., 2005) because they are user-centric approaches, but context should not be limited to the user because other objects may also be considered. Finally, most approaches relate context to "some situation or surrounding circumstance", but none except the one from Dey et al. (Dey et al., 2000) introduces the notion of entity, which is "a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and the application themselves". This definition is the most widely accepted up to date. From these definitions we can conclude that context is a set of information pieces which can characterize some situation that is of interest for the system, and these information pieces are related to entities that the system must know.

There has been significant progress at technological level in the development of context-aware systems but it has not been the case in the theoretical foundations. Evidence of this is that, to the best of our knowledge, no author has proposed a new definition of context (which gives new insights) after the one proposed by Dey (Dey et al., 2000).

In the following section, we will discuss the current approaches to represent the context information.

2.2 Context model representation approaches

Since the first generation of context-aware systems, various formalisms have been proposed to represent the context information. A survey of the most relevant context modeling approaches was made by Strang and Linnhoff-Popien (Strang et al., 2004). They identified the following types of representation.

Key-value pairs

This is the simplest structure for modeling context information. Context information is defined as a collection of key-value pairs, like the way environment variables are defined for some operating systems. For example, Schilit et al. (Schilit et al., 1994b) used key-value pairs to model the context by using an environment variable to provide the value of a piece of context information (e.g. location information) to an application. Figure 2.1 shows an example of key-value model where keys are typed in bold face. The value "in room 35-2200" is assigned to the key "Location".

Date and time	`after April 15`
	`between 10 and 12noon`

Location	`in room 35-2200`

Co-location	`with {User Adams}`
	`with {Type Display}`
	`having {Features Color}.`

Figure 2.1 Example of key-value model.

While key-value pairs are easy to manage, they lack capabilities for enabling efficient context retrieval algorithms due to its simple structure. They are also error prone because they are not type-safe. It is worth noting that this representation is used by some NoSQL database systems (Sadalage et al., 2012).

Mark-up schemes

Context information is defined by using a mark-up language. These models use a hierarchical data structure formed of mark-up tags with content and attributes. The content is usually defined recursively by using other mark-up tags. Figure 2.2 shows an excerpt of an XML model, which represents a building "Empire Hotel". This building is formed of floors and each floor has several rooms. Each room contains the number of the room and the name of the guest. Mark-up schemes are usually based on the Standard Generic Mark-up Language (SGML) which is the superclass of all mark-up languages such as XML. In practice, these models are often based on XML and XML schemas or DTDs (see for example CC/PP (Ryan, 1999)), but some approaches use other languages, for instance CSCP (Buchholz et al., 2004) which uses RDFS, an extension of RDF (Resource Description Framework).

```
<?xml version="1.0" encoding="UTF-8"?>
<building name="Empire Hotel">
  <floor>
          <number> 1 </number>
          <room>
                    <number>101</number>
                    <name>Mr. Martin Flowler</name>
          </room>
          <room>
                    <number>102</number>
                    <name>Mr. P.J. Saladage</name>
          </room>
  </floor>
  <floor>
          <number> 2 </number>
          <room>
                    <number>201>/number>
```

Figure 2.2 An excerpt of an XML model.

Mark-up languages are architecture independent languages suitable for structured information representation that can be used to exchange and store data, but due to its verbosity, the amount of information to be stored it is increased. One of the main advantages is that the context represented is searchable.

Graphical Models

Graphical modeling languages can be used to model context. A well-known and general-purpose modeling language is the Unified Modeling Language (UML) (OMG, 2000). UML class models are especially adequate to model information since they provides the basic elements (classes and relationships between classes) to apply information conceptual modeling. Figure 2.3 shows an example of an UML class model which represents a User and a Shop, each one with some properties like the name of the user or the location of the shop. The user has a Smartphone, which contains the user's preferences. UML also provides the Profiles mechanism to create domain-specific UML extensions. Other examples are ContextUML (Sheng et al., 2005) which is a modeling language for the model-driven development of context-aware Web services inspired on UML, and CML (Henricksen et al., 2004b), which is an extension to the Object-Role Modeling (ORM) (Halpin, 2010). As stated in (Strang et al., 2004) this kind of approach is particularly applicable to derive an ER-model from it.

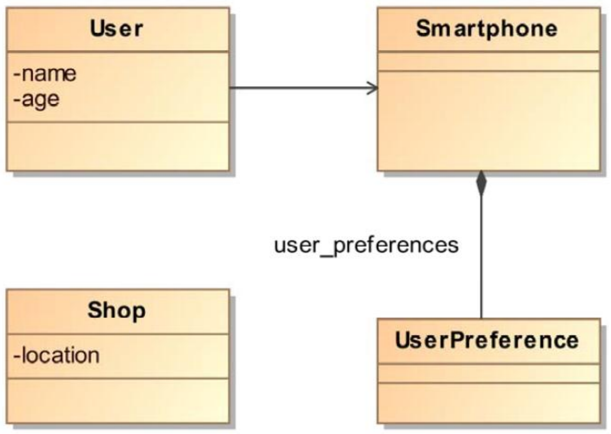


Figure 2.3 Example of an UML graphical context model.

One drawback of this kind of models is that they become unclear and difficult to read if there are a large number of objects and relationships to be represented.

Object Oriented Models

This type of models benefits from the object-oriented concepts (classes and inheritance). The details of context information are encapsulated on an object, and they provide specified interfaces to access to the context information. Two examples for this approach are the Active Object Model of the GUIDE project (Cheverst et al., 1998), where all data collections are encapsulated within the active objects, and Hydrogen (Hofer et al., 2002), which has a context acquisition approach specialized for mobile devices.

These models normally are created by means of an object-oriented programming language (directly or by using a predefined API). They do not have an own visual representation but usually makes use of a graphical object-oriented model instead. UML is an object-oriented graphical modeling language, so that it could be also included in this category.

Logic Based Approaches

In these context models, the context is defined as facts, rules and expressions. These terms are managed by a logic based system which allows adding, updating and removing new facts. Existing rules in the system can be used to derive new facts by means of an inference (also called reasoning) process. One of the first logic based context model was defined for McCarthy (McCarthy , 1993), which uses a basic relationships approach. Another approach is the Sensed Context Model (Gray et al., 2001), which is based on first-order predicate logic as a formal representation for context propositions. An excerpt of a logic model is show in Figure 2.4. S_1 is the set of objects that are birds. S_2 is the set of flying objects. Finally, C states that any object in S_1 must be present in S_2.

$$S_1 = [s \mid s \models \ll \mathit{bird}, \mathrm{a}, 1 \gg]$$

$$S_2 = [s \mid s \models \ll \mathit{flies}, \mathrm{a}, 1 \gg]$$

$$C = S_1 \Rightarrow S_2$$

Figure 2.4 A simple logic-based model example.

All these approaches assume that logical assertions are supposed to hold, and they do not contain a straightforward representation for any kind of meta-information, like quality attributes (Mühlhäuser, 2008). They can also be difficult to understand for an inexperienced user.

Ontology Based Models

An ontology is a description of concepts and the relationships that can exist among them. It uses a set of representational primitives which are typically classes, properties, and relationships. OWL ontologies, which are a particular case of mark-up schemes, are the most used. Reasoning techniques can be applied to ontologies. Therefore, ontologies are useful for modeling context information. One example of this kind of model is used by the OCP framework (Nieto et al., 2006). It uses an OWL-DL ontology (OWL, 2004) to describe the contextual information of the application in order to generate its knowledge base. Another example is the SOUPA ontology (Chen et al., 2005), which consists of nine ontology documents that use the OWL language. Ontology-based techniques have the possibility of representing the semantics of context information and they deserve further investigation.

Figure 2.5 shows an excerpt of an OWL-DL ontology which describes a class "Person" which has a property "name" in the String domain.

```
<?xml version="1.0"?>
  <!DOCTYPE rdf:RDF [
    <!ENTITY Example1 "http://www.modelum.es/Ontology/Example1Ontology.owl">
    <!ENTITY owl "http://www.w3.org/2002/07/owl#">
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
  ]>
  <rdf:RDF
    xmlns    ="&Example1;#"
    xml:base ="&Example1;#"
    xmlns:owl ="&owl;"
    xmlns:xsd ="&xsd;"
    xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <owl:Ontology rdf:about="">
    <rdfs:comment>Generated with MLContext 1.1</rdfs:comment>
    <rdfs:label>Example1 OCP Ontology</rdfs:label>
  </owl:Ontology>
  <owl:Class rdf:ID="Person">
  </owl:Class>
  <owl:DatatypeProperty rdf:ID="name">
    <rdfs:domain rdf:resource="#Person"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
```

Figure 2.5 Example of an OWL-DL ontology (excerpt).

2.3 Quality of context

As mentioned in Chapter 1, context information is often imperfect and a wrong assumption about its values can lead to incorrect behavior in context-aware applications. Therefore this kind of applications should be aware not only of their context but also of the quality of the context information they manage. In this section we will provide a background on the notion of quality of context for context-aware systems.

The word *quality* is commonly related to the degree of excellence of something. The Oxford dictionary defines quality as "a distinctive attribute or characteristic possessed by someone or something". The importance of this definition is that it emphasizes that quality can be measured using certain attributes or properties from entities.

Quality is a concern that has also been considered in different areas of software engineering such as quality of information (Lillrank, 2003), quality of databases (Johnson et al., 1981; Laudon, 1986) quality of service (QoS) (ITU-T, 2008) or quality of software (Kan, 2002).

As indicated in (Wang et al., 1996), quality of information (QoI) is a major dimension for evaluating the success of information systems. Throughout the years, several research works have proposed different dimensions or classifications for QoI.

Wand et al (Wand et al., 1996) classify data quality dimensions as related to internal or external views. The internal view refers to the design point of view and includes data-related quality parameters: accuracy, reliability, timeliness, completeness, currency, consistency, precision; and system-related parameters: reliability. The external view refers to the point of view of use and includes parameters related to data: timeliness, relevance, content, importance, sufficiency, usableness, usefulness, clarity, conciseness, freedom from bias, informativeness, level of detail, quantitativeness, scope, interpretability, understandability; and related to system: timeliness, flexibility, format, efficiency.

Wang et al (Wang et al., 1996) identifies a set of 179 quality attributes (e.g, corruption, cost, stability) and 20 quality dimensions (e.g., value-added, interpretability, ease of understanding, traceability, reputation, accuracy, timeliness or variety of data). Then, they grouped them into four categories from the data consumers point of view: Intrinsic (accuracy, objectivity, believability, reputation); Contextual (relevancy, value-Added, timeliness, completeness, amount of information); Representational (interpretability, format, coherence, compatibility); and Accessibility (accessibility, access security).

Kim et al (Kim et al., 2006) present quality dimensions of context information based on user's need: accuracy, completeness, representation consistency, access security and up-to-dateness.

Lachica et al. (Lachica et al., 2008) identify three main dimensions of quality which are important in information retrieval with fuzzy semantic networks: reliability (accuracy, concise, objectivity, believability, reputation, understandability), availability (security, accessibility, navigation, consistency) and relevancy (useful, efficiency, timeliness, value-added, usability, amount, completeness).

Sidi et al (Sidi et al., 2012) presents a survey of quality data dimensions and identify 40 dimensions such as: data integrity fundamentals, concise, data decay, amount of data, free-of-error, interpretability, effectiveness, believability, duplication, and presentation quality.

Quality of context (QoC) is a particular case of QoI. One of the first definitions for quality of context is "any information that describes the quality of information that is used as context information" (Buchholz et al., 2003). Although this definition was widely accepted by the context-aware community, it is not very useful since it is unclear what quality of information is. Later QoC was defined as "any inherent information that describes context information and can be used to determine the worth of information for a specific application" (Krause et al., 2005). Note that this definition shows that the way we measure the quality of the context information is specific to a particular context-aware application. This will be a determining factor when considering our MDE approach as discussed in later chapters.

From these definitions we can conclude that QoC is any information that describes context information related to some entity, which can be measured by using the values from certain attributes of that entity and can be used to determine its worthiness for a specific context-aware application.

The research work in the area of context quality focuses on two main aspects (McKeever et al., 2009): (1) the identification of context quality parameters, such as the work done by (Wang et al., 1996; Pipino et al., 2002) and (2) the use of reasoning techniques that incorporate quality of context such as Bayesian networks (Ranganathan et al., 2004) and fuzzy logic (Korpipää et al., 2003), which is beyond the scope of this thesis. The choice of quality parameters is primarily based on industrial experience or intuitive understanding but there is no general agreement on data quality parameters (also called data quality dimensions by several authors). Therefore there are several studies which include different quality parameters such as accuracy, precision, resolution, comparability or timeliness, but there is no definition of their meanings or a set of quality parameters that are commonly accepted.

In Section 6.1.1 a review of the literature will be conducted to identify the most common used quality parameters and their relationships.

2.4 Model-Driven Engineering (MDE)

Model-Driven Engineering (MDE) is the term that is commonly used to refer to the area of Software Engineering that involves the study of the model-based solutions in the development of software. Actually, MDE involves several paradigms which are based on four basic principles (Brambilla et al., 2012; García-Molina et al., 2013): (1) models are used to represent aspects of a software system at some abstraction level; (2) they are expressed using DSLs (a.k.a. modeling languages) (3) these languages are built by applying the meta-modeling technique and (4) model transformations provide automation in the software development process.

Model-Driven Architecture (MDA) (Kleppe et al., 2008) and Domain-Specific Development (Kelly et al., 2008) are two well-known MDE paradigms. The MDA initiative was launched by OMG in 2001 and has played a crucial role in the emergence of MDE as a discipline. MDA separates the functionality of an application from its implementation on a particular platform. Thus, developers create platform-independent models (PIM) which are transformed into platform-specific models (PSM), and code is finally generated from PSM models. In (Stahl et al., 2006), the concept of "generative architecture" is proposed to generalize the MDA vision. A generative architecture is composed of one or more DSLs which automate the development of applications in a platform. Developers use the DSLs to write models or specifications which are used as input to a DSL engine that generates software artifacts of the final application.

Domain-specific development and Language-oriented programming are some of the terms commonly used to refer to the creation and use of DSLs to automate any software development task. While MDA and generative architecture paradigms focus on the automated generation of applications, domain-specific development emphasizes the use of DSLs as an alternative to write code with a general-purpose language (GPL). As indicated in (Whittle et al., 2014) the creation of small DSLs for narrow domains is becoming the main manifestation of MDE in software companies. Actually, a generative architecture can be considered a particular case of domain-specific development in which DSLs aim to partially or totally automate the development of an application.

2.4.1 Metamodeling

A metamodel is a model that describes the concepts and relationships between them in a certain domain. A metamodel is commonly defined by means of an object-oriented conceptual model expressed in a metamodeling language such as Ecore (Steimberg et al., 2008) or MOF (MOF, 2006). A metamodeling language is in turn described by a model called the meta-metamodel. Metamodeling languages provide four main constructs to express metamodels: *classes* (normally referred as *metaclasses*) for representing domain concepts, *attributes* for representing properties of a domain concept, *association* relationships (*aggregations* and *references*) between pairs of classes to represent connections between domain concepts and *generalizations* between child metaclasses and their parent metaclasses for representing specialization between domain concepts. Usually, the notation for the UML class diagrams is used to represent each of the concepts and their relationships (see Figure 2.6).

We will use the term *structural feature* to refer to both attributes and relationships. Figure 2.7 shows a metamodel called *Graph* that represents a simple graph with concepts such as node and edge, which have been modeled as metaclasses. This metamodel will be used for explaining the concepts of metamodeling and the creation of a model.

The *Graph* and *Node* metaclasses have a name attribute which they inherit from a *NamedElement* metaclass. An example of aggregation is shown from the *Graph* metaclass to the *Node* metaclass. The aggregation *nodes* represents that a graph contains zero or more nodes (see the cardinality at the end of the aggregation). The example also shows two references from *Edge* to *Node*.

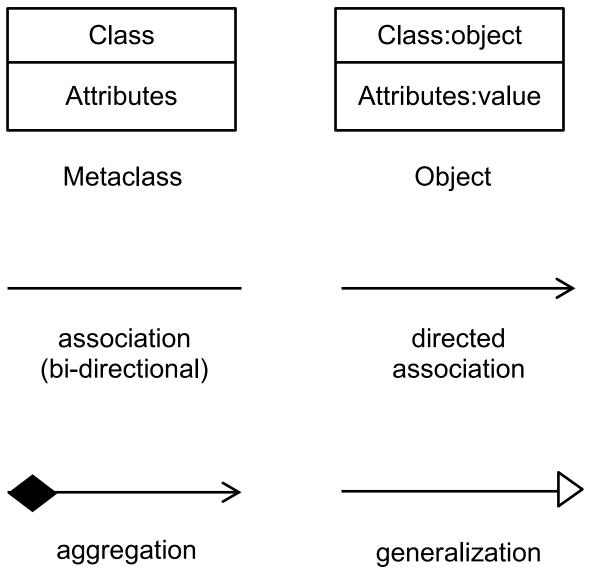


Figure 2.6 Metaclasses and relationships notation.

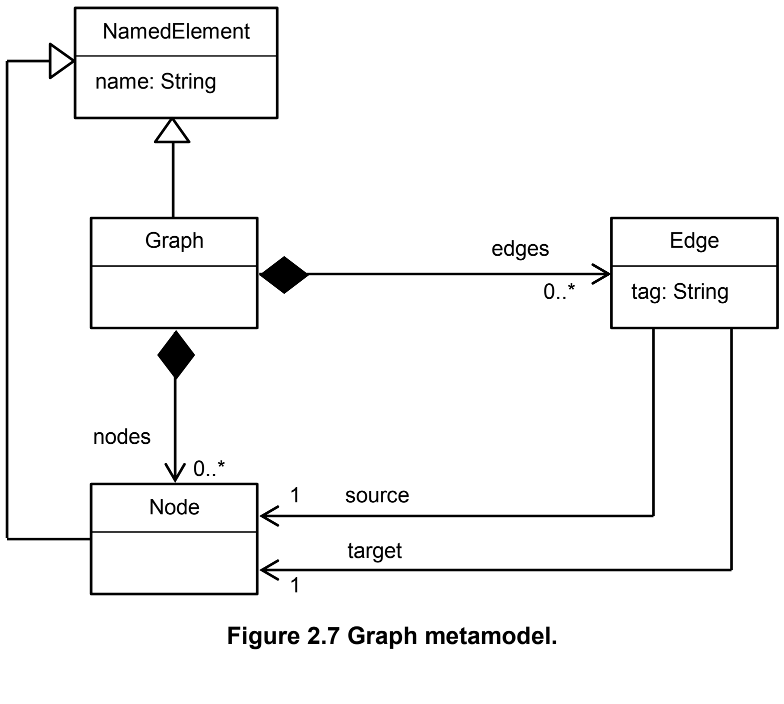


Figure 2.7 Graph metamodel.

Two kinds of association relationships can be established between metamodel metaclasses (and therefore between model elements): *containment* and *reference*.

A *reference* relationship, which is represented as an UML directed association, expresses that an instance of the source metaclass will have a reference to one or more instances of the target metaclass. For instance, the two relationships shown in Figure 2.7 from *Edge* to *Node*, which are named source and target, are references (i.e. each *Edge* instance has two references to *Node* instances, their *source* and *target* nodes).

A *containment* relationship, which is represented as an UML aggregation, expresses a part-of relationship from a container element to a contained element. Such a relationship has three properties:

- Exclusive ownership: the contained element cannot be part of more than one container element.
- Dependency: the lifetime of a contained element is the same of the one of its container element.
- Transitivity: if an element A is contained by an element B and B is also contained by another element C, then A is contained by C.

For instance, the relationships from *Graph* to *Node* and *Edge* in Figure 2.7 are of kind *containment*, i.e. *Node* and *Edge* instances are part of a *Graph* instance satisfying the three above properties.

Actually, the different kinds of relationships between classes are not shared by all metamodeling languages. Here we assume the existence of containment and reference relationship because they are considered by EMF (which is the modeling framework used in this thesis).

In MDE, the *four-level metamodeling architecture* (Clark et al., 2008; Brambilla et al., 2012) is normally used to explain the relationships between models, metamodels and meta-metamodels. An instance-of relationship (a.k.a. conforms-to relationship) is given between a model and its metamodel as well as between a metamodel and its meta-metamodel. The elements of a (meta)model are instances of (they conform to) the metaclass of its (meta)metamodel. For example, Figure 2.9, which represents the graph from Figure 2.8, shows (using an UML object diagram representation) an instance of the Graph metamodel from Figure 2.7.

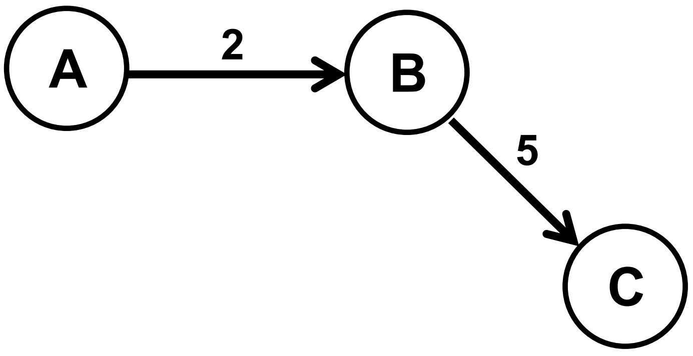


Figure 2.8 Sample graph.

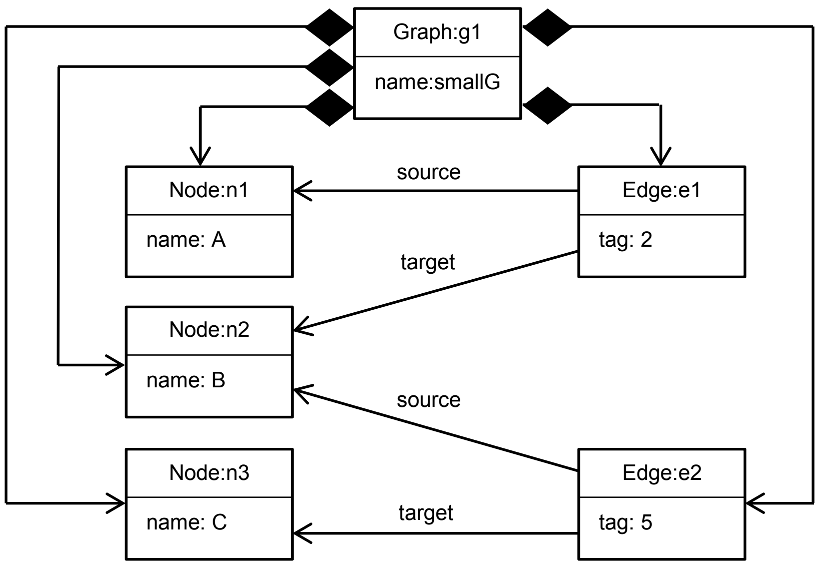


Figure 2.9 Sample graph object model.

The *object n1* is an instance of the *Node* metaclass, which is in turn an instance of the element from the meta-metamodel that represents metaclasses (e.g. *EClass* in the *Ecore* meta-metamodel).

2.4.2 Model transformations

The MDE approach, as explained at the beginning of Section 2.4, is model centric, where software artifacts can be fully generated from models. The technique that can be used to achieve this is commonly referred to as model transformation. Model transformations allow automating the conversion of models between different levels of abstraction. An MDE solution usually consists of a model transformation chain that generates the desired software artifacts from the source models. Three kinds of model transformations are commonly used: model-to-model (M2M), model-to-text (M2T) and text-to-model (T2M). Figure 2.10 shows a schema of transformation chain.

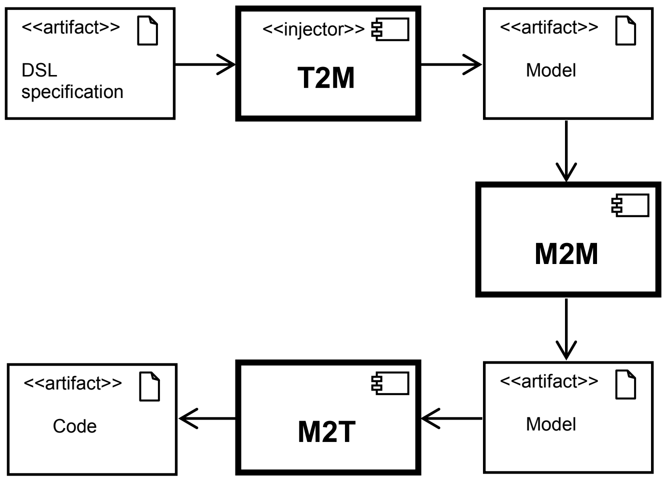


Figure 2.10 A transformation chain schema.

M2M transformations generate a target model from a source model by establishing mappings between the elements defined in their metamodels. One or more models can be the input and output of a M2M transformation. M2M transformations are used in a transformation chain as intermediate stages that reduce the semantic gap between the source and target representations. Frequently, a model transformation chain is formed by a single M2T transformation because this semantic gap is not large.

M2T transformations generate textual information (e.g. source code) from an input model. M2T transformations produce the target artifacts at the last stage of the chain.

Obtaining models from existing textual artifacts (e.g. GPL code, XML documents or data file) is required in scenarios such as software reengineering or tool interoperability (Brambilla et al., 2012) in order to obtain the model that is input to a model transformation chain. The "injection" term is commonly used to refer to such processes that involve parsing the artifact's text and generating the corresponding model. T2M transformations can be used to implement model injectors.

M2M transformations are normally expressed with languages that allow specify mappings between metamodels in a declarative way. The most frequently used M2M transformation languages (e.g., QVT (QVT20), ATL (AtlanMod), ETL (Kolovos et al., 2008)) have a hybrid nature (they also include imperative constructs) since M2M transformations can be very complex to be expressed only by using declarative constructs. M2T transformations are normally expressed with template languages which allow traversing models by following the structure imposed by its metamodel, and specify which code must be generated (e.g. MofScript (MOFScript), Acceleo (Acceleo) and Xpand (Klatt, 2008)). Finally, an example of T2M transformation language is Gra2MoL (Izquierdo et al., 2008) which generates a model injector from a specification of the mapping between the source artifact grammar and the target metamodel. In this thesis, M2T transformation languages have only been used to build a DSL compiler as one-step model transformation chain.

A survey on model transformation languages can be found in (Czarnecki et al., 2006), and a survey on the state of the art in model transformation can be found in (Mens, 2013).

2.4.3 Domain-specific language (DSL)

To create an application, a developer must first study its requirements and design a solution by using concepts from the problem domain. Then the application can be implemented by using concepts from some specific platform domain, by using a programming language. The semantic gap between both domains requires an additional effort for the developer who must express those concepts and algorithms in a concrete machine language. General Purpose Languages (GPLs) offer a higher level of abstraction and reduce the semantic gap because the developer does not need to worry about concrete machine details, but he/she still needs to adapt the concepts from the problem domain to the solution domain.

In contrast to General Purpose Languages (GPLs), Domain-Specific Languages (DSLs) are languages that are designed to solve problems in a specific domain. Creating a DSL is only worthwhile if the benefits of using it are greater than the cost of implementing it. In the MDE setting, the DSL and modeling language terms are used to refer to the languages used to build models, which are usually created by applying metamodeling, that is, the language allows creating models whose structure is determined by a metamodel.

We can distinguish three types of stakeholders in the creation of a DSL (Kolovos et al, 2006; Kleppe et al., 2008):

- DSL developers. They define, design and implement the DSL and associated tools.
- DSL users. They use the DSL and the tools, and also provide feedback to improve the DSL.
- Domain experts. They have in-deep knowledge about the DSL domain and they usually are DSL users. Sometimes developers are domain experts as well.

The process for creating a DSL has four steps (Mernik et al., 2005): study of viability, domain analysis, design and implementation. The domain analysis identifies the main concepts of the language and relationships between them. In the design phase we need to distinguish between graphical and textual DSLs. This decision will depend on the language constructors and the kind of final users. There are several techniques to implement a DSL: external DSL, internal DSL (embedded into some programming language) and DSL created by using a DSL workbench (e.g. Xtext and EMFText) (Fowler, 2010).

DSLs have been used since the early years of programming; however, MDE has substantially increased the interest in them. Most MDE solutions involve the definition of one or more DSLs in order for users to create the models that are required.

DSL elements

A DSL consists of three basic elements: *abstract syntax*, *concrete syntax* and *semantics*.

The *abstract syntax* describes the set of language concepts and their relationships, along with the rules to combine them. Metamodeling provides a good foundation for abstract syntaxes, and it is the most widespread formalism in MDE but other formalisms have also been used over the years, such as grammars and XML schemas.

The *concrete syntax* defines the notation of the DSL, which can be textual or graphical (or a combination of both). To decide the kind of notation, DSL developers have to consider the group of users to which the DSL is targeted. Textual syntaxes are preferred for developers while graphical ones are intended for users with little software knowledge (Kelly et al., 2009). Since the development of a textual DSL is normally easier than a graphical one, it is recommended to first define a textual DSL and then transform it into a graphical one if necessary (Völter, 2009)

Figure 2.11 shows how the graph in Figure 2.8 could be expressed by means of a textual DSL whose abstract syntax could be defined by the metamodel in Figure 2.7.

The *semantics* defines how the DSL specifications (i.e. DSL models) are interpreted; there are several approaches for defining it [Kleppe, 2008]: denotational, operational, translational and pragmatic. *Denotational semantics* makes use of usually complex mathematical formalism and operators to describe the DSL semantics. *Operational semantics* explains the DSL semantics by means of a sequence of operations which describes a virtual machine executing the DSL programs. *Translational semantics* is provided by building a translator (i.e., a compiler) to another language that already has a well-defined semantics (e.g., a programming language) or an interpreter. *Pragmatic semantics* is perhaps the simplest of them and tries to explain the meaning of the DSL sentences by showing several examples and the results obtained from executing them. However, this kind of approach makes difficult the language comprehension because it forces developers to figure out the semantics from the examples.

```
Graph graph {
      Nodes { A,B,C}

      Edges {
            A -> B (2)
            B -> C (5)
      }
}
```

Figure 2.11 SimpleGraph DSL example.

DSL implementation

As indicated above, the DSL implementation techniques are classified in three categories in (Fowler, 2010): *External DSL*, *Internal DSL* and DSL created with metamodeling-based tools. Really, these tools aim to define external DSLs. Therefore, two main categories could be considered external and internal.

An *external DSL* is created from scratch. Therefore, developers are free to define any kind of syntax. SQL and HTML are examples of extern DSLs. Grammar formalisms are normally used to define the concrete syntax of these DSLs; once the grammar is defined, a parser is created to recognize the DSL sentences. The data structure generated by the parser is used to build a compiler or interpreter that defines the semantics. With MDE, DSL interest is growing continuously, and several metamodeling-based tools are available to facilitate the creation of external DSLs. These tools are named DSL workbench in (Fowler, 2010).

A *DSL workbench* can automatically generate a complete infrastructure for the DSL. There are two types of tools: (1) grammar-oriented, when the definition process starts from a concrete syntax specification by using an annotated grammar to guide the abstract syntax's metamodel creation (e.g.

Xtext (Eysholdt et al., 2010)); and (2) metamodel-oriented, when the definition process requires the abstract syntax's metamodel in addition to the grammar specification (e.g. EMFText (EMFText)). This kind of tools generates tooling support for injecting a model from a DSL textual specification and generating such a specification from a model.

An *embedded DSL* (or *internal DSL*) is a particular form of using a general purpose programming language which becomes the host language. This kind of DSL exploits the characteristics and syntax of the host language by creating a library of methods which provides the concrete syntax of the DSL. Object-oriented languages and functional languages are normally used as host languages. Examples of embedded DSLs are RubyTL (Sánchez et al., 2006) which is embedded in Ruby and Kiama (Kiama) which is embedded in Scala. Fluent-APIs is gaining acceptance as technique to create internal DSL in typed object-oriented languages as Java (Fowler, 2010).

Unlike internal DSLs, the definition of an extern DSL at hand requires a major implementation effort because a complete infrastructure for the language is necessary to be created. However, it is easier to define error control, static analysis and verification mechanisms because the developer is not constrained by a host language. DSL workbenches have emerged with the purpose of automate the building of external DSLs and they are increasingly used as its quality improves.

Quality requirements for a DSL

The MDE paradigm promotes the use of DSLs in the development of systems and applications. The quality of these DSLs has a direct impact on the quality of the development process and the software obtained. A set of desired quality requirements for a DSL is the following (Kolovos et al, 2006; Hermans et al., 2009).

- **Expressiveness**. Concepts defined in the language must correspond to important domain concepts and each element of the language must represent only one domain concept.
- **Usability**. A DSL must have tool support for creating and editing its specifications.

- **Integrability**. A DSL must be able of being integrated with other tools or languages.
- **Reusability**. A DSL makes easy to reuse partial or entire solutions at model level.
- **Extensibility**. New elements can be incorporated to the DSL.
- **Learnability**. This requirement refers to any concept like language comprehension or ease of learning which makes easier the use of the DSL.

2.5 The Eclipse modeling framework (EMF) and Ecore

We have developed the tooling that supports the generative architecture presented in this thesis using the Eclipse Modeling Framework (EMF) (Steimberg et al., 2008). This framework is the basic infrastructure for the Modeling Project of the Eclipse Platform. This project integrates a set of tools for applying MDE. EMF is currently the most widely used modeling framework, and has significantly contributed to that MDE can be experimented in the academic and industry communities.

EMF is composed of a metamodeling language called *Ecore* as core element, and the tooling needed for the creation and manipulation of Ecore (meta)models, which is usually referred also as EMF.

Ecore is based on the MOF meta-modeling language [MOF, 2006]; more specifically it was conceived as a reduced version of MOF that includes the basic elements normally required to build metamodels. In fact, after the definition of Ecore, OMG organized MOF in two languages: EMOF (Essential MOF) that is very similar to Ecore and CMOF (Complete MOF) that extends EMOF with advances modeling concepts.

The Ecore meta-metamodel supports all the constructs defined in Section 2.4.1 for metamodeling languages. Figure 2.12 shows the main concepts and relationships of the Ecore metamodel (the least relevant references and attributes have been omitted for the sake of clarity).

Metaclasses are represented as EClasses and structural features are represented as EStructuralFeatures, which may be attributes (EAttributes) or relationships (EReferences). The type of a structural feature may be an EClass if it is a reference or an EDataType if it is an attribute. The same meta-metaclass EReference is used for representing both containment relationships and references (it has a boolean attribute to indicate if the relationship that represents is a containment or a reference). A metamodel is represented in Ecore as an EPackage, which may include subpackages (i.e. composite metamodels). In addition, Ecore includes a few elements focused on code generation or integration with Java and Eclipse; for example, the EFactory to create model elements. EMF provides an API to programmatically manage models and metamodels.

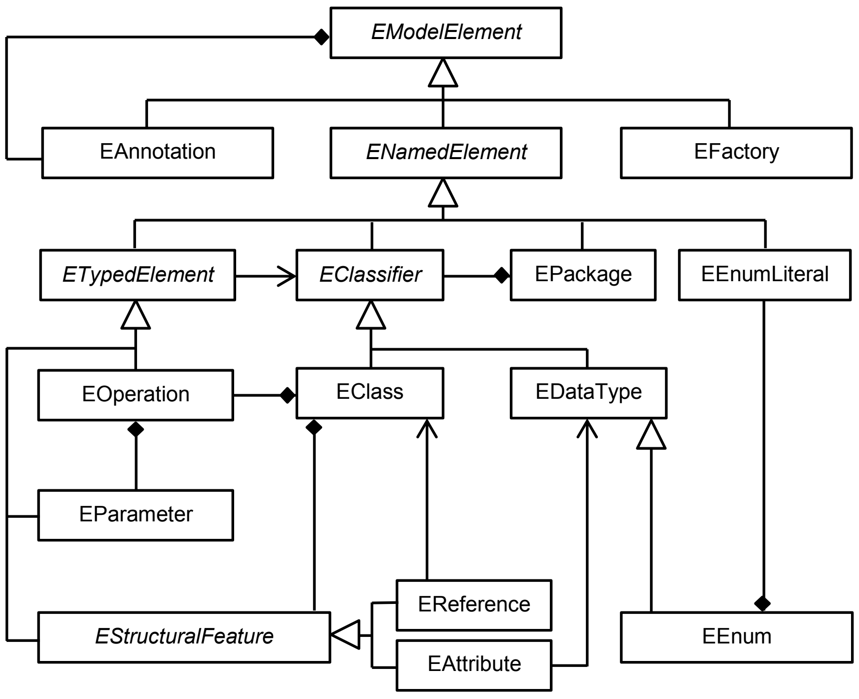


Figure 2.12 Ecore meta-metamodel.

The two basic interfaces of the EMF API are EObject and Resource: they represent the root class of every model element and the medium in which model elements are stored, respectively, and are essential for any application that uses the framework.

Finally, EMF provides tools and runtime support to produce a set of Java classes for the model, a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor.

2.6 The EMFText tool

Several DSL definition tools (a.k.a. DSL workbench (Fowler, 2010)) are available in the MDE setting, which have achieved high-level of maturity (i.e. robustness, good support and documentation, and usability among other qualities). Xtext (Eysholdt et al., 2010), MPS (Campagne, 2014) and EMFText (EMFText) are some of the most widely used DSL workbench.

They are based on metamodeling and normally generate an editor, injector and extractor from the DSL grammar and the DSL metamodel. While an injector obtains models from DSL programs, an extractor performs the opposite transformation.

As indicated in section 2.4.3, some DSL workbenchs (e.g. Xtext) can generate the metamodel from the language grammar specification. However, the generated metamodel has poor quality because it includes superfluous elements and grammatical aspects. A M2M transformation is then required to convert models generated by Xtext into models conforming to the desired metamodel. A crucial task in our work has been the definition of the metamodel used to represent context information and context quality. Therefore, we have used a DSL workbench that given a DSL metamodel (i.e. the DSL abstract syntax) requires to developers specify a grammar that defines the DSL concrete syntax, in particular we have chosen EMFText.

EMFText is an Eclipse plug-in that allows developers to define a textual concrete syntax for languages whose abstract syntax is described by means of an Ecore metamodel. In order to define a new DSL, it provides a simple concrete syntax specification language - the Concrete Syntax Specification Language (CS) - which is based on EBNF (EBNF, 1996).

A DSL specification for EMFText consists of a DSL metamodel and a concrete syntax specification. Taking both specifications the EMFText code generator creates an advanced textual editor along with an injector and a printer (i.e. an extractor). The injector is based on the ANTLR parser generator (Parr, 2013), a language tool for constructing compilers and translators from a grammatical description.

The basic language development process with EMFText is depicted in Figure 2.13. This process is an iterative process composed of the following basic tasks: (1) Specifying a Language Metamodel, (2) Specifying the Language's Concrete Syntax, (3) Generating the Language Tooling and (4) Optionally Customizing the Language Tooling.

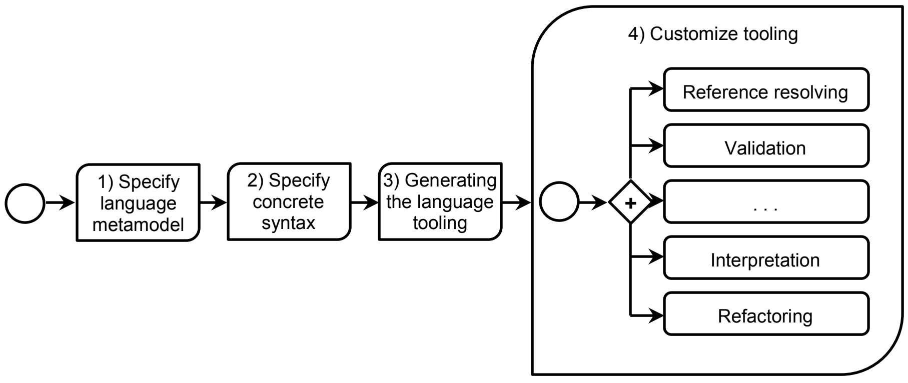

1) Specify language metamodel
2) Specify concrete syntax
3) Generating the language tooling
4) Customize tooling
Reference resolving
Validation
. . .
Interpretation
Refactoring

Figure 2.13 EMFText language development process.

As EMFText is integrated with EMF, the DSL abstract syntax's metamodel has to be specified with Ecore. For example, the metamodel shown in Figure 2.7 could be used to define the abstract syntax of the SimpleGraph DSL shown in Figure 2.11. Once this metamodel is defined, the concrete syntax is expressed with the CS language. A CS specification is formed by a set of EBNF-like rules. Figure 2.14 shows an excerpt of the concrete syntax for the SimpleGraph DSL

From a complete syntax specification, the EMFText code generator is used to generate an advanced textual editor and a customizable language infrastructure as several Eclipse plug-ins. As indicated above, it also generates the extractor and injector components to load and store model instances, respectively.

```
SYNTAXDEF graph
FOR <http://www.modelum.es/graph>
START Graph


TOKENS {
      DEFINE COMMENT $'//'(~('\n'|'\r'|'\uffff'))*$;
...
}


TOKENSTYLES {
      "Graph" COLOR #7F0055, BOLD;
...
}


RULES {
      Graph ::= "Graph" name[] "{"  "Nodes" "{"
           nodes ("," nodes)* "}" "Edges" "{" edges* "}"  "}";
      Node ::= name[] ;
      Edge ::= source[] "->"  target[] "(" tag[] ")"  ;
}
```

Figure 2.14 Concrete syntax for the SimpleGraph.

The last step is optional. EMFText helps in customizing the language tooling with a number of additional functions such as semantic validation of language expressions, language compilation, language interpretation, or new editor functions like custom quick-fixes, extended code completion, post-processors or refactoring.

2.7 MOFScript

We have used the MOFScript language (Oldevik et al., 2005) to write the M2T transformations that implement the engine for the DSL defined in this thesis. MOFScript is implemented as an Eclipse plug-in that is integrated into EMF, but it can also be executed as a standalone application. A MOFScript transformation has as input one or more models and their corresponding Ecore metamodels, and generates one or more target software artifacts as output. Such transformation is formed of a set of rules which are called explicitly and appear as methods which can include parameters.

One or more rules are defined for each of the elements of the Ecore metamodel. Additionally, the developer must define a main rule which will be the starting point of the transformation. Each rule can also be specified within the context of one element of the model, in the sense that the values of the context element can be accessed directly from inside the rule.

Figure 2.15, shows an excerpt of a MOFScript transformation for a model specified with the SimpleGraph DSL. This transformation generates a Java file which includes the code for painting the graph. The excerpt includes the main rule (the starting point of the transformation) as well as part of a rule which generates a Java paint method for depicting the edges and the nodes of the graph.

```
texttransformation paint (in gf:"http://www.modelum.es/graph") {

gf.Graph::main () {
    file (self.name+".java")
    self.header()
    self.initialize()
    self.paint()
}

gf.Graph::paint(){
    println('       public void paint(Graphics g){')
    println('       g.setColor(Color.BLACK);')
    println("");
    self.edges->forEach(e:gf.Edge){
        '           g.drawLine(nodes.get("' e.source.name
        '").width,nodes.get("'e.source.name
        '").height,nodes.get("'e.target.name
        '").width,nodes.get("'e.target.name'").height);'
        println("")
    }
    self.nodes->forEach(e:gf.Node){
        '           g.setColor(Color.GREEN);
                    g.fillOval(nodes.get("'e.name'").width-radio/2,
        nodes.get("'e.name'").height-radio/2, radio*2, radio*2);
                    g.setColor(Color.BLACK);
                    g.drawString("'e.name'",
        nodes.get("'e.name
        '").width, nodes.get("'e.name'").height);'
        println("")
```

Figure 2.15 A MOFScript transformation for a SimpleGraph model.

The *main* rule is defined within the context of the *Graph* element, so it can access to the properties of this element, such us the name of the graph (*self.name*). This name is used to create an output file with the name of the graph and a '.java' extension. Then the main rule explicitly calls other rules (*header*, *initialize* and *paint*), which will generate some parts of the Java file.

The *paint* rule uses the *println* sentence to write to the output file. As it is defined in the context of a graph object, it can access to the "edges" relationship. The *forEach* sentence iterates through all the *Edges* which belong to these relationship, generating code for each of the edges.

3 State of the art

The latest advances

By the time a man realizes that maybe his father was right,
he usually has a son who thinks he's wrong.

Charles Wadsworth

This chapter presents the state of the art in the field of context and QoC modeling. Having introduced the basic concepts regarding context and quality of context for context-ware applications, and model–driven engineering in Chapter 2, we shall now carry out a detailed analysis of the most relevant proposals published in literature which are related to our approach. This analysis is organized as follows. We shall first describe the different approaches, separating those which are based on MDE (Section 3.2) from those which are not (Section 3.1), and some approaches that are less relevant to the purpose of this thesis (Section 3.3). The works presented will then be contrasted in order to determine how they address the central aspects of our approach (Section 3.4) which is based on a DSL for the modeling of context named MLContext.

3.1 Context modeling approaches without MDE

In this section, we analyze the principal approaches related to context and QoC modeling which do not make use of MDE techniques. For each approach, we indicate its main aim and how context is represented and managed in applications. We have considered the frameworks JCAF, OCP, and SAMURAI, from which only the latter supports QoC. Moreover, we present the SOUPA standard ontology.

3.1.1 JCAF

The Java Context-awareness Framework (JCAF) (Bardram, 2005) is a Java-based infrastructure and programming API for creating context-aware applications. The most important concept of JCAF is the *context service*, which receives, manages and stores the context information for the entities. Several context services can cooperate within an application.

The JCAF framework provides a Java API for the definition of the context of an application as an object-oriented model (see Figure 3.1). This API includes some core modeling interfaces, such as *Entity*, *Context*, *Relation*, and *ContextItem*, as well as default implementations of them. One example of default implementation is the *GenericEntity* class, which implements the Entity interface and can be used to create concrete entities (e.g. in the context of an Office: desks, clerks, pens, etc.) using specialization. A context (e.g. an Office context) would be modeled as a class implementing *Context*. Physical location and the status of an operation are examples of context items which could be modeled as instances of a class that implements *ContextItem*. Examples of relations are "uses" or "located". Therefore, we can model that "John is located in office 123" where "John" is the entity, "located" is the relation, and "office 123" is the context item. An entity can be in the context of another, because it also implements the *ContextItem* interface.

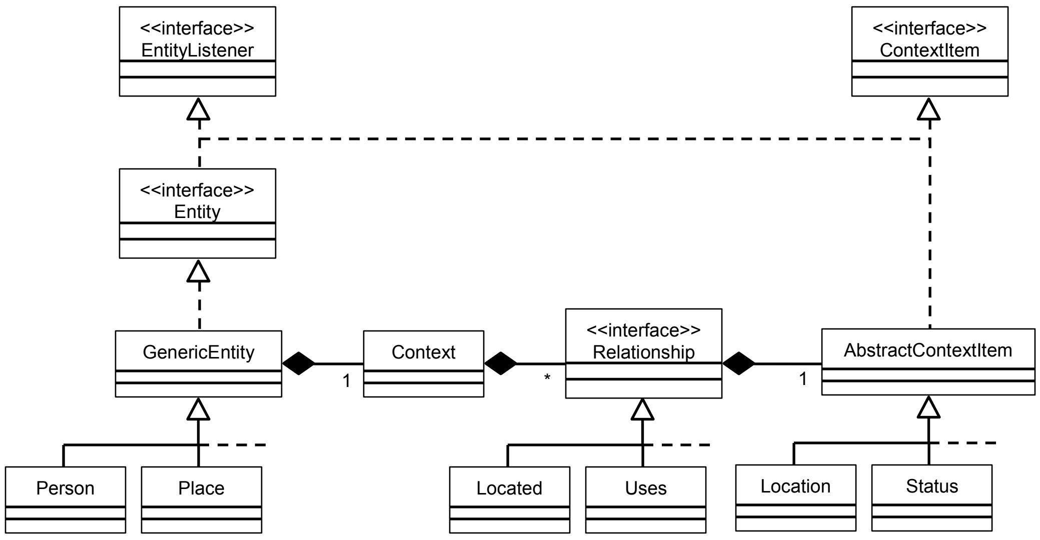


Figure 3.1 UML model for the JCAF context classes.

The JCAF framework can also handle the acquisition and transformation of context information. A context monitor can continuously supply context information (i.e. items) to an entity. Finally, the JCAF API provides the ContextService interface which has methods for adding, removing, getting and setting entities, and also for storing models by using Java serialization.

3.1.2 OCP

OCP (Open Context Platform) (Nieto et al., 2006) is a framework for the development of adaptive applications. Adaptiveness is the capacity of a system of reacting in order to appropriately respond to changes in the environment. Context-awareness is a necessary feature of these systems since it permits these changes to be characterized. Adaptive systems can be developed as Java applications on top of OCP in a layered arrangement. These applications use OCP to generate and obtain up-to-date context in a variety of configurations ranging from totally decentralized to simple distributed and monolithic systems. Thus, applications are not aware of the communication infrastructure used to distribute context information. They only see an API through which they may play two different roles: *producers* and/or *consumers*.

A *producer* is a software element (which usually represents a hardware element, e.g. a sensor) which produces data and sends it to OCP to fill up the context of entities involved in the system (usually users). A *consumer* is a service or application which consumes context information (e.g. a location service which delivers location information to other services or applications which need the context of the users that must be located in the system). OCP assume a dichotomic nature for the context information: *static context*, which is information that remains static over time, and *situation context*, which is information that changes dynamically.

Figure 3.2 shows the context hierarchy of OCP. Simple contexts are pieces of context information, and complex contexts are composed of several simple contexts. OCP has different types of context for the situational context as well as for the static context.

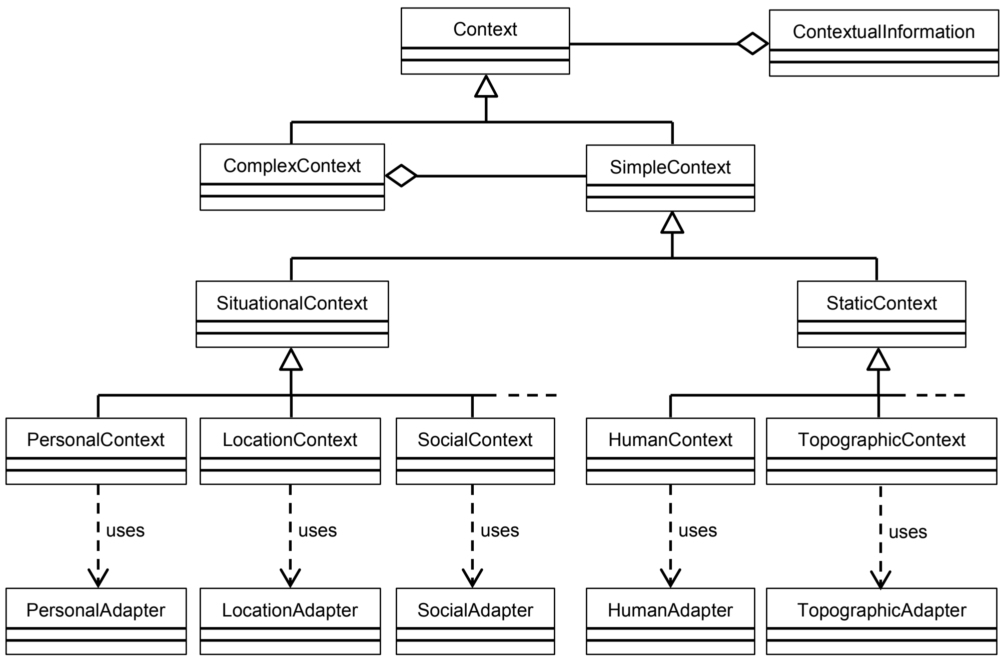


Figure 3.2 OCP context hierarchy.

Four elements are required to develop an OCP application:

1) An OWL-DL ontology to describe the contextual information of the application, which constitutes the knowledge base.

2) A Java class for each of the information producers.

3) A Java class for the consumers of the information, which depends on the context-aware application to be developed.

4) An initialization Java class that generates the individuals.

Different versions of the middleware have been developed, including a non-distributed application (monolithic), a totally decentralized version of the middleware over an ad hoc network and an OSGi (OSGI, 2012) based version of the middleware (Navarro et al., 2009).

3.1.3 SAMURAI

SAMURAI (Preuveneers et al., 2005) is a scalable event-based stream mining architecture developed within the frame of the FP7 BUTLER project as part of a secure and context-aware architecture for the Internet of Things. SAMURAI integrates and exposes well-known software building blocks for complex event processing, machine learning, knowledge representation, NoSQL persistence and in-memory data grids, and exposes them to applications as RESTful services. Therefore, SAMURAI can offer high level situational awareness with support for QoC. It includes the following capabilities among others:

1) It can convert raw low-level data and events into features that are more meaningful for use by the context-aware application or for comparing the quality of the context.

2) It can aggregate data from different context sources to increase the confidence in the quality of the inferred information.

3) It can leverage background information by specifying it in semantic models and ontologies (this capability includes spatial reasoning with a GeoSPARQL specification).

4) It can identify frequent co-occurrences in event streams to derive event patterns of interest for the context aware applications.

To make use of SAMURAI, an application developer identifies the sources of context information and the events these context providers can offer. Typical examples are location events where the attributes of the events not only provide the coordinates but also the precision and accuracy quality parameters of the positioning technology if available.

Users, context providers and context sources are entities in SAMURAI and they are represented as Java beans. These beans are created by loading an XML-based Spring beans descriptor file. When the context providers are bound to SAMURAI by using the Spring descriptor, the events generated by these context providers can be processed to ascertain whether a particular relevant context situation emerges.

SAMURAI can analyze the quality of context at runtime by using JavaScript code (variables and functions) which implements quality requirements for context situations. It can easily evaluate this code by leveraging Java's built-in JavaScript engine. By using JavaScript code rather than Java code, additional Java compilation steps at run-time are avoided.

3.1.4 The SOUPA ontology

We conclude this section by considering the Standard Ontology for Ubiquitous and Pervasive Applications (SOUPA) (Chen et al., 2005), which has been used to building the Context Broker Architecture (CoBrA) (Chen et al., 2003). The core of SOUPA consists of nine ontology documents that use the OWL language. There is also a set of SOUPA Extension ontologies which extended from the core to define additional vocabularies for specific types of applications (see Figure 3.3 from (Chen et al., 2005)).

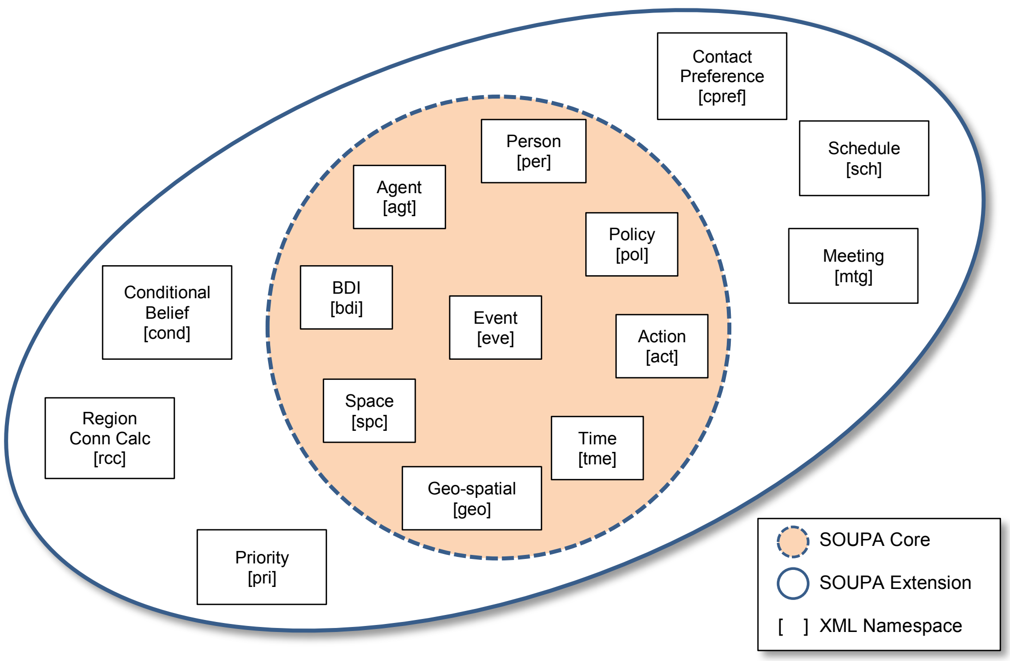


Figure 3.3 SOUPA core and SOUPA extension.

- The *Person* ontology defines a vocabulary to describe the profile and the contact information of a person.
- The *Agent*, *Action* and *BDI* ontologies are used to model entities as agents (Wooldridge et al., 1995).
- The *Policy* ontology defines concepts for security and privacy. In SOUPA, a policy is a set of rules defined by a policy creator (a user or an agent).
- The *Time* ontology is used for expressing time and temporal relations.
- The *Space* and *Geo-spatial* ontologies are designed for supporting reasoning about spatial relationships between geographical regions and to represent measurements of space.
- Finally, the *Event* ontology is used for describing the occurrence of different events (activities, schedules, and sensing events).

At this time, the SOUPA Extension consists of a set of experimental ontologies for supporting pervasive context-aware applications in smart spaces:

- The *Priority* ontology defines a vocabulary to assign priority values to agents. It can be used to set the precedence when there are conflicts between agents.
- The *Conditional* and *Unconditional Belief* ontology defines the vocabulary to describe a conditional belief, based on temporal and accuracy values. A conditional belief is true if its associated accuracy is above a predefined value and its timeliness is valid.
- The *Contact preference* ontology is used to define a set of rules to specify how the user likes to be contacted by the system in different situations.
- The *Meeting* and *Schedule* ontologies define the vocabulary for describing a meeting event and its attendees.

3.2 Context modeling approaches with MDE

In this section, we analyze the principal approaches related to context and QoC modeling which makes use of MDE techniques. It is organized by following the same structure as Section 3.1, where approaches were presented starting from those without QoC support. For each approach, we indicate its main aim and how the context is represented and managed in applications. We have considered the approaches CAMEL, ContextUML, PervML, CAUCE, Ayed et al. and COSMOS, from which only the last two supports QoC.

3.2.1 CAMEL

Context Awareness ModEling Language (CAMEL) (Sindico et al., 2009) is an Ecore metamodel for representing the behavior and adaptation of context-aware systems. It has been defined as an UML extension which can be used to enrich the UML model of an application with elements related to context awareness. Figure 3.4 shows an excerpt of the CAMEL metamodel from (Sindico et al., 2009). In CAMEL the context-awareness concern is handled by means of three separated parts: context sensing, context adaptation triggering and context adaptation.

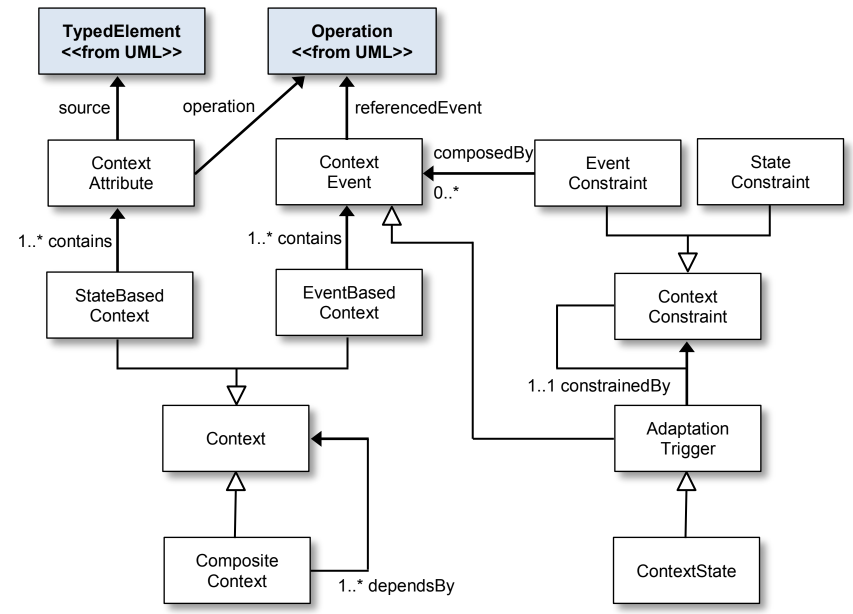


Figure 3.4 Excerpt of the CAMEL metamodel – context sensing.

StateBasedContext is a container for static context information with a set of attributes represented by the *ContextAttribute* metaclass. This metaclass has a *source* relationship with UML *TypedElement* from which it takes its value. The UML *TypedElement* represents a target system structural feature (e.g. an attribute, an association, a parameter or a reference).

The *EventBasedContext* consists of a set of events (*ContextEvent*) that are relevant for the context definition.

A *CompositeContext* is an aggregation of other contexts (both types, *StateBased* and *EventBased*).

CAMEL lacks a concrete syntax and no code generation has yet been defined. Instances of CAMEL metaclasess are directly created by using an EMF Eclipse modeling editor. It does not use a context taxonomy. Instead, it refers to generic contextual information. The adaptation and behavior specified in the model makes it non-reusable, because it depends on the application which it was created for.

CAMEL is presented as a first step for realizing a complete MDE framework for context awareness. Next step will be the realization of an enriched graphical representation for the CAMEL models.

3.2.2 ContextUML

ContextUML (Sheng et al., 2005) is a UML-based graphical language aimed at the design of context-aware web services, which includes constructs for modeling context information. The MOF metamodel in Figure 3.5 defines the abstract syntax of the language.

Context information is modeled by a *Context* metaclass, which has two subtypes: *AtomicContext* (simple, low-level context) and *CompositeContext* (aggregates multiple atomic or composite context). The *ContextSource* metaclass models the resources from which context can be obtained (in ContextUML the context sources are context services).

Context UML focuses on the definition of context-aware services. Therefore most of its metaclasses are related to context-aware adaptation and services definition. The context-aware adaptation mechanism is modeled by using the *CAMechanism* metaclass, which can be a *binding* or a *triggering*. It relates the context with a context-aware object (represented by the *CAObject* metaclass) through the *ContextBinding* metaclass. The *ContextTriggering* metaclass models contextual adaptation. It contains a set of *ContextConstraints* and a set of *Actions*. The actions are executed only if all context constraints are evaluated to true.

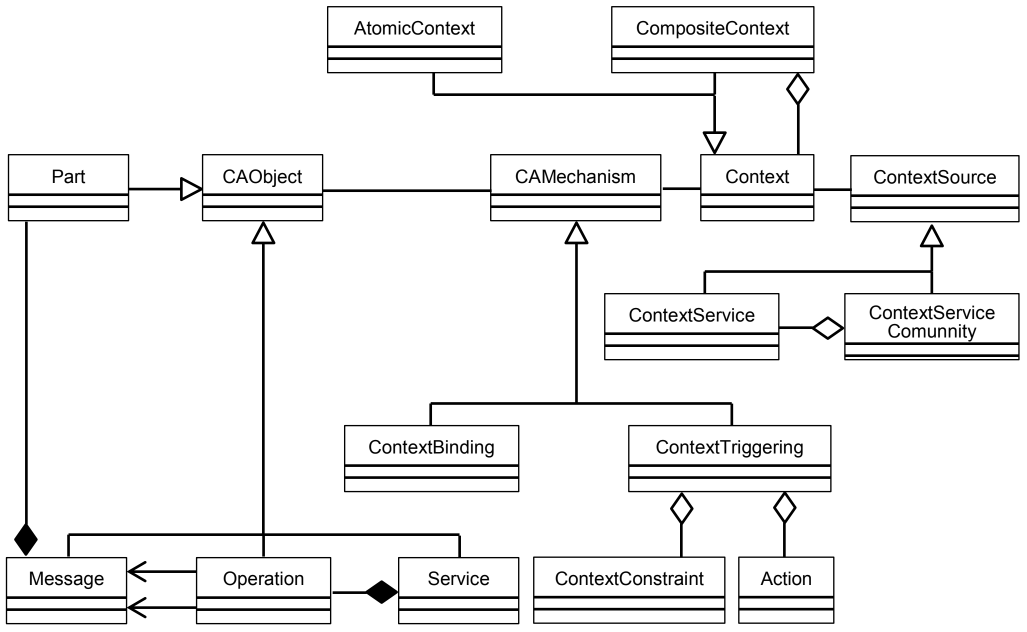


Figure 3.5 ContextUML metamodel.

There are four types of CAObjects: *Service*, *Operation*, *Message* and *Part*. Each service offers one or more operations and each operation have an input and/or an output message. Finally, messages can have one or more parts, which are parameters of the message.

ContextUML notation makes use of UML stereotypes. Figure 3.6 shows an example of the notation from (Sheng et al., 2005), which models an attractions searching service. It defines a composite context *HarshWeather* formed of two atomic contexts (*Temperature* and *RainLikelihood*) which are related to a context triggering *WeatherTrigger*. This trigger filters the outdoor activities if the weather is harsh.

ContextServ (Sheng et al., 2009) is a platform for the development of context-aware Web services, which is based on ContextUML. Web-services are specified by using the UML editing tool ArgoUML (extended with the ContextUML diagram) and executable implementations are automatically generated. It uses the RubyMDA transformer, which takes as input the XMI document representing the ContextUML diagram and generates a WS-BPEL specification (a standard for specifying executable processes). RubyMDA implements a set of transformation rules which define a mapping from ContextUML stereotypes to BPEL elements that can be executed by the open source execution engine jBPM-BPEL. ContextUML does not offer QoC support.

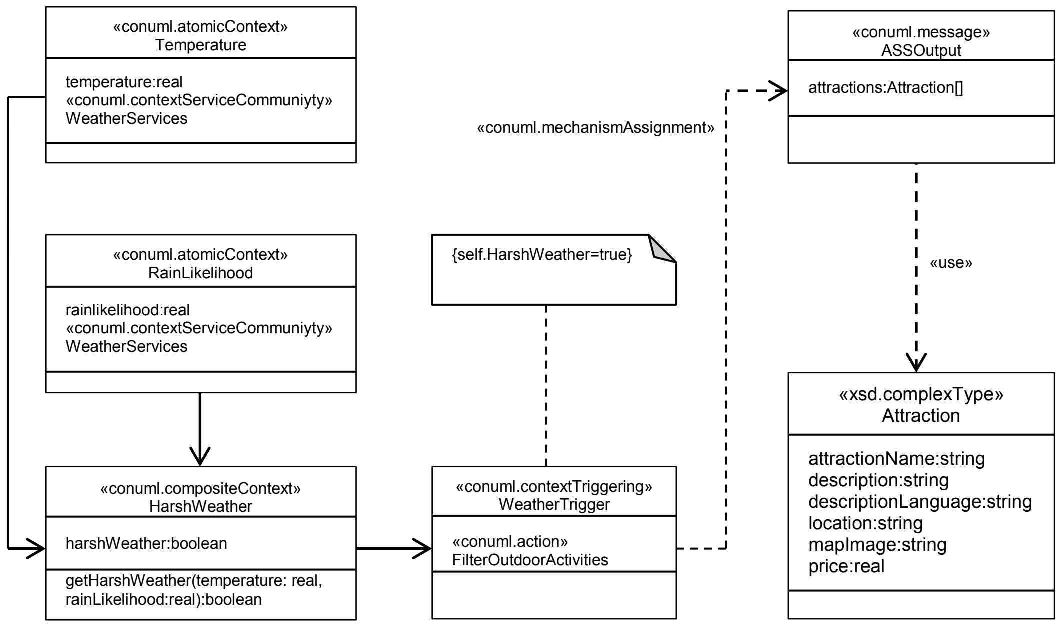


Figure 3.6 Example of ContextUML notation.

3.2.3 PervML

PervML (Serral et al., 2008) is a DSL for specifying pervasive systems, whose abstract syntax has been defined as an MOF metamodel and its concrete syntax is based on the UML 2.0 notation as indicated below. It promotes the separation of the developers' roles which can be classified as *System Analysts* and *System Architects*. The *System Analyst* describes the pervasive system (by using a service-oriented approach) and its requirements, and the *System Architect* specifies the devices (or the existing software) which support the system services. Figure 3.7 from (Serral et al., 2008) shows the PervML diagrams notation.

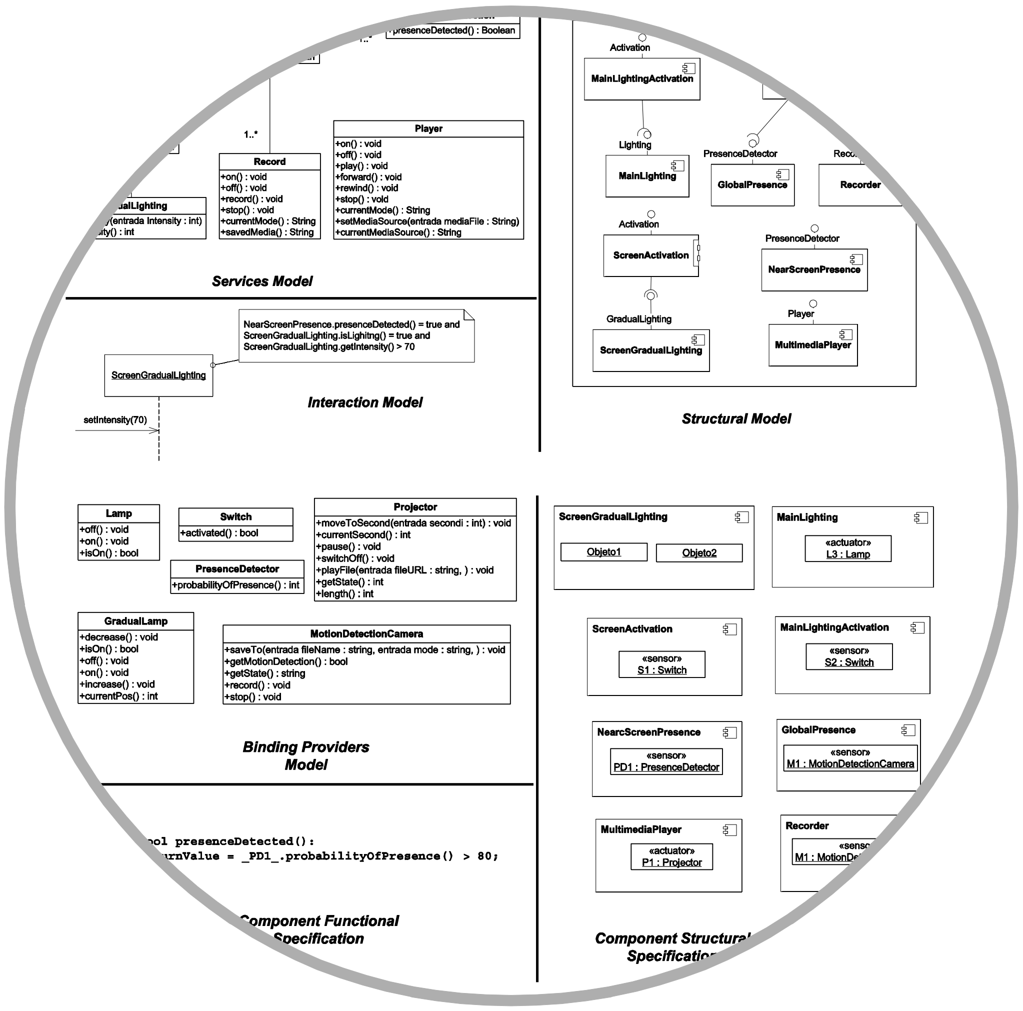


Figure 3.7 PervML diagrams notation.

Three models are specified by the *System Analyst*:

- The *Services model*, which describes the services provided in the system by means of class and statechart diagrams of UML.
- The *Structural model*, which indicates the instances of the services provided by the system and their location. This is represented by means of a location diagram which is based on the UML component diagrams.
- The *Interaction model*, which specifies the interactions produced as a response to some system event. UML sequence diagrams are used to represent these interactions.

And other three diagrams are created by the *System Architect* :

- The *Binding providers model*, which describes the types of devices in the system. It uses the UML Class notation to specify the methods provided by each device.
- The *Component structure model*, which is used to assign devices to instances of the services by means of a notation based on UML component diagrams.
- The *Functional model*, which specifies the actions to be executed when a service is invoked. The Action Semantics Language (ASL) is used to specify these actions.

Note that, PervML does not have an explicit context model because it is a service-oriented approach. Context information is implicit in the *Binding providers model* through the definition of the methods for retrieving it from the devices used in the system.

The PervML metamodel has been implemented as an Ecore metamodel by using the Topcased plug-in of Eclipse (Topcased, 2005). Following an MDE strategy, PervML models are transformed into OSGi bundles (OSGI, 2012) of the built pervasive system. The generation of the bundles′ Java code is achieved by means of a model to text transformation implemented in MOFScript. This transformation also generates an OWL ontology that can be used for adaptation purposes as explained in (Serral et al., 2007).

3.2.4 CAUCE

CAUCE (Tesoriero et al., 2010) is presented as a Model-Driven Architecture (MDA) solution for developing context-aware applications which defines three layers of models: Computing-Independence Models (CIM), Platform-Independence Models (PIM) and Platform-Specific Models (PSM). The third layer is used for the code generation from PSM models by means of M2T transformations. The other two layers are used to model different aspects of the context-aware system.

The first layer contains the *Task*, *Space* and *Social* metamodels which are defined in the Essential Meta Object Facility (EMOF) (EMOF, 2004). They represent the conceptual characteristics of the context-aware applications. The second layer represents the deployment, architecture and communication characteristics of the software. This layer contains the *Information Flow*, the *Referential Space* and the *Entity Context* metamodels. Figure 3.8 shows and schema of the CAUCE models.

Models of the second layer are obtained by using M2M transformations from the models of the first layer. These transformations are defined using the ATL transformation language (AtlanMod). The source code of the third layer is obtained through M2T transformations defined with the MOFScript language.

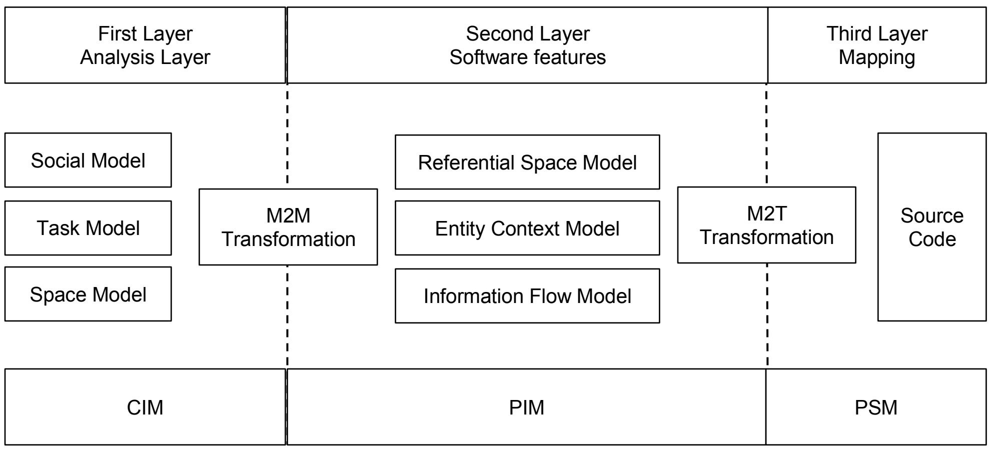


Figure 3.8 CAUCE overview.

- In the *Task model*, a task represents the work to be performed by an entity to fulfill some goal. This model represents the tasks performed by the system.
- The *Social model* describes the entities that are part of the system (e.g. PDAs, or LightSensors). In CAUCE, entities can play the role of sensors or actuators, in the sense that they are able to "perceive" environmental changes or "modify" the environment.
- The *Space model* is used to define entity locations.
- The *Information Flow model* represents the communication between entities. An entity can produce and consume information flows.

- The *Referential Space model* describes the dependency between entities.
- The *Entity context model* describes is the context that an entity can perceive from the environment.

This approach has some similarities with the PervML approach. Both make use of UML profiles for their modeling language and propose a set of models for describing similar aspects of the context-aware application by focusing on services, but CAUCE models contains more information about the system. While PervML makes use of the *Services model* to describe the services provides by the system and the *Functional model* to describe the actions to be executed by those services, CAUCE represents the tasks performed by the system and the task synchronization by means of the *Task model*. The physical space and dependences between devices and space are described in PervML by means of the *Structural* and *Binding providers* models. CAUCE describes that information in the *Space* and *Referential Space* models. Finally, PervML *Interaction model* represents interactions into the system and the CAUCE *Information Flow* model between entities.

CAUCE has been validated by generating code for the Microsoft .NET platform.

3.2.5 Ayed et al.

Ayed et al. (Ayed et al., 2007) describe an MDE approach for the development of context-aware systems based on MDA. They propose a process with six phases. This process takes into account both the collection of the context and adaptation mechanisms, and the languages needed for each phase are defined by means of an UML profile. Figure 3.9 from (Ayed et al., 2007) shows an excerpt of the extensions of the UML diagram for modeling context.

With regard to the context modeling, the identification of the required context information must be performed during the first phase, specifying several instances for the Context stereotype and the required context quality (instances of *ContextQuality* stereotype).

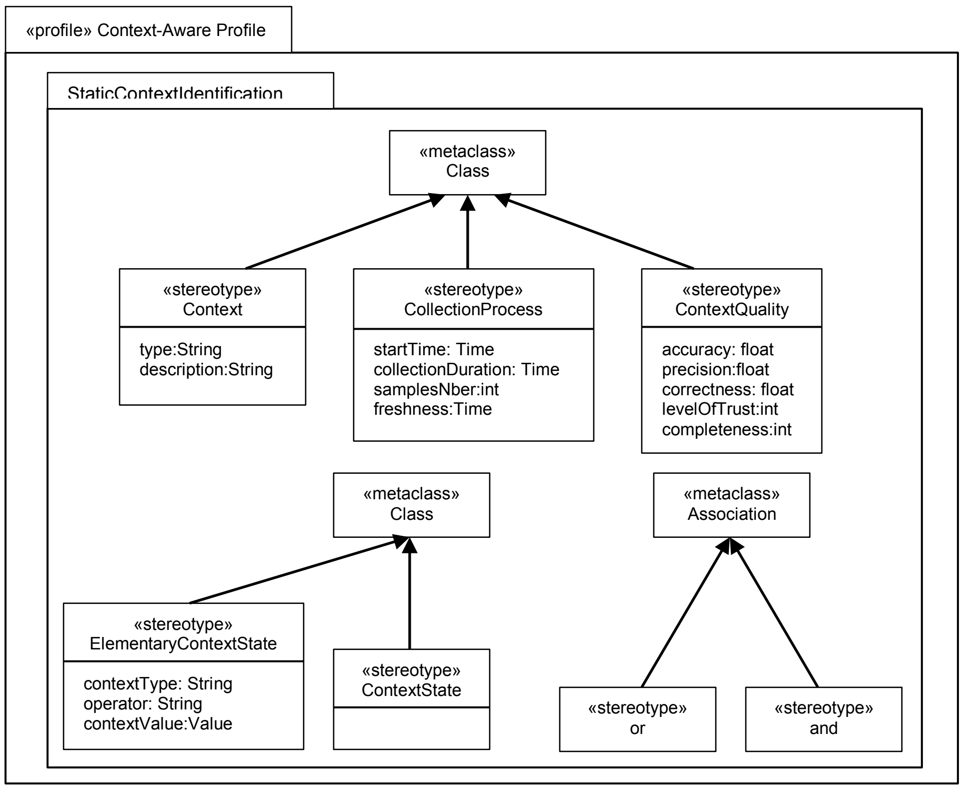


Figure 3.9 Extensions of the UML class diagram to model the context.

As shown in the picture, this approach considers the quality attributes: accuracy, precision, correctness, level of trust and completeness, but not freshness as it is part of the context information collection process. Note that though the developer can specify quality attributes, he/she can not model quality requirements for the application.

The *<<ContextState>>* stereotype can be used to define relevant context states for the application. The *<<ElementaryContextState>>* stereotype represents the context information, an operator, and a particular value of the context, and the relationships between instances of *<<ContextState>>* and *<<ElementaryContextState>>* are specified using Associations by means of the *<<And>>* and *<<Or>>* stereotypes.

These specifications must be made independently of the platform that will be used to collect the information. The UML profile for the context model includes the elements for collecting information (starting time of collection, number of samples, rate of sampling, etc.) which depends on a particular application. This approach does not propose different types of context, and it does not allow generating code but makes use of model-to-model transformations to create platform specific models of the same type (e.g. a platform-specific variability model from a platform-independent variability model), and they admit that the transformations in their approach could be complicated to write, but they do not explain the reasons.

3.2.6 COSMOS

COSMOS (COntext entitieS coMpositiOn and Sharing) (Chabridon et al., 2013) is a component-based framework with QoC support for context management. It is based on the notion of context node, i.e. context information modeled by a component (see Figure 3.10 from (Conan et al., 2007)). Context nodes are organized into hierarchies and they have the possibility of sharing information. Each node can receive information from their children nodes and can send information to their parent nodes. The nodes contain one or more operators to manipulate the context information.

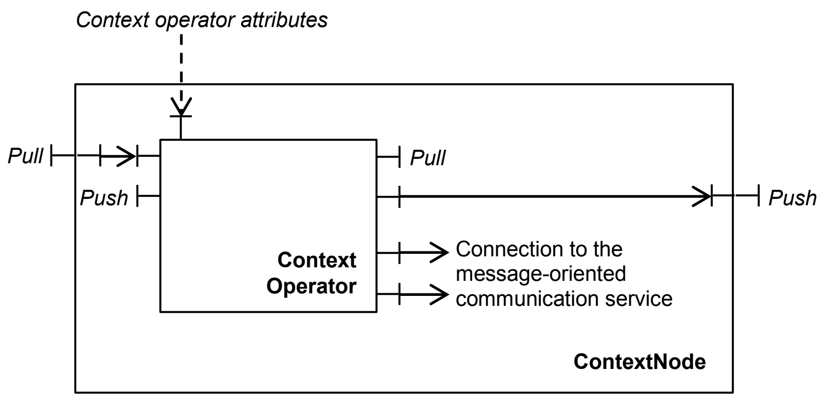

Context operator attributes
Pull
Push
Pull
Push
Context Operator
Connection to the message-oriented communication service
ContextNode

Figure 3.10 COSMOS abstract context node.

Figure 3.11 shows an example of context nodes hierarchy. At the leaves of the hierarchy are the context collector nodes, which are software entities that provide raw data about the environment (they can also provide information coming from user preferences). For example, *GPS manager* and *WIFI manager* are collector nodes. Each elementary context information provided by the collector nodes gives rise to a typed *chunk*. The context information is encapsulated into *messages* that circulate from the leaves to the root of the hierarchy. These messages contain chunks and/or other messages (composition). In our example, the *GPS manager* and the *WIFI manager* provide raw data about the user location. A chunk Location is created and sent to their parent nodes *GPSLocation* and *WLANLocation*. These nodes contain one operator which converts raw data to coordinates (application-dependent format). These coordinates are encapsulated into messages and sent to the *LocationChoice* node, which is in charge of selecting the best location information from their children nodes (*GPSLocation* and *WLANLocation*), based on the QoC.

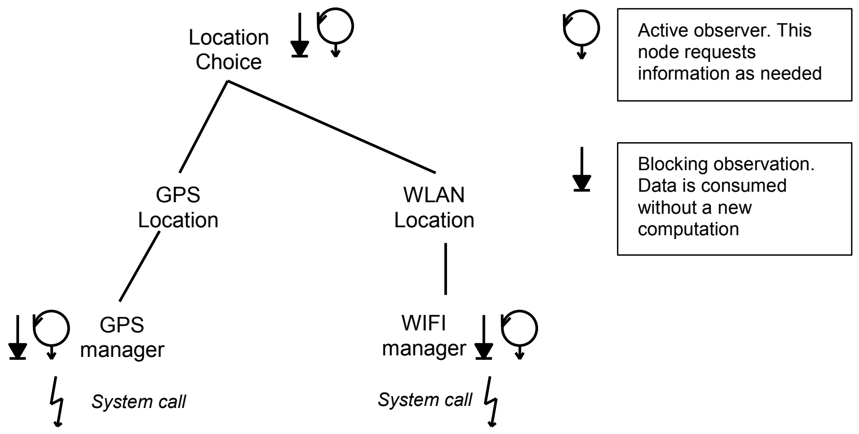

Location Choice
GPS Location
WLAN Location
GPS manager
WIFI manager
System call
System call
Active observer. This node requests information as needed
Blocking observation. Data is consumed without a new computation

Figure 3.11 COSMOS context nodes hierarchy example.

QoC is managed in COSMOS in the same way that context information. Messages containing information about QoC are processed by the context node operators through the context node hierarchy.

The architecture for the context nodes hierarchy is implemented in COSMOS with the Fractal Component framework (Bruneton et al., 2006) by using the Fractal ADL architecture description language (Leclercq et al., 2007) that is based on XML. Writing a composition of context information using Fractal ADL can be a complex task because of the very technical aspect of this language. To address these difficulties, the COSMOS framework provides a simple domain specific language called COSMOS DSL, which basically is used to specify the node hierarchies. Figure 3.12 shows an excerpt of the COSMOS DSL grammar, and Figure 3.13 an excerpt of a COSMOS DSL specification for the example of the Figure 3.11. The chunk definition for the *UserLocation* information includes the name of the Java classes which implement the chunk and the data type for the location information. The *gpsManager* collector node definition uses this chunk in order to send the data through the output to its parent node. The excerpt includes a *ComputeGPSUserLocation* operator definition which is used by the *gpsUserLocation* node to convert collector data (from its child *gpsManager*) to coordinates. This operator is implemented in the Java class *every2mallqoc.ComputeGpsUserLocation*.

```
cosmosdsl ::= (import_list)? (package_declaration)? (cosmosdsl_content)+ (facade)?
package_declaration ::= package_kwrd package_name ;
import_list ::= import_def ( import_def )_
cosmosdsl_content ::= chunk
                        | collector
                        | configuration
                        | context_node
                        | message
                        | operator
chunk ::= chunk_kwrd chunk_id = {
                              chunk_def
                              }
collector ::= collector_kwrd collector_id = {
                              collector_def
                              }
configuration ::= configuration_kwrd : { node_id | operator_id | collector_id }
                              attribute_list ;
context_node ::= node_kwrd node_id = {
                              context_node_def
                              }
message ::= message_kwrd message_id = {
                              message_def
                              }
```

Figure 3.12 An excerpt of the COSMOS DSL grammar.

A case study and an example of a context-aware scenario modeled with the COSMOS DSL will be presented in Section 6.5.

```
chunk userLocation = {
   classname : every2mallqoc.UserLocationChunk,
   typeparam : every2mallqoc.UserLocationInfo
}

collector gps_managerCollector = {
   output: {userLocation},
   classname: Gps_managerCollector
}

operator computeGpsUserLocation = {
   output : userLocationMsg,
   classname : every2mallqoc.ComputeGpsUserLocation,
   input : {userLocation}
}

node gpsUserLocation = {
   operator : computeGpsUserLocation
   children : gps_managerCollector
}
```

Figure 3.13 An excerpt of a COSMOS DSL specification.

3.3 Other approaches.

Next, we comment some approaches that have not been included in the previous sections because they are less relevant for the objectives of this thesis in the sense that they have a restricted notion of context or they follow a modeling approach which is not specific for context modeling.

Middlewhere (Ranganathan,et al., 2004b) is a distributed middleware infrastructure for location that separates applications from location detection technologies. MiddleWhere uses a spatial database for maintaining a model of the physical layout of the environment and storing location information from various sensors. It measures the quality of location information according to three metrics: resolution, confidence and freshness. Unlike our approach, this middleware only models physical space and the models cannot easily be extended by application developers. Moreover, the resulting quality of location information is never exposed to the applications.

WebML (Ceri et al., 2002) is a visual language for specifying the context structure of web applications and its presentation in one or more hypertexts. Firstly, an Entity-Relationship model or a UML class diagram is used to specify how the application contents are organized. Then, a WebML hypertext model describes how contents specified in the data schema are published in the application hypertext. WebML can generate code for adaptive web applications that can be manually adapted by web programmers. This proposal cannot be applied to model context-aware systems in general. The WebML model is also dependent on the application for which the hypertext was created.

Henricksen et al. (Henricksen et al., 2002) have attempted to model a context-aware communications scenario by using both entity-relationship (ER) models and class UML diagrams. The agents in this scenario rely upon information about the participants and their communication devices and channels. The authors experienced difficulties in distinguishing between different classes of context information (for example static versus dynamic information or sensed information versus information supplied by users). They found UML constructs to be more expressive than those provided by ER, but also more cumbersome. They suggest that the most appropriate approach for modeling context information is that of using special constructs designed with the characteristics of context in mind. In this respect, the DSL proposed in our approach have constructs specially designed to model context information.

3.4 Analysis and discussion

In this section, we will compare the approaches presented in Sections 3.1 and 3.2 and they will be contrasted to our proposal. This comparison is done using a set of criteria that we have identified to characterize each approach.

Table 3.1 summarizes the comparison conducted. The table shows to which extent the different approaches cope with some of the objectives defined in this thesis. The first column expresses the kind of context representation in which the approach is based, and the remainder columns indicate whether the approach has some property or not: (1) it offers a taxonomy of context types; (2) it is based on a DSL; (3) it supports QoC; (4) it uses models to represent the QoC (if any); (5) it allows context situations to be modeled and, finally the last column indicates if the approach is based on an MDE solution. The approaches are alphabetically sorted in the table. The last row (MLContext) corresponds to our approach.

Table 3.1 Comparison of context modeling approaches.

	Context model	Context taxonomy	DSL	QoC support	QoC model	Modeling of context situations	MDE
Ayed et al (a)	UML profiles	No	Yes	Yes	Yes	Yes	Yes(b)
CAMEL	UML extension /Ecore	No	No	No	No	No	Yes (c)
CAUCE	UML based model	No	Yes	No	No	No	Yes
ContextUML	UML extension	No	Yes(d)	No	No	Yes(e)	Yes
COSMOS	Component model	No	Yes	Yes	Yes(f)	Yes(g)	Yes
JCAF	Java classes	No	No	No	No	No	No
OCP	Ontology/ Java classes	Yes	No	No	No	No	No
PervML	UML based /Ecore	No	Yes(h)	No	No	Yes(i)	Yes
SAMURAI	Spring descriptor/ Javabeans	No	No	Yes	No	Yes(j)	No
SOUPA	Ontology	Yes(k)	No	Yes(l)	Yes(m)	No	No
MLContext	Ecore	Yes	Yes	Yes	Yes	Yes	Yes

(a) only a method proposed but not implemented
(b) incomplete approach and no code generation
(c) incomplete approach and no code generation
(d) based on UML 1.4 notation
(e) as composite context
(f) QoC is represented in the same context model
(g) context situations are defined by the context nodes architecture
(h) based on UML 2.0 notation
(i) partial support (context situations defined as a service)
(j) as JavaScript functions
(k) as different types of ontologies
(l) partial support (SOUPA extension for defining conditions)
(m) QoC is scattered through ontologies

Next, we discuss the results of the comparison for each criterion.

Context model

Some context modeling approaches lack of an adequate representation for certain context elements. For instance, OCP does not include the context sources in the models (i.e. an ontology). Therefore, the developer must write directly Java code for them, which is error prone. Others as PervML and ContextUML do not have the notion of entity. This makes difficult to generate code from their models if the target platform manages entities (e.g. JCAF code from ContextUML models). Another limitation found in ContextUML is that the assignation of different context sources to the same attribute in different instances of a class is not possible.

Noting that most approaches are based on an object-oriented model (e.g. components in COSMOS, Java classes in JCAF, Javabeans in SAMURAI, UML in PervML or Java classes in OCP), though they can use different types of context representation (e.g. COSMOS DSL, SAMURAI Spring descriptor, OCP ontology, PervML Graphical DSL or ContextUML DSL).

Context representation in Ayed et al., ContextUML, CAUCE and PervML is based on the UML notation, but UML profiles are not a first-class extension mechanism and do not allow to modify the UML metamodel so these context models must not violate the semantics of UML. DSLs created from scratch are considered to be more appropriate and more expressive, and the required development effort is less than for creating UML profiles (Kelly et al., 2009; Völter, 2009).

Context information in CAUCE is related to an entity, but the notion of entity is different from the one proposed by Dey et al. (Dey et al., 2000) and also differs from other approaches like JCAF. A CAUCE entity represents an active component that is able to perform system tasks to achieve the system goals.

Ontologies are suitable for statically representing context (e.g. OCP and SOUPA are approaches based on ontologies). They are well-known formalism for knowledge representation and reasoning which have been proposed for the Semantic Web. In spite of their advantages, it has been pointed out that they are not adequate to deal with uncertain or imprecise information

(Gómez-Romero et al., 2011), which is a problem inherent to context knowledge (e.g. sensor data may be inaccurate). Moreover, they are not easy to handle. The building and management of ontologies is a laborious task which often involves the design and implementation of complex algorithms (Khattak et al., 2009) and the creation of ontology management systems is still in an early stage (Maedche et al, 2003; Gómez-Romero et al., 2011). Ontology-based context management approaches have a low performance because existing ontology languages are not designed for retrieving large amount of context data or query optimization (Vanathi et al., 2010).

Nowadays there are several approaches which use ontologies (e.g. CAMPUS (Hisazumi et al., 2003) and OCP), but SOUPA is one of the most comprehensive ontologies for pervasive computing (Vanathi et al., 2010). The SOUPA ontology includes context aspects which are also supported by some of the approaches addressed here, such as user profiles or spatial and temporal aspects. SOUPA does not include a context taxonomy as such, but different types of ontologies regarding different types of context information such as personal information, time and space. One of the main drawback of this ontology is that SOUPA was defined bearing in mind the Context Broker Architecture (CoBrA) (Chen, 2004), so it includes concepts for interfaces, agents, security policies and agents priority, which are specific of certain types of applications and it is intended for user-centric applications and context reasoning.

As most of the proposals, our approach is based on an object-oriented model (i.e. an Ecore metamodel), but we include the notion of entity (as defined by Dey et al (Dey et al., 2000)) as one of their main concepts and remove the specific details of concrete applications from the context model. In this sense, our DSL does not allow to represent information specifying services as it is expressed by ContextUML, CAUCE and PervML models, because we do not consider them as part of the context model but application-specific elements. These approaches include them because are more oriented towards representing services than context. Moreover, they mix context information with technical details from the services, making the models application-dependent so they can not be reused by other applications.

Context taxonomy

The context information that can be included in a context model can be of different types. There are several studies about types of context and the kind of information that they may include, and some proposals for classifying them (Göker et al, 2002; Pires et al., 2005; Alarcón et al., 2005; Ardissono et al., 2007) (a survey on types of context can be read in Section 5.1.1). However only a few approaches has incorporated a context taxonomy which allows a developer to explicitly specify the type of context information to be modeled (OCP and SOUPA).

OCP is the only one of the reviewed approaches which incorporates a context taxonomy for specifying the type of context. The taxonomy includes some types of context like environment, personal, human or topographic, but the authors does not explain what context information fit in each of these categories.

SOUPA does not include a context taxonomy as such, but different types of ontologies regarding different types of context information such as personal information, time and space. In this sense, OCP is richer in types of context.

Our approach defines a taxonomy which includes the most used types of context in the literature as explained in Section 5.1.3. This taxonomy is a result of the domain study we have performed to define the main concepts of our DSL, and it can be extended with new types of context.

DSL approaches

Ayed et al., ContextUML, PervML and COSMOS provide a DSL to create models. On the one hand, the notation of Ayed et al., ContextUML and PervML is based on UML, which has the aforementioned limitations of this implementation technique. Moreover, these DSLs are mainly focused on the modeling of services instead of being devised to model context in an application-independent way. On the other hand, the COSMOS DSL is mainly aimed to express a low-level abstraction description of a component-based software architecture (i.e. how software components are connected) instead of modeling context.

Context-aware frameworks could benefit from a DSL-based approach. For example, in the case of OCP, a developer must be aware that there are minor differences when programming different versions of the middleware. Using a DSL is possible to ignore some particular aspects of specific versions of OCP, focusing only on context modeling aspects. JCAF illustrates another example of framework that could take advantage of a DSL. JCAF provides a Java API along with a set of generic implementations of Java classes as indicated above. Therefore, context information is represented at a low-level of abstractions by using the API to create Java objects. This difficult the development task regarding to the use of context models of higher level of abstraction, since Java code is harder to create and maintain than such models. Note that the JCAF context representation is tightly-coupled with the Java technology and it has a specific data stream formalization (Java serialization) for storing models. In contrast, our approach allows the context information to be represented at a high-level of abstraction by using our DSL instead of the JCAF API. JCAF code could be automatically generated from our DSL models as shown in Section 5.5.2.

Like JCAF, we have been able to automatically generate from our DSL models the OWL ontologies as well as the Java code for basic OCP producers and consumers, and software artifacts for COSMOS and SAMURAI, as shown in Sections 5.5.1 , 6.6.1 and 6.6.2.

QoC support and modeling

QoC has not been considered a critical issue on context modeling till recent years. So it is not surprising that only two of the reviewed approaches, COSMOS and SAMURAI, offer full support for computing it. However QoC requirements are not part of the context model in SAMURAI. As mentioned before, QoC is managed in SAMURAI by directly programming code as JavaScript code which can be used to define a context situation. COSMOS manages QoC data in the same way that does with context information. Therefore, QoC is processed by context node operators, and it is encapsulated into messages along with other context information, as explained in Section 3.2.6. These messages are sent through the context node hierarchy from the children nodes to the parent nodes. The architecture of nodes defines how a context situation is computed in COSMOS. ContextUML can also define context situations as composite context, and PervML as services which compute them, but without quality requirements.

QoC is represented in SOUPA in a rudimentary way. It is possible to define quality requirements but only related to accuracy, because SOUPA has an extension which can be used to describe a conditional belief based on accuracy values.

Ayed et al proposes an UML stereotype for modeling QoC, which can specify some quality parameters but do not justify why they have chosen these parameters instead of others. Though quality parameters can be specified, their approach does not let developers to model quality requirements for the application.

Our approach let us define QoC attributes as well as QoC requirements for a context-aware application. Moreover, it is possible to define quality levels for the quality requirements as explained in Section 6.1.2.

MDE approaches

Most of the approaches are specific for some framework or have a low-level of abstraction to be processed in an efficient way. However, MDE approaches like ContextUML or PervML offer a higher-level of abstraction. These approaches use model transformations to generate lower-level specifications and executable code.

PerVML and CAUCE are generative architectures for creating context-aware pervasive applications from several types of models. COSMOS aims to manage context information with QoC and it has a simple and low-level DSL. ContextUML is aimed at the design of context-aware web services, which includes constructs for modeling context information.

There are some incomplete MDE approaches. Ayed et al., propose an MDE method for developing context-aware systems but it is not implemented and there is no validation of their approach. Therefore there is no code generation and the authors state that it could be very difficult to write model-to-text transformations from their models, but they do not give any explanation. CAMEL is another incomplete approach. It currently only offers a metamodel for context-awareness, but it lacks of a concrete syntax.

The management of ontologies is a laborious task and it would be complex to create an MDE approach which makes use of a SOUPA ontology context model. The main obstacle in this case would be the difficulty for creating a mapping between context model elements because the context information in SOUPA is spread through several ontologies. However, on the contrary, it would be possible to define a mapping from other context models to SOUPA (e.g. it would be possible to write a transformation to create part of the ontology documents of SOUPA, like the user profile, from our approach).

All reviewed MDE approaches include target platform or application dependent aspects in the context model. Since our approach proposes to represent this information in a separated model, the models are thus more readable and closer to the users' conceptual domain. It is worth noting that these MDE approaches are used to generate code only for a specific framework. To the best of our knowledge, our approach is the only one which has been validated by generating software artifacts for several frameworks.

Table 3.2 shows a comparison of the different MDE approaches. This table contrasts how the involved modeling languages or DSLs have been implemented. We have considered as criteria the metamodeling language used to define the abstract syntax, how the notation or concrete syntax has been implemented, what type of model transformations have been used to implement the translational semantics, and the target platforms if code has been generated.

The abstract syntax of most approaches has been defined by using MOF or Ecore except COSMOS that includes the JULIA model, which is a Java implementation of the Fractal component model (Bruneton et al., 2006). The type of metamodel is normally imposed by the tool used to implement the DSL (e.g. EMF editor for CAMEL, ArgoUML for ContextUML, JULIA for COSMOS, Topcase for PervML and EMFText for MLContext) or by the type of selected approach: during some years MDA approaches were normally associated to MOF although the use of Ecore offered some advantages (e.g. a MOF or EMOF metamodels for MDA approaches like Ayed et al. or CAUCE).

There are several approaches which have a concrete syntax based on UML stereotypes or profiles. At the early years of MDE, this technique was commonly used but the experience evidenced that creating DSLs from scratch was more simple and required less effort as noted (Völter, 2009; Kelly et al., 2009).

Table 3.2 MDE approaches comparison.

	Abstract Syntax	Concrete Syntax	Semantic	Target platform
Ayed et al.	MOF	UML based	No	No
CAMEL	Ecore	No	No	No
CAUCE	EMOF	UML based	M2M / M2T	.NET framework(*)
ContextUML	MOF	UML based	M2T	jBPM-BPEL
COSMOS	Fractal component model / JULIA	COSMOS textual DSL	M2T	Fractal framework
PervML	Ecore	UML based	M2T	OSGi/Java code
MLContext	Ecore	MLContext textual DSL	M2T	JCAF, COSMOS, OCP, SAMURAI, Java code, GeoSPARQL

Actually, the UML metamodel is very large and complex and this makes difficult the use of the profile mechanism (Selic, 2012). On the other hand, CAMEL does not have a concrete syntax. Therefore CAMEL models are defined by directly creating instances from the metaclases using the EMF editor.

Most of the approaches define their semantic by using M2T transformations. CAUCE makes also use of M2M transformations for generating the models for the second layer of its approach. In MLContext, we have only implemented M2T transformations because the semantic gap between the source MLContext models and the generated code was not too large though we can make use of intermediate M2M transformations if necessary. Ayed et al. and CAMEL are incomplete MDE approaches, so they do not define a semantic and do not generate code for any framework.

Last column of Table 3.2 shows the target frameworks for which a transformation engine has been defined for the MDE approaches. CAUCE has been designed to generate code for different frameworks, but it has only been validated for the .NET framework. PervML has some similarities with CAUCE, but the authors do not mention if this approach could be used for generating code for non OSGi-based frameworks. COSMOS has been designed to specifically generate code for the Fractal framework.

Our approach is not targeted to specific middlewares. Currently, we have created MLContext transformation engines to generate code for JCAF, COSMOS, OCP, SAMURAI, Java code and GeoSPARQL.

4 Overview

A general review

We don't see things as they are.
We see things as we are.

Anais Nin

Since the advent of Context-aware systems, developers have confronted the challenging task of eliciting and representing every aspect of the context information that can be used to characterize the situation in which the system is involved. This task normally requires the specification of users, objects (e.g. buildings) and any entity that is necessary for the design of the system, along with the relationships between them, which must be taken into account when reasoning about the context. These entities sometimes need to be classified in a hierarchy of categories which must also be defined.

Entities have properties that must also be modeled if they are of interest for the system, and the developer must therefore confront the task of modeling different types of properties such as physical properties (e.g. the temperature of a room) or personal properties (e.g. the name of a user), among others. Note that some types of entities (e.g. users and computers) are capable of performing different tasks, which need to be represented if it is necessary for the system to know them in order to make decisions or produce a correct behavior.

Most of the values of context information that the system must know originate from different sources of context. Sources of context are mainly (typically) sensors that gather data on the environment, but context information can also be obtained from other sources (e.g. networks or by browsing user profiles). Since the system must be aware of the existence of these sources, they also need to be represented along with entities and properties. For each context source, the developer must model their characteristics, such as the API provided to retrieve the value of context information supplied or the name of the entity for which this value is supplied.

According to the rationale explained in Chapter 1, QoC must also be part of a context model. This is usually achieved by specifying quality parameters for the sources of context which supply the information. The quality of the context information managed can determine the system behavior. Therefore, and especially in critical systems, the system should only make a decision if it is based on context information with a minimum level of quality. In this case, the developer should be able to model the context situation involving the system decision, and the minimum quality requirements in order to take into account the related context information.

In short, the modeling of context involves the modeling of entities, their properties, sources of context and the quality of context. Once the developer has specified the context, s/he must specify other aspects related to adaptation, services, communications, etc., but they are not part of the context model.

The remainder of this chapter provides an overview of our approach and the implementation and the validation carried out. Section 4.1 explains the MLContext approach and introduces the MLContext DSL; Section 4.2 describes the tooling that provides the MLContext language with support. Finally, Section 4.3 shows how our approach has been evaluated.

4.1 Modeling context

As explained in previous chapters, we propose an MDE approach that can be used to represent context in the context-aware systems development process, in which models are created at different abstraction levels to represent aspects of the system, and model transformations are used to generate software artifacts.

4.1.1 MLContext DSL

The core element of our approach is a DSL named MLContext, which has been specially tailored to model context information and QoC. This DSL allows the creation of platform-independent context models which are used to automatically generate context management software artifacts for concrete context-aware middleware platforms.

MLContext has been designed to deal with the following three features:

1) to provide a high-level abstraction for the building of platform-independent models.
2) to be a language which is simple and easy to learn, signifying that final users can write and understand context models.
3) to promote the reuse of context models by separating the implementation dependent aspects from the domain aspects.

We carried out a domain analysis presented in Chapter 5, in order to elicit the requirements and design choices to be taken into account when creating our DSL. This domain analysis allowed us to define the metamodel of the MLContext abstract syntax.

MLContext has constructs with which to modeling the context elements mentioned above: entities, properties, categories, sources of context, quality attributes, quality requirements, and context situations and to define levels of quality.

An overview of our MDE approach is shown as follows.

4.1.2 MLContext approach overview

When a model-based approach is defined for context-aware applications, context models are not the only models required, since other models are also used to represent different aspects of the system. The purpose of each model and the relationships among them may vary depending on the proposal.

Tasks, goals and roles, among other aspects, are normally modeled as part of a domain model which also represents the concepts of the application domain (like an airport or a hospital). Furthermore, most existing approaches consider that the context models should not only represent the context information (i.e.information on the application entities), but also additional information such as situations, activities or sensor details (Sheng et al., 2005; Sindico et al., 2009). Figure 4.1 shows how a context model is one of the different aspects to be considered in a domain model of context-aware applications.

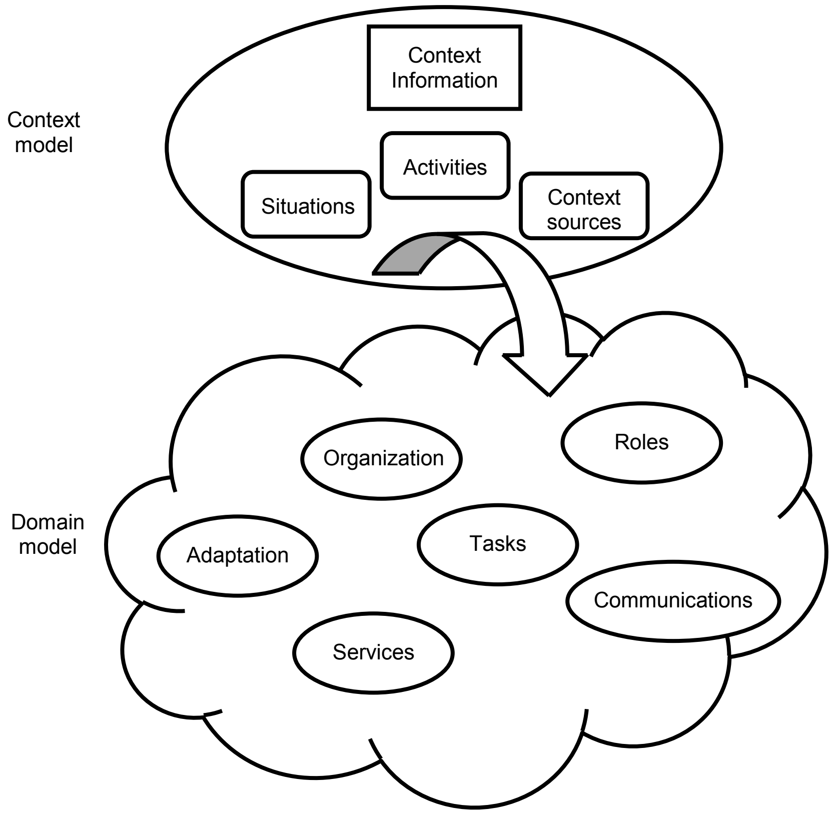


Figure 4.1 Domain model and context model with application details.

As indicated above, some information, which is normally included in the context models, is really specific to a particular application. These models cannot therefore be reused in other application scenarios. Since context model reuse is one of the requirements to be satisfied in our approach, we propose to remove the specific details of concrete applications from the context model as will be explained in Chapter 5. If we remove these details from the context model, any developer wishing to create the application domain model in its entirely will consequently also need to specify the technical details of the sensors and sources of context, and the quality parameters of these sources, among other information. The reliability and availability of the information depends on the quality of the sources of context. This information would not be part of the context model, since devices can be replaced with others with different characteristics, and should be specified in one separate application-specific model, which could not be reused by other applications.

This separation of concerns promotes reusability and keeps MLContext models simple and readable. The models also retain sufficient information to be useful as regards generating software artifacts related to the context management as shown in this work. Note that removing application details from the context model is appropriate for applications that use simple relationships (e.g. simple human-centric applications). However, in complex applications (such as device-centric or self-* aware systems), we must confront complex relationships. One example of this kind of relationships could be the "is near" relationship. The meaning of this relationship may vary depending on the domain of the application, signifying that it can be "a few kilometers" for a GPS car application or "less than fifty centimeters" for an automatic door opening mechanism. As the semantics of this relationship may vary depending on the application, it must be specified separately from the context model.

We have therefore organized the information typically included in a context model into two models: the application model and the context model, as shown in Figure 4.2. The context model represents the context information which includes the entities, their properties and the sources of context available for the context-aware application. The application model, meanwhile, represents those aspects related to activities and situations, along with information related to sensors (including technical details such as precision, resolution, etc.). When defining an application model it is necessary to refer to elements included in the context model. This approach also has some benefits as regard evolvability. For instance, we could make changes at the application level (e.g. one of the sensors is exchanged for another with a higher precision) and we would not therefore need to change the context model or the domain model.

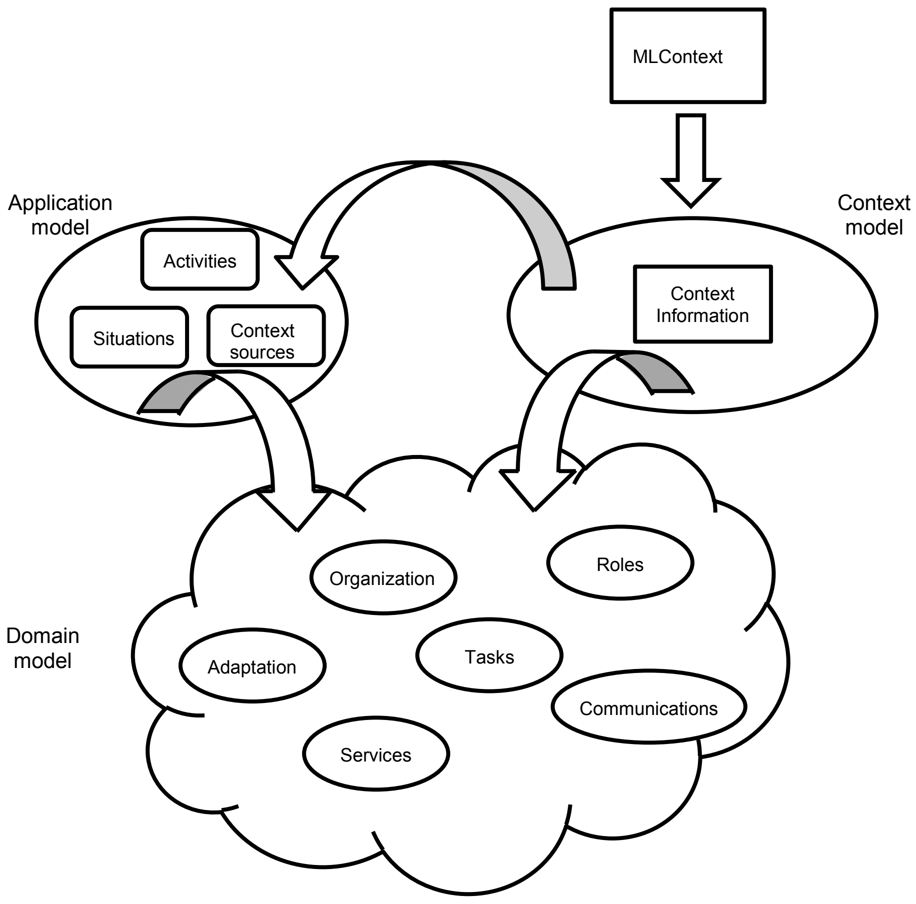


Figure 4.2 MLContext approach.

A method for context modeling

Our MDE approach includes a proposal for a simple method with which to model context with QoC in the earlier stages of the development process.

This method recommends the steps to be followed in context modeling, and will be discussed in detail in Section 6.4.

As noted in the previous section, the context information is represented by means of two models in MLContext: the context model and the application model. The context model should be created before the application model since the latter includes references to the former. We therefore propose the following steps:

1) Specify the entities, with their properties and simple relationships.
2) Specify the categories for the entities.
3) Specify the context sources.
4) Specify technical details and quality parameters for the sensors attached to the sources of context
5) Specify complex relationships and context situations.
6) Specify quality requirements.

The context model is created by following steps 1-3. The developer can then create application model in steps 4-6.

Once the context model and the application model have been created, they can be used to automatically generate software artifacts for the target middleware. Both models serve as input for the MLContext transformation engine, as explained in the following section.

4.2 Tooling for MLContext

The definition of a textual DSL involves the development of the tools needed to assist the language users. As indicated in Section 2.4, some basic tools are:

1) an editor with which to create DSL specifications
2) a parser (commonly named "model injector") that is capable of extracting models from a DSL textual specification.

3) a code generator which is able to transform the DSL models into software artifacts.

As indicated in Section 2.4.3 several DSL definition tools that automatically generate editors and model injectors from either the DSL metamodel or the DSL extended grammar currently exist. As explained previously, we have used EMFText to apply a metamodel-based approach. EMFText can use this metamodel to generate an editor which does not contain dependencies on this tool.

We have created an Ecore metamodel to represent context information and context quality, which has been used to define the abstract syntax of MLContext (see Section 5.2). EMFText was then used to define the concrete textual syntax by attaching the syntactic information (e.g. keywords) to the metamodel elements by means of a specification formed of a set of rules expressed in the CS language provided by the tool. From this specification, EMFText generated: (1) an Eclipse editor, which features syntax highlighting, an outline and hyperlinks, (2) a model injector in Java that takes a source specification expressed in its textual concrete syntax and generates a model conforming to the DSL metamodel, and (3) an extractor that performs the reverse operation to that of the injector, and generates textual specifications from the models.

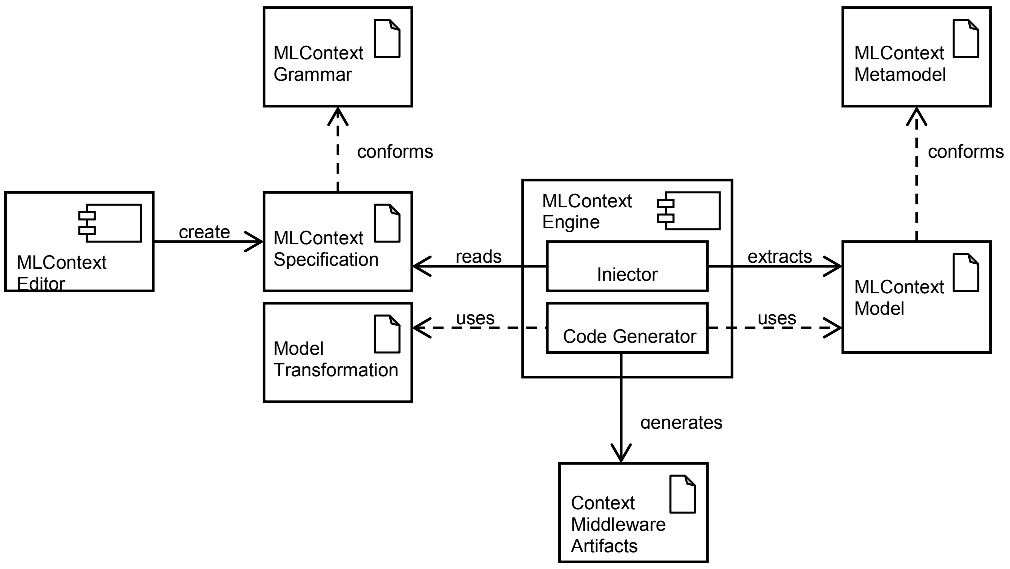


Figure 4.3 Tooling for MLContext.

Figure 4.3 shows a schema of the tooling for MLContext. Firstly, MLContext textual specifications are written by using the editor, then this text is converted into an MLContext model, by using the injector, and finally this model is transformed into a software artifact by using a model-to-text transformation. The metamodel, notation and transformations are explained in Chapters 5 and 6.

4.2.1 Transformation engine

The transformation engine is in charge of generating software artifacts for specific middleware by means of a transformation chain. It is therefore possible to generate code for any middleware for which a transformation engine can be created (i.e. we can define a mapping from the MLContext metamodel elements to the middleware elements).

In this thesis, M2T MOFScript transformations have been used to build a transformation engine as a one-step model transformation chain. This approach has several advantages:

- The developer only needs to create one transformation, which is easier to maintain.
- As only one model-to-text transformation is carried out, it is easier to update and synchronize the context model and the artifacts generated than in other approaches that use intermediary models, because the modifications are isolated and completely focused.
- The resulting transformation is faster than approaches which use intermediate models, because intermediate steps are not required.

However, this approach has one main drawback: the one-step transformation could be very complex if the semantic gap between the context model and the target artifacts generated is too wide. In order to solve this, we have created an MLContext API to help developers in writing transformations for MLContext models. This API provides direct access to any element of the context model and its properties, without the need to navigate through the model. Nevertheless, several M2M transformations can be used as intermediate steps to reduce the semantic gap.

4.3 Evaluation of our approach

This work has been validated from several dimensions according to the objectives of the thesis:

Learnability

A DSL must be easy to learn and to use. We measured some classical metrics which allowed us to provide a better characterization of the language and obtain an estimation of its complexity. These measures were calculated by analyzing the DSL grammar structure, as explained in Section 5.6, and were used to estimate the ease with which the language can be learnt and the effort needed to read and understand a context model specified with MLContext.

According to the values obtained, MLContext is much easier to learn and use than, for example, the Java or C general-purpose languages.

Improving productivity

We have measured the increase in productivity (Section 5.6) when using the MLContext DSL by calculating the effort saved when writing code for the final application.

The development effort was calculated as lines of code by measuring the difference between the lines of code of the DSL specification (i.e. the model) and the lines of code needed to manually write the artifacts that were generated automatically. The measurement takes into account the complexity of the language, as explained above.

This work started as part of the CADUCEO[2] project, whose principal objective was to optimize asset management in hospitals by improving the services with which the medical staff is provided. We therefore, used the

[2] CADUCEO Project (IPT-2011-1080-900000), INNPACTO Program, 2011 founded by Ministry of Economy and Competitiveness (Spain) and the European Regional Development Fund (ERDF).

example of a simplified case study of a hospital to represent the context of the application. We generated code for two middlewares: OCP and JCAF.

We later defined the QoC extension for MLContext and we evaluated the productivity improvement again by generating code for two middlewares with QoC support: SAMURAI and COSMOS. As the hospital example did not include QoC, we used a new case study which had already been used by the creators of COSMOS. This therefore allowed us to evaluate the gains of productivity provided by our approach when compared to those of COSMOS and SAMURAI. An estimation of the gains in productivity measured is shown in Section 6.7.

Portability and reusability

Our approach proposes the use of a generic platform-independent context model. An MLContext model could therefore be used in any middleware for which a transformation engine can be written. To prove that platform-specific models can be generated from MLContext models, we wrote transformation engines for several middlewares to generate different software artifacts (e.g. OWL ontologies, Java code, XMI code, JavaScript code, a COSMOS DSL specification, etc.) as explained in Chapters 5.5 and 6.6.

Integration with other tools

One of the characteristics that is desired for a DSL is that can be integrated with other tools. As MLContext is integrated into EMF, any Eclipse modeling tool can load and manage an MLContext model. For example, transformation languages (e.g. Acceleo, ATL), storage tools (e.g. EMFStore, Teneo), validators (e.g. EMF Validation Framework), etc. Other tools have also been integrated with MLContext: Protegé can open and manage OWL-DL ontologies generated from MLContext models, and GeoSPARQL can use the GeoSPAR specifications of MLContext. Java programs are able to read MLContext models through the use of an MLContext Java API.

Expressiveness

The concepts defined in the language must correspond to important domain concepts and each element of the language must represent only one domain concept. In this respect, MLContext has constructs for Entities, Sources of context, Context information, Categories and properties, as explained in Section 5.2.

Usability

A DSL must have tool support as regards creating and editing its specifications. We have created an editor, a parser (injector) and code generators, as explained in Section 4.2.

5 MLContext

A proposal for context modeling

A journey of a thousand miles
begins with a single step.

Lao Tzu

As explained in previous chapters, this work proposes an MDE approach for context management in context-aware systems. The core element of this approach is a DSL called MLContext, which has a high-level of abstraction and has been specially tailored for the modeling context. The creation of this DSL is one of the main objectives of this thesis.

MLContext has the typical benefits provided by DSLs: a higher level of abstraction, an increase in productivity and better communication between users and developers. MLContext is easy to learn and use, and promotes the writing of concise context specifications which are easy to read and understand, thus permitting non-programmers domain experts to validate them.

The creation of MLContext was principally motivated by the limitations of the previously proposed context modeling approaches, which were identified in Chapter 3. We have taken advantage of the maturity of language workbenches, and the EMFText tool in particular, to define a metamodel-based domain specific language.

Once the decision to create a DSL has been made, the stages of the construction process can be followed: domain analysis, language design, implementation and validation, as indicated in Section 2.4.3. This chapter provides a detailed explanation of how these stages have been applied in the case of MLContext from the domain analysis to the validation of the approach. Section 5.1 explains the design of MLContext, the domain analysis performed to elicit the main concepts and requirements for the language, and the design choices made. Section 5.2 presents the abstract syntax of MLContext. Section 5.3 shows the concrete syntax of MLContext. Section 5.4 introduces the case study used to validate our proposal. Section 5.5 explains the code generation from the MLContext model for two middlewares: OCP and JCAF. Finally, we validate our proposal in Section 5.6.

5.1 MLContext design

One of the main phases in the development of a domain-specific language is the domain analysis which aims to identify and describe the domain concepts and their properties. The domain knowledge is gathered from sources such as technical documents, source code or domain experts. In our case, we have analyzed the most relevant approaches and frameworks dealing with context-awareness and context representation. Next, we show the conclusions of the domain analysis, the DSL requirements elicited from such analysis and the choices that we made in the design of the language.

5.1.1 Domain analysis

In building a DSL with which to model context, one of the main obstacles is the lack of a commonly accepted definition of what context really is. As shown in Chapter 2, dozens of context definitions can be found in literature, each of which differs in what information is actually part of a context. The widely used definition of Dey and Abowd (Dey et al., 2000), considers that context information is "any information that can be used to characterize the situation of any entity". An entity could be a person, a place, a physical or abstract object, or an event that is considered to be relevant to the interaction between a user and an application, including the user himself. For instance, a patient and a hospital room are some of the entities in the example shown in this chapter.

This idea of linking context information to an entity or subject is a key aspect of our approach. When entities are modeled, they can be classified in a hierarchy of categories of entities. For example, in the context of a hospital, patients and doctors are persons, and doctors are also hospital employees.

We must take into account that the information which is part of a context may come from several sources (e.g. a sensor, a tablet or smartphone, a computer system), which may use different formats to represent the information they provide. For example, a person′s location could be expressed as address information, latitude-longitude coordinates, the ID of the location or in other ways. As indicated in (Henricksen et al., 2002), there is usually a significant gap between sensor output and the level of abstraction of the information that is useful to applications, and this gap may be bridged by various types of context information processing. For example, a location sensor may supply raw coordinates, whereas an application might be interested in the identity of the building or room a user is in.

Applications usually represent the context information as a set of designer-defined descriptors (e.g. numerical values of time or device IDs). While this format might be appropriate for developers, it could be difficult for the users (who are used to handling higher-level semantic descriptors) to understand and use. Moreover, requirements may vary between applications. For these reasons, the contextual information should be modeled at a high level of abstraction, near the user-level, as is stated in (Vildjiounaite et al., 2007).

When modeling context, it is necessary to distinguish among different types of context information, so that a taxonomy is useful to appropriately model the concepts. However, most existing approaches do not define such a taxonomy. Instead, they define a single type of context, no matter what type of context information they are dealing with (Ayed et al., 2007; Buchholz et al., 2004b; Henricksen et al., 2002; Riva, 2006; Sheng et al., 2005; Vildjiounaite et al., 2007). Some taxonomies have been proposed such as (Ardissono et al., 2007; Chen et al., 2000; Göker et al, 2002; Hofer et al., 2002; Pires et al., 2005; Schmidt, 2005), and when they are compared, the following is found. Some types of context, such as the social context or the physical context, are common to nearly all the proposals, while others like the computational context only appear in some of them. This is because most of these classifications are focused on specific domains. For example, Ardissono et al. (Ardissono et al., 2007), make their classification by bearing in mind the management of contextaware web services based workflow systems, and Schmidt (Schmidt, 2005) only considers contexts that are relative to e-learning systems in order to consider the learner's situation in an appropriate manner. Moreover, the same information is considered to be of different context types, such as the role of a user in (Göker et al, 2002) and (Pires et al., 2005), or the bandwidth consumption cost in (Chen et al., 2000; Göker et al, 2002; Pires et al., 2005). On the other hand, some contextual information should be modeled as a state of the context rather than as part of it, e.g. the "safety level of a scenario" context type defined in (Alarcón et al., 2005) should be represented as a set of rules defining a range of temperatures or a level of pollution in a physical context.

A context model should normally include information on spatial and temporal aspects (Göker et al, 2002). Spatial information describes concepts such as location, direction or places. With regard to temporal aspects, context information can be characterized as being static or dynamic. The value of static information remains unchanged over time whereas the value of dynamic information may vary depending on the time instant it is requested.

Two important observations are made in (Henricksen et al., 2002). The first is that "context information is imperfect" and hence the system must therefore assure (insofar as possible) information reliability and availability. These two properties can not be really assured by the system but they should be represented in the context models to be able to reason about the levels of availability and reliability at any given time. For instance, the accuracy of the information (i.e. the distance to the true value) normally depends on the sensors' accuracy and it should be modeled with them. Then, this information could be used to choose the source of information with the required accuracy when more than one is available for the same context information. The second observation is that "context information is highly interrelated" because complex information could be composed of several simple contexts. Furthermore, the context of an entity often refers to the other entities' contexts (for example, an entity formed of other entities).

Other features of a context model such as the user profile and the context history (Hong et al., 2009) are normally considered to be optional. The user profile is a collection of personal data associated with a specific user. This information can be exploited by systems taking into account the users' characteristics and preferences. Context history becomes necessary when information is measured over time and it is necessary to keep a trace of its values in the time dimension.

Finally, note that contextual information is one of the aspects to be considered when creating the domain model of a context-aware application. Other aspects would be related to services, system adaptation, sensors, communications or security among others. This analysis is focused on the modeling of the context and these other aspects are not considered. We use the "context model" term to refer the model representing the entities and context sources which are part of a particular domain, as well as the entity–entity and entity–source relationships.

5.1.2 DSL requirements

The MLContext requirements have been identified from the performed domain analysis and our experience using the OCP platform. We have identified therefore the following requirements:

- *High-level abstraction*. The language should provide a high-level abstraction by means of constructs that are close to the domain concepts (e.g. entity, context or type of context). This would thus encourage users who are not developers to participate in context modeling.
- *Types of context*. Instead of having only a single type of context, the language should support the ability to model different types of context, such as physical, social or computational context, because

this is closer to reality. Moreover, separating information in multiple contexts can be exploited to achieve improvements such as more efficient storage and retrieval methods (i.e. an application might be only interested in context information of the physical type).

- *Platform-independent models*. The language should free the user from providing implementation details. Since both the context of an entity and the source from which context information is obtained can vary, a language with which to model context should allow the declarations of entities and sources to be separated in a platform-independent model.
- *Application domain independent and model reuse*. The language should promote the reuse of models. Since contexts have many similarities in different context-aware applications, MLContext should allow contexts to be modeled regardless of the context-aware application.
- *Traceability, quality and temporal mechanisms*. The language should provide a traceability mechanism to support historic information, a mechanism for representing the quality of the supplied information, and it should support both static and dynamic temporal information.
- *Ease of use*. Finally, the language should be easy and intuitive to use.

5.1.3 Design choices

In order to satisfy these requirements, we have made some design decisions, which are discussed below. As output of the domain analysis task, we have also established the concepts and relationships between them, which have been used to build the metamodel of the abstract syntax of the language, which is described later. The three main concepts around which the definition of the language is organized are entity, context and source of context, and a context can also be composed of other related contexts.

Unlike other approaches, the modeling task in MLContext is centered on entities rather than categories. That is, in the specification of a context, we do not model categories of entities (e.g. a patient) but rather specific instances (e.g. patient "Burt Holmes"). This is mainly because context sources are linked to entities rather than categories. In fact, two entities from the same category may have different context sources for the same property. Another reason is that the developer must give detailed information about each entity when instances of a category are created. The user groups different entities under a common name (i.e. the name of the category), then MLContext automatically creates a category with that name and infers its properties based on the properties of the grouped entities, excluding inherited properties from other categories. Changes in the properties of the entities will be automatically propagated to their categories. Therefore, the developer does not have to specify the properties of the categories. This clearly saves time and effort.

It may sometimes be necessary to explicitly define a category, as when dealing with legacy systems in which the number of entities may be very high and the information about the entities can be obtained from a database or repository to create the instances. In these situations, the developer should specify a generic entity (an entity with no specific values for its attributes), and assign it to the category, so MLContext can generate the category based on the properties of this entity. In the case of a legacy database, a migration application would generate each of the instances of this category from the database information.

As part of the domain analysis described in Section 5.1.1, we surveyed several proposals of context taxonomies. This review allowed us to define a taxonomy for our approach, which includes the most commonly used types of context. Table 5.1 shows the correspondence between the types of context chosen for MLContext and those proposed in other approaches. The top row of the table shows the types of context of our taxonomy organized in columns. The other rows show the types included in the different context taxonomy proposals.

The columns of each row are organized to show how the types of the taxonomy correspond to the types of our proposal (top row). Note that the proposals differ in the types of context and also in their meanings. Sometimes, the context information of one context type corresponds to more than one type in our proposal. An (*) in a cell of the table indicates that the context information corresponds to several types of context in our taxonomy. Next, we indicate the context information for each one of the types included in the proposed taxonomy.

Table 5.1 Taxonomy of context.

	Physical	Environment	Computational	Personal	Social	Task
Hofer et al.	(*) physical + logical				logical	
Schmidt		(*) social		(*) personal	social + organizational	organizational
Chen et al.	physical + time	(*) user + computing	computing	user		
Alarcon et al.	physical	(*) task	computational + interaction +tecnology	social		task
Goker et al.	environment + spatio temporal + personal			personal	social	task
Pires et al.	(*) spatio-temporal	(*) spatio-temporal +social		physiological + preferences	(*) spatio-temporal	spatio-temporal

- *Physical*: This includes all physical magnitudes (e.g. time, speed, temperature, light level, and noise level).
- *Environment*: A description of the physical space distribution by providing information on the people and objects surrounding the user (e.g. distances, user′s location and objects within other objects).
- *Computational* (System): This provides information related to the software and hardware of computer systems (e.g. network traffic conditions, hardware status, information accessed by the user and memory requirements).
- *Personal*: This describes the user′s profile and his/her psychological state (e.g. competences, preferences, age, gender and clothes he/she is wearing).
- *Social*: This describes the social aspects of a user, region or place, (e.g. law, friends, enemies, neighbors, co-workers, relatives and role).
- *Task*: This provides information about the different tasks the user (or other entity) can perform (e.g. activities that must be carried out, activities that the user is currently performing or tasks that the user is able to perform).

The designer can use these context types to construct models which are closer to reality than the commonly used representations for context information.

Since entities are the first class elements, in MLContext, and a context is always linked to an entity, greater granularity is achieved in our approach than in category-centered proposals. The context of an entity is generally composed of several related contexts of different types, and a distinction can therefore be made between the complex context (the parent context) and simple contexts (the child contexts) of which it is composed. Simple contexts are formed of a set of contextual information of the same type and can refer to other entities. The information gathered in simple contexts may come from different sources. Naturally, as sources are usually a set of devices, and are potentially heterogeneous, each source can have a different information format. In MLContext models, the source descriptions are separate elements in order to offer users a high level of abstraction. In this way, context information is independent from both the sources of context and the format of raw data retrieved from sensors.

The language allows us to express whether the information is static or dynamic, whether or not it must be registered in the trace, or the accuracy of the information supplied by the sources.

MLContext models do not contain platform or application-dependent details in order to promote the reuse of context models in different context-aware applications, as explained in Section 4.1.2. For example, a complete and detailed context model of an art museum could include information about the location of surveillance cameras, presence detectors and fire detectors. It could also include the spatial distribution of rooms, location of emergency exits and a complete profile of every art object on display in the museum. By applying MLContext, the same context model could be used in different applications as a surveillance system, which can trigger an alarm if an intruder is detected at night; an emergency system, which can detect the presence of fire and guide the people to the nearest emergency exit; a guided-tour system, which can guide visitors through the museum and explain each object of art they are looking at.

MLContext has been defined as a textual DSL whose specifications (i.e. models) are readable and easy to learn for domain expert users who are not experienced in modeling languages such as UML, as will be shown later in

Section 5.6. It is applicable to any context-aware application since it only includes the basic concepts for context modeling. It is worth noting that even when a graphical notation is appropriate for a DSL, the creation of a textual DSL is recommended as a first step since the required effort is, in this case, considerably smaller and a better understanding of the problem is acquired during the implementation (Völter, 2009).

5.2 Abstract syntax

The MLContext metamodel is shown in Figure 5.1. It has been designed around the concept of entity represented by the *Entity metaclass*, and all context information always refers to an entity as explained above. A complex context (*ComplexContext metaclass*) is an aggregation of simple contexts (*SimpleContext metaclass*) of different types (e.g. physical and social). For example, the context of a person can be an aggregation of a personal context (e.g. name and age), a physical context (e.g. temperature and pulse) and a social context (e.g. role patient). A simple context is formed of one or more pieces of context information (*ContextInformation metaclass*) of the same type (e.g. temperature and pulse are physical information).

There are several types of simple contexts according to the defined taxonomy (e.g. *physical* and *personal*). The context information of a simple context can contain a reference to other entities of the model (e.g. the complex context of one floor of the hospital can contain a simple context with environmental information referring to the rooms it is composed of). This is represented by the *include* relationship in the metamodel.

The value of a *ContextInformation* can be a constant or can come from a source of context (*ContextSource metaclass*) as indicated in the *fromSource* relationship. The information from the source is supplied through a method (*Method metaclass*) from the physical device interface. The same source can supply context information to several entities and can use a different method for each of them.

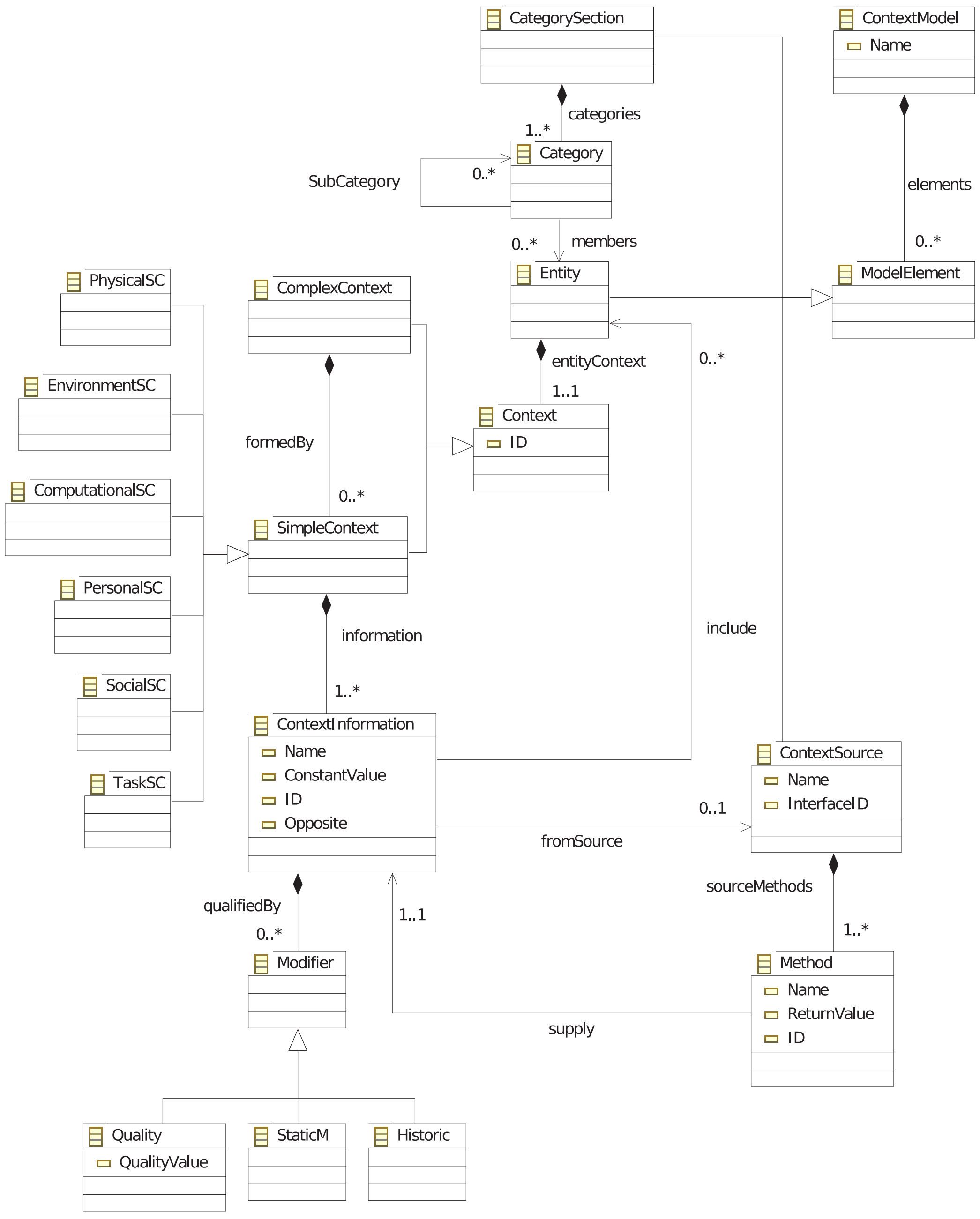


Figure 5.1 MLContext's abstract syntax metamodel.

The *Modifier* hierarchy represents the three optional modifiers for the context information: quality (*Quality metaclass*) to specify the accuracy of the information, historic (*Historic metaclass*) to keep trace of the information, and static (*Static metaclass*) to specify that the information does not change over time (if this is omitted, then dynamic information is assumed).

The CategorySection metaclass represents the set of categories of the model. Each category (Category metaclass) contains all the entities belonging to that category, and can also be a subcategory. An entity or subcategory can belong to more than one category (i.e. multiple classification).

As explained in Chapter 2, MLContext's abstract syntax has been modeled using the Ecore metamodel language supported by the Eclipse Modeling Framework (EMF) (Steimberg et al., 2008).

5.3 Concrete syntax

A textual concrete syntax has been created for the previous metamodel which can be used to create MLContext models. Figure 5.2 shows the CS specification which defines the syntax.

We have defined one rule for each element of the metamodel which has a textual representation. As we can see, a context model is defined by using the "contextModel" keyword, followed by the name of the model, and contains the definitions for all of the elements of the model.

This grammar is also annotated with some values that will be used by the printer to format the output file when generating text from the models.

```
SYNTAXDEF mlctx
FOR <http://www.modelum.es/MLContext>
START ContextModel

TOKENS{
           DEFINE COMMENT$'//'(~('\n'|'\r'))*$;
           DEFINE INTEGER$('-')?('1'..'9')('0'..'9')*|'0'$;
           DEFINE FLOAT$('-')?(('1'..'9') ('0'..'9')* | '0') '.' ('0'..'9')+ $;
           DEFINE REFERENCE$('A'..'Z'|'a'..'z')
           ('A'..'Z'|'a'..'z'|'0'..'9'|'_'|'-')* '.' (('A'..'Z'|'a'..'z')
           ('A'..'Z'|'a'..'z'|'0'..'9'|'_'|'-')*)?$;
           DEFINE NULO$'nullContext'$;
}

RULES{

   ContextModel::= "contextModel"   Name[] "{" !0 elements*  "}" !0  ;

   Entity::= #2 "entity"  Name[] "context" "{" !0 entityContext !0 #2 "}" !0  ;

   ComplexContext::=  formedBy*  ;

   PhysicalSC::=  #5 "physical"  "{" !0  information+ #5 "}"  ;

   EnvironmentSC::= #5 "environment"  "{" !0 information+ #5 "}"  ;

   ComputationalSC::= #5 "computational"  "{" !0 information+ #5 "}"  ;

   PersonalSC::= #5 "personal"  "{" !0 information+ #5 "}"  ;

   SocialSC::= #5 "social"  "{" !0 information+ #5 "}"  ;

   TaskSC::= #5 "task"  "{" !0 information+ #5 "}"  ;

   ContextInformation::= #8 Name['"','"'] #1 (((":" #1 ConstantValue['"','"'])
        |(HasSource["source" : ""])) | (("(" Opposite['"','"'] ")")? ":"
        #1 include[] (#0 "," #1  include[])* )) qualifiedBy* !0 ;

   ContextSource::= #2 "contextSource" Name[] "{" !0 #5 "interfaceID"  ":"
   InterfaceID['"','"'] !0 sourceMethods+ #2 "}" !0  ;

   Method::= #5 "methodName" ":" Name['"','"'] "{" !0 #8 "supply" ":"
   ("nullContext"|(supply[REFERENCE] ("," supply[REFERENCE])*)) !0 #8
   "returnValue" ":" ReturnValue['"','"'] !0 #5 "}" !0 ;

   Quality::= "quality" ":" QualityValue['"','"'] ;

   Static::= "static" ;

   Historic::= "historic" ;

   CategorySection::= #2 "categories"  "{" !0 categories+ #2 "}" !0  ;

   Category::= #5 Name[] ":" (members[] ("," members[])*)? !0  ;

}
```

Figure 5.2 MLContext concrete syntax specification.

5.4 Case study

In order to show the generation capabilities of the MLContext language, we have chosen an example based on a simplified case study of a hospital, in which doctors, patients and the hospital equipment must be taken into consideration to represent the context of the application.

Let us suppose that we need to develop a ubiquitous computing-based application for the management of the daily tasks of both doctors and assistant personnel (e.g. nurses and assistants). Let us also suppose that patients are identified by RFID labels attached to their personal health report. Moreover, personnel at the hospital are equipped with either smartphones or tablets with mobile communication capabilities. In this scenario, a typical ubiquitous service quickly allows the doctor to automatically access information about a patient he is dealing with by using both doctor and patient context location.

The root entity is the hospital, which is formed of floors and people. Each floor is formed of rooms and has a fire alarm with a fire presence detector. For the sake of simplicity, we have represented only two floors and two rooms. *Room01* contains a bed, a television, a temperature sensor and equipment, which can be attached to a patient in order to monitor his/her vital signs. While doctors (e.g. *person01*) are part of the hospital itself, patients (e.g. *Person02*) are located on beds. MLContext allows us to use any identifier we wish, but entity and source identifiers must be unique in the model.

A declaration must be written for each model element. Since the *hospitalUMU* entity is formed of two floor entities, the context information of the hospital must include the context information of each of its floors. The following declaration shows that *hospitalUMU* has a complex context formed of an environment context, which only has a "contains" property, which expresses that the hospital contains two floors (*floor01* and *floor02*). These entities must be specified later in the model.

```
entity HospitalUMU context {
   environment { "contains" : floor01 , floor02 static }
}
```

The "contains" property references the entities floor01 and floor02. The modifier static indicates that this information does not change its value. Note that doctors are not included in the declaration of any room because they can change their locations from one room to another in the model. They are not attached to a specific room or floor but they are in the model. Each floor could be specified as follows.

```
entity floor01 context {
   environment { "contains" ("locatedIn") : room01, room02 static }
   physical    { "fire_presence" source }
}
```

The specification of the *floor01* entity shows a complex context formed of two simple contexts of the *environment* and *physical* types. The environment information expresses that the floor contains two rooms and the physical information expresses that it has a "fire_presence" property. The *source* keyword indicates that the value of this property comes from a source of context which needs to be declared later in the model (a fire alarm).

Note that, in this case, the "contains" property is followed by the name of a new relationship ("locatedIn"). This is an abbreviated form of defining opposite relationships. As a result, a property named "locatedIn" which references the entity *floor01* is defined in the entities *room01* and *room02*. It would be now necessary to model each of the rooms in the hospital.

In the *room01*, there are a bed named *bed01* and a television named *tv01*.

```
entity room01 context {
   environment { "contains" : bed01 , tv01 static }
   physical    { "temperature" source }
}
```

We are interested in knowing the temperature of the room in order to activate the central heating if the room becomes too cold. This is also context information of a *physical* type that we can obtain from a source (which we will model later).

If there is a high number of rooms with the same characteristics, in order to avoid modeling all of them, we should only model one generic entity "rooms" and specify that the floors contains "rooms" as stated in Section 5.1.3, but this is not the case in this example. We might also be interested in knowing if the room's television is turned on.

```
entity tv01 context {
   computational { "status" source }
}
```

The context of the television is of the *computational* type. The status of the television can be on or off (we suppose it can be sensed, so we can obtain this information from a source of context). Now, let us suppose that there is a patient *person02* in *bed01*.

```
entity bed01 context {
   environment { "contains" : person02 } }

entity person02 context {
   personal  { "name"    : "John"  static
               "surname" : "Smith" static
               "age"     : "39" }
   social    { "role"    : "patient" }
   physical  { "temperature" source historic
               "pulse"       source } }
```

The *person02* entity has a lot of context information which is useful in our modeling example. This entity would have a complex context formed of the aggregation of *personal*, *social* and *physical* contexts. The *personal* context refers to the profile of the patient which is context information of the *personal* type. Note that there is no context source associated with the "name", "surname" or "age". This is considered to be information provided by the user.

As indicated in the social context information section, the "role" this person is playing in the hospital is "patient". Other people can play other roles such us "visitor" or "doctor". To monitor the state of the patient, we need to know his "temperature" and "pulse". This information is enclosed in a *physical* context. We assume that the patient is connected to an *equip01* source, which can provide us with the required context information. Note that, in this case, we will associate the same source with two different pieces of physical context data as we will see in the *equip01* source definition.

While the patient is in *bed01*, the source *equip01* could be associated with the physical equipment in room *room01* containing this bed, but if the patient is moved to *room02* the source *equip01* will be attached to the physical monitor of this new room. The binding between an entity and its context sources is expressed in the definitions of the sources, as will be shown later. We have specified that "temperature" is contextual information of the *historic* type because we are interested in keeping a trace of its values over the time.

The declaration for a *doctor* entity provides an example of *task* context.

```
entity person01 context {
   personal { "name"    : "Bob"      static
              "surname" : "Steward" static }
   social   { "role"    : "doctor" }
   task     { "current" : "none"
              "task_01" : "patients_attending"
              "task_02" : "surgery" } }
```

In this case, the *task* context provides us with information about the tasks that this person can perform and the task he is currently performing ("none" in the above example). Notice that we are specifying at design-time a property whose value will be changed at run-time by the middleware.

Once we have modeled all the entities, we have to specify the context sources, which correspond with the source elements, in order to establish a match between each source and its physical device. For the sake of space we only show one example of a source definition.

```
contextSource equip01 {
   interfaceID   : "AdapterX670"
      methodName : "getTemp" {
                   supply     : person02.temperature
                   returnValue: "float"
      }
      methodName : "getPulse" {
                   supply      : person02.pulse
                   returnValue : "float"
      }
}
```

The *interfaceID* keyword refers to the identification of the physical device adapter (e.g. an X10 standard address), which will provide us with the desired information. This source provides values for the temperature and pulse properties of the *person02* entity.

The *methodName* keyword refers to the method to be invoked to retrieve information from the adapter, and the *returnValue* to the type of the information returned. These methods provide the interface for accessing the source independently of the actual interface of the source (i.e. the Adapter design pattern is applied).

In order to specify the categories to which entities belong, it is necessary to add a categories section to our model as follows.

```
categories {
   Hospital : HospitalUMU
   Floor : floor01 , floor02
   Room : room01, room02
   Bed : bed01
   Person : person02, Doctor
   Doctor : person01
}
```

Note that we could specify the category in each entity definition but we have decided to put all the definitions together instead. A change in the name of a category does not, therefore, force us to change it for each of the entities. Moreover, we can use the same declaration to discover all the entities belonging to one category rather than searching through the whole model.

The aforementioned definition states that the *hospitalUMU* entity belongs to the *Hospital* category, the *floor01* and *floor02* entities belong to the *Floor* category, and so on. A category can have subcategories resulting in a category hierarchy. In the previous example the *Doctor* category is a subcategory of the *Person* category.

We do not need to specify the properties for each category because MLContext can infer them from its entities (the union of the properties of the entities belonging to that category). In MLContext we can define relationships between entities by defining properties whose values are also entities. MLContext automatically establishes relationships between categories based on the relationships of their entities. Note that it is possible to explicitly define a category as we explained in Section 5.1.3.

MLContext takes into account the properties inherited from other categories (for example the *Doctor* category will have only the "task" properties included on *person01* because *Doctor* is a subcategory of *Person* and the "name", "surname" and "role" properties are inherited from it). A property in a category indicates that a member of that category could exhibit the property, but this is not mandatory. When generating code, the data-type of a property is determined by its source of context or by analyzing its constant value. In the case of a property name collision (two entities which have the same property but different data-type under the same category), a warning is shown so that the user can make the appropriate decisions.

When an MLContext model is compiled an executed to generate software artifacts, if the designer did not specify a category for some of the entities, a warning message is shown and the categories are automatically created from the name of the entities. For example, a *C_tv01* category would be created for entity *tv01* in our model because we have not specified it.

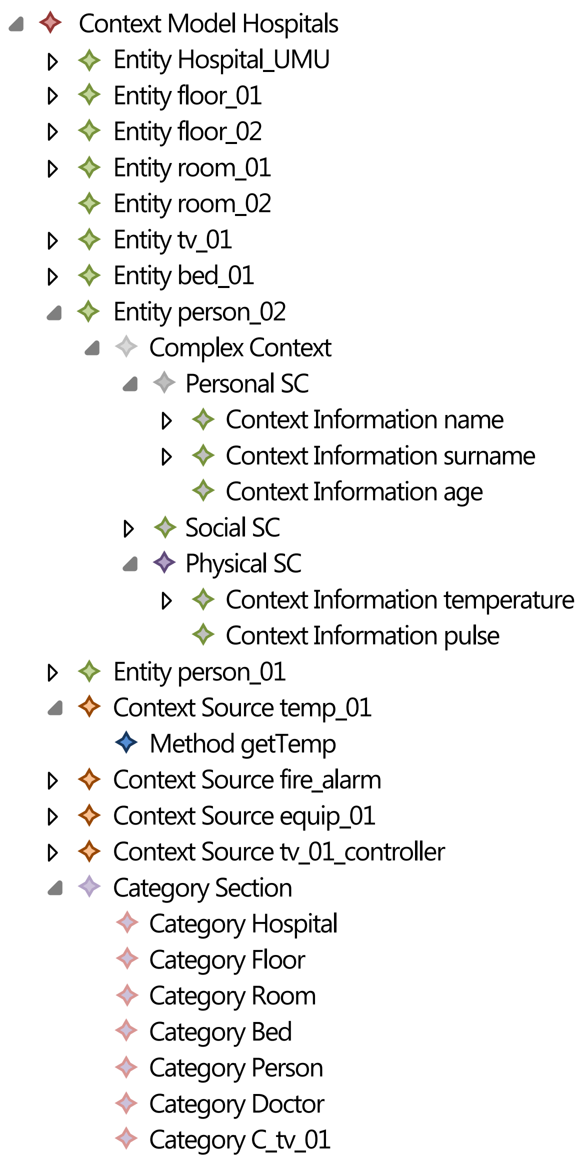
Figure 5.3 shows an excerpt of the outline for the MLContext context model for the Hospital example.

5.5 Code generation

As we previously mentioned, MLContext is a DSL specially designed to be used following an MDD approach. A DSL engine normally includes model-to-text transformations which convert the DSL models into software artifacts of the final application. When the semantic gap between the source model and the target artifact is high, one or more intermediate model-to-model transformations can be applied to implement the transformation process in several steps. In our case, MLContext models have been converted into software artifacts using only model-to-text transformations, since we have not encountered great difficulty in the mapping between MLContext and code (e.g. Java code or ontologies). Moreover, we have created an API to facilitate writing model-to-text transformations for MLContext models.

The MLContext models represent the context information at a high-level of abstraction and they are independent of any concrete middleware. With regard to the formalisms provided by existing context-aware middleware for representing context information, MLContext makes modeling context easier to developers.

An MLContext engine (also known as code generator) generates therefore the context representation used by a concrete middleware (e.g. an ontology in OCP and Java classes in JCAF. The OCP and JCAF middleware platforms were explained in Section 3.1.



- Context Model Hospitals
  - Entity Hospital_UMU
  - Entity floor_01
  - Entity floor_02
  - Entity room_01
  - Entity room_02
  - Entity tv_01
  - Entity bed_01
  - Entity person_02
    - Complex Context
      - Personal SC
        - Context Information name
        - Context Information surname
        - Context Information age
      - Social SC
      - Physical SC
        - Context Information temperature
        - Context Information pulse
  - Entity person_01
  - Context Source temp_01
    - Method getTemp
  - Context Source fire_alarm
  - Context Source equip_01
  - Context Source tv_01_controller
  - Category Section
    - Category Hospital
    - Category Floor
    - Category Room
    - Category Bed
    - Category Person
    - Category Doctor
    - Category C_tv_01

Figure 5.3 An excerpt of the context model for the hospital example.

In addition, other software artifacts required by a middleware (e.g.Java code of producers in OCP), could be derived from MLContext models. As indicated above, an MLContext engine for a particular middleware is composed of several model-to-text transformations, one for each kind of target artifact.

Since MLContext models are independent of the platform, the same model could be used in different middleware platforms to generate software artifacts. Naturally, this requires the existence of an MLContext engine for a particular middleware. For instance, we have generated OCP and JCAF artifacts from the same MLContext model. On the other hand, MLContext models could be reused for different applications on the same platform. For instance, the MLContext model for the hospital example could be reused in different applications by combining it to specific application models as explained in Section 4.1.2. In addition, artifacts generated as an ontology are useful for applications based on the same context.

It should be noted that the code generation will be limited by the capabilities of the target platforms. For example, the Context Toolkit middleware (Dey, 2000) does not have any representation for entities or categories, so we can not generate any code to manage these concepts.

We have chosen OCP and JCAF middleware in order to assess the potential of MLContext in the generation of code. OCP is a framework we are developing for adaptive applications, and JCAF is a well-known Java-based middleware. Figure 5.4 shows how a separate code generator has been created for each middleware.

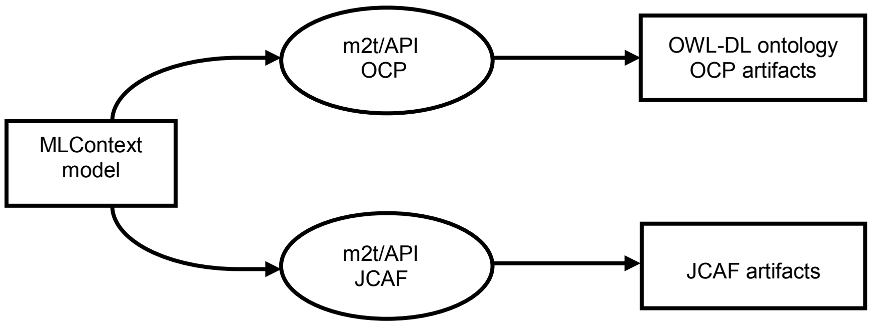


Figure 5.4 Code generation from MLContext models.

To facilitate the implementation of MLContext engines, we have created a MOFScript API which simplifies the writing of model-to-text transformations by avoiding a complex navigation of the MLContext models. In the following sections we will explain briefly the main characteristics of the OCP and JCAF middleware platforms and then, we will show how we have automatically generated code for them.

5.5.1 Generating code for the OCP middleware

The transformation from an MLContext source model to an OWL ontology is carried out in four steps. Each one of the steps corresponds to a mapping between an MLContext element and an ontology element:

1) *Generate ontology headers*. In this step, a common XML header and a set of entity definitions are generated as part of a document type declaration (DTD). These entities are used to define the RDF namespaces of the ontology that will be used as part of the XML tags in the next step. The *name of the ontology* is created from the *name of the MLContext model*. A comment, a label for the ontology and some OCP imports are also generated in this phase.
2) *Generate ontology classes*. The OWL classes are generated from the categories of the MLContext model, specifying subclasses when necessary and bearing in mind the inheritance hierarchy. An OWL class is generated for each category.
3) *Generate ontology datatype properties*. A *DatatypeProperty* is generated for each property of the *model's categories*. The XML Schema datatype of the property is obtained from the return value of the source method that supplies the property value, or is inferred from the constant value of the property if it has no source of context. At this moment, the supported types are: *string*, *boolean*, *integer* and *float*. The domain of the property will be one of the OWL classes generated in the previous step.
4) *Generate ontology object properties*. When a property in the MLContext model refers to other entities (a *relationship*), an OWL *ObjectProperty* is generated, rather than a *DatatypeProperty*. The domain of this property is the OWL class generated from the MLContext category containing it or, if the property is present in more than one category, the union of these classes. The range of the *ObjectProperty* is the union of all the classes from each entity belonging to the relationship.

Finally, we close all labels and include some comments in the generated ontology. Figure 5.5 shows a partial view of the ontology generated for the running example.

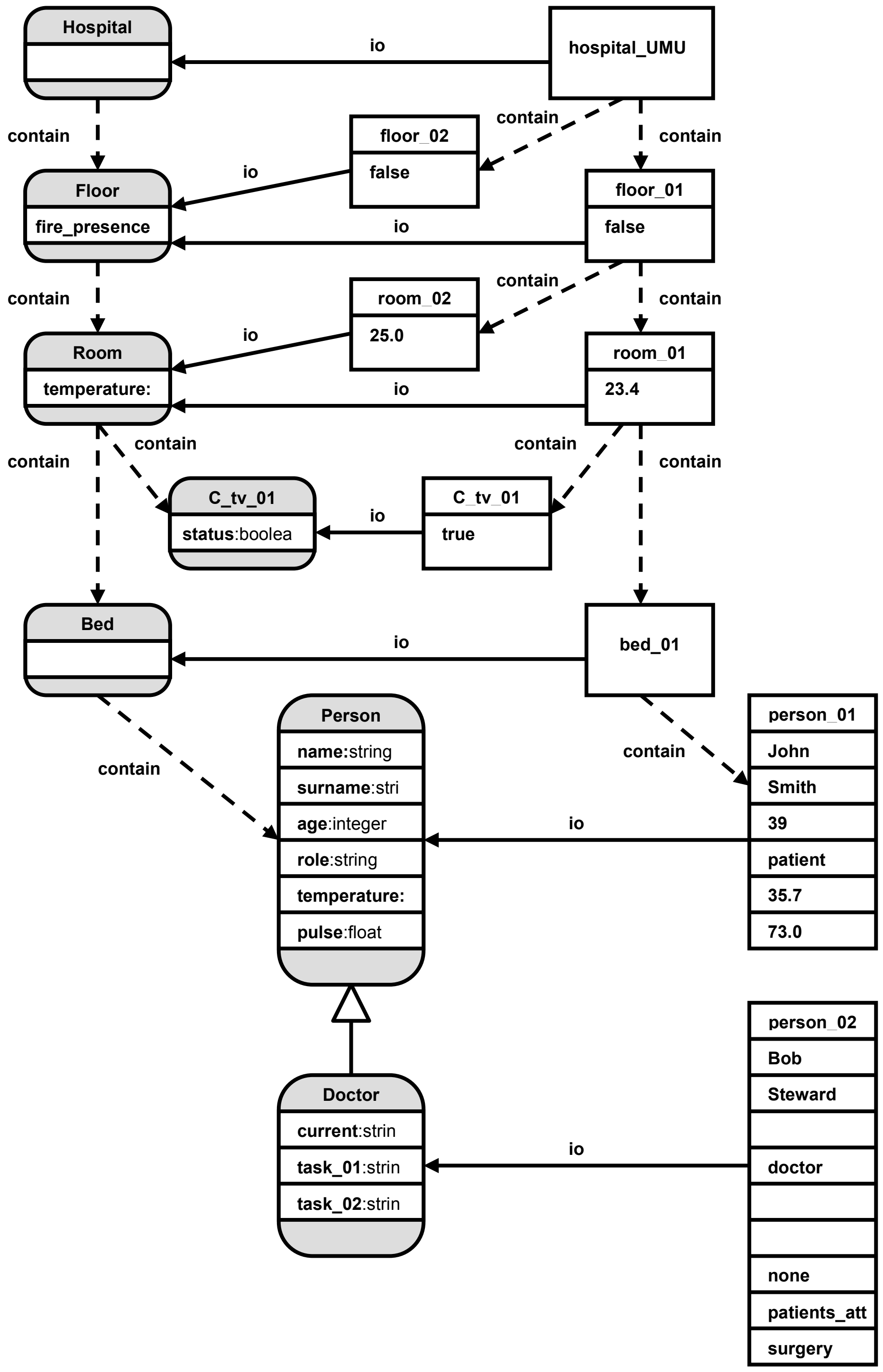


Figure 5.5 An excerpt of the OCP generated ontology for the hospital example .

We have used our own graphical notation rather than XML code for the sake of clarity, where rounded squares represent classes and squares represent individuals. The individuals are created during run-time as instances of the OWL classes (referenced by the io relationship) by the main Java program. The individual *person02* has all the properties of the *Doctor* class and the inherited properties of the *Person* class. The values of the properties are shown in the rectangles.

The OCP information producers are Java classes which are executed as a thread. As indicated above, these classes can be partially generated from the information contained into an MLContext model. This code generation is organized as follows:

1) A producer class is generated for each information source in the model. The information needed for the Java code of the producer class is obtained from the context information sources of the model, which supply the values of some properties of the entities. Figure 5.6 shows the run( ) method for the *equip01* source producer.
2) Each producer class includes the methods needed to update the value of the entities' properties based on the methods imported from the adapter (Figure 5.6, lines 23–28), and creates a context service to obtain the context of the entity (not shown in the figure). Then, the producer class adds (as a context item) the value of the supplied information to the context of the entity (Figure 5.6, lines 32–34).
3) The name of the Java class is obtained from the name of the source in the MLContext model (in Figure 5.6 it is equip 01 class).
4) There is also a call to an Adapter object method to obtain the value of the information supplied by the physical adapter (Figure 5.6, line 25). The Adapter Java class must be implemented by the developer or the manufacturer of the physical adapter, because the Java code only includes a call to the adapter method based on the name of the source method of the MLContext model.
5) We also generate an additional Java file (not shown in the example) which creates the skeleton of the main program that initializes all context instances and producers.

```
public void run() {

            int i = 0;
            while (i < NUM_OPERACIONES) {
                i++;
                try {
                    Thread.sleep( (long)(WAIT_TIME) );
                } catch (InterruptedException ex) {
                i = NUM_OPERACIONES; continue;
                }

         //---- Add adapter interface code here ----
                float returnValue;
                //returnValue=AdapterObject.getTemp();
                float r = (float)Math.random()*100.f;
                returnValue=r;
         //---- End Adapter interface section ----

               // check that context service is active
               waitForServiceActive();
               // update context entity property
               cs.setContextItem( entityClass, entityId, "temperature",
                                  returnValue);
            }
            // finalize
            ((OCPService)cs).terminate();
            System.out.println(id + " closed.");
        }
}
```

Figure 5.6 Code example from the equip01 source producer.

5.5.2 Generating Java code for the JCAF middleware

The model-to-text transformations are organized in the following way:

1) A JCAF entity is generated from each one of the categories of an MLContext model. They are Java classes which extend the *GenericEntity* (or other entity if they come from a subcategory). For example (see Figure 5.7, line 07) the entity *Person* extends *GenericEntity* and the entity *Doctor* is created as a subclass of *Person*.

2) Each of the entities has several attributes (e.g. Figure 5.7, lines 08–13) which are generated from the category properties of the context model, taking into account the inheritance relationships. For example, the name and surname attributes are not generated for the *Doctor* entity because it inherits those attributes from the *Person* entity.

```
07 public class Person extends GenericEntity {
08  private String name;
09  private String surname;
10  private int age;
11  private String role;
12  private float temperature;
13  private float pulse;

...

22  public Person(String id) {
23     super(id);
24  }
25
26  @Override
27  public String getEntityInfo() {
28     return "Person entity";
29  }
30
31  public String getName() {
32     return name;
33  }
34  public void setName(String x) {
35     name = x;
36  }

...

68  @Override
69  public void contextChanged(ContextEvent event) {
70     float new_temperature;
71     float new_pulse;
72
73     if (event.getRelationship() instanceof equip_01) {
74        if (event.getItem() instanceof Temperature) {
75          new_temperature = ((Temperature) event.getItem()).getTemperature();
76          this.setTemperature(new_temperature);
77        }
78     }
79     if (event.getRelationship() instanceof equip_01) {
80        if (event.getItem() instanceof Pulse) {
81           new_pulse = ((Pulse) event.getItem()).getPulse();
82           this.setPulse(new_pulse);
83        }
84     }
85  }
86 }
```

Figure 5.7 Excerpt of code generated for the Person JCAF entity.

3) We also generate public setter and getter methods for each of the attributes (e.g. Figure 5.7, lines 31–36), which are declared as private.

4) A JCAF entity also needs a special constructor with an id parameter (Figure 5.7, lines 22–24), which is instantiated when an object of the class is created, and is used internally by the middleware. This constructor is also automatically generated.

5) A JCAF relationship class is created to establish a relation between an entity property and the information source that provides it a value. Each of these classes implements the Relationship interface (not showed in the example code).

6) Entities are notified when changes to their context happens. If this occurs, the context service calls the *contextChanged()* method on the entity (Figure 5.7, lines 68–85). Therefore, when the value of some properties vary along the time, depending on a source of context, a *contextChanged()* method is automatically generated for these entities, but not for static properties. This method keeps an entity attribute up-to-date when events occur. The new value is a context item based on a relationship with a source of context.

5.6 Validation of the proposal

The benefits and disadvantages of using DSLs are well-known (Völter et al., 2013) and they must be contrasted to decide if the creation of a DSL is worthwhile. It is necessary to measure to what extent the productivity will be increased with the use of the DSL, as well as other software quality factors as maintainability, portability, integrability and reusability. Both, the productivity improvement and maintainability level reached, mainly depend on easiness of writing and reading, and on the understandability of the DSL code.

With regard to the disadvantages, the main factors to be assessed are the costs of development and maintenance, and the cost of training the developers. Next, we evaluate these factors in the case of MLContext and show how a proper balance has been achieved by using the hospital example.

Easy to read and understand

We have measured some classical metrics to calculate an estimation of the complexity of the language, in order to show some measurement that supports our claims on the simplicity, readability and understandability of

MLContext. The metrics applied are TERM, VAR and HAL (Power et al., 2004), which will allow us to obtain a brief description of the grammar complexity. We shall additionally use the LRS and LAT/LRS metrics proposed in (Crepinsek et al., 2010), which will allow us to provide a better characterization of the language. For the sake of concreteness, we shall not detail each metric, but simply discuss their meaning with regard to our DSL. These values were calculated by analyzing the DSL grammar structure and rules, using the gMetrics tool from (Crepinsek et al., 2010). Table 5.2 shows the results obtained, along with a brief explanation of each metric, and compares them with those of Java 1.5 and ANSI C.

Table 5.2 Results of applying grammar metrics.

Metric	Explanation	Java 1.5	C	MLContext
TERM	Number of grammar terminals	102	83	24
VAR	Number of grammar non-terminals	129	66	8
HAL	Designer effort to understand the grammar	140.38	42.34	3.54
LRS	Grammar complexity independent of its size	7741	2512	49
LAT/LRS	Facility to learn the language. Lower value is easier to learn	0.17	0.18	0.09

According to the values in the table, MLContext is clearly simpler than Java or C programming languages (see HAL and LRS). Moreover, the low value of LAT/LRS, denotes that MLContext is much easier to learn than the other languages.

Productivity improvement

The increase in productivity when using DSLs is often measured by calculating the effort saved in writing code for the final application. This effort is normally calculated as lines of code, and it is measured as the difference between the effort to write a DSL specification (i.e. the model) and the effort to manually write the artifacts which are automatically generated. Obviously, this effort should take into account the measurements on the complexity of the language showed in Table 5.2.

In the case of MLContext, the number of lines of code is 70 for the abstract syntax, 27 for the concrete syntax, 500 for the MOFScript API, 190 for the OCP transformation and 100 for the JCAF transformation.

For the small hospital example, we have automatically obtained 17 Java files for the OCP ontology, main program and producers, as well as the JCAF entities, relationships and monitors, with about 700 lines of code. The generated OWL-DL ontology is an XMI file with about 100 lines of code, which can be used by other tools like Protégé 2000. In contrast, the MLContext model source is a file with only 90 lines of code, which are written with a simple language. This is a significant saving in programming effort. Note that the benefit obtained for this small example almost compensate the cost of the MLContext development. For other OCP or JCAF new projects we only need to write the context model.

The model-to-text transformations should be written by experts in the target technology in order to generate highquality code. The MLContext API has been written to help those developers write their own transformations. OCP has been chosen because we have developed it, therefore we are able to write high-quality model transformations. On the other hand, a third-party middleware helps strengthen the evaluation of our approach. This is the motivation to select JCAF, a middleware in which we had no experience. In this case, the transformation was written using examples available in the JCAF web site as implementation patterns.

Portability and reusability

There are a number of middleware platforms for context-aware systems, and each of them has its own format for representing the context information (e.g. an ontology or Java classes). Our approach proposes the use of a generic Ecore metamodel to represent the context information in a platform-independent way. As proved with OCP and JCAF, the platform-specific context models can be generated from MLContext models. Therefore, MLContext models could be used in any middleware for which a transformation engine can be written (i.e. a mapping between the MLContext metamodel and the target platform can be defined). With regard to the model reuse, the MLContext models do not contain specific application code, so it is easy to reuse the same context model within other context-aware applications.

Maintainability

This is a typical benefit of using DSLs as they are simpler and more legible than general purpose languages. MLContext models are more maintainable than, for example, JCAF code or XML code.

Integration with other tools

MLContext is integrated into the Eclipse EMF framework, one of the most widely used MDE development platforms, so the MLContext models can be used by a wide variety of Eclipse modeling tools (parsers, editors, report generators,. . .) and the context model can be automatically serialized in XMI format.

Middleware migration

Most approaches proposed for representing context information do not support code generation for different middleware platforms (see Chapter 3). Through a separation of concerns, our approach allows the same context information to be used in different platforms, which could be useful to save effort in a middleware migration.

Cost of training developers

MLContext is easy to learn and favors the interaction between domain experts and developers. Actually, an MLContext specification could be written by a domain expert (i.e. a person with great knowledge on a given domain but, normally, with any or little expertise on programming).

In order to define a DSL that is easy to learn, we have taken into account some of the criteria and guidelines proposed in (Kolovos et al, 2006; Oliveira et al., 2009; Völter, 2009) such as:

- The language constructs must correspond to important domain concepts (conformity). In this sense, MLContext has constructs for Entities, Sources of context, Context information, Categories and properties.
- In MLContext, each construct in the language is used to represent exactly one distinct concept in the domain (orthogonality).
- The language should be as simple as possible in order to express the concepts of interest and to support its users and stakeholders in their preferred ways of working. To keep our DSL simple, the MLContext grammar has a low number of terminals.
- The semantics of the domain is implicit in the language notation. This allows their users to easily create mappings between the syntax of the language and the objects of the problem domain.
- Notations that only express concepts of the domain makes the language efficient for being read and learned by the domain experts.

6
Quality of context support

Improving quality

A good tool improves the way you work.
A great tool improves the way you think.

Jeff Duntemann

Handling context is a crucial activity in context-aware systems. When building these systems, the creation of models helps developers understand and reason the context information. The context information is principally obtained from sensors that gather data on the environment, but the context managers often erroneously assume that the information retrieved from the sensors is accurate and reliable. Limitations of sensors and the situation of a specific measurement can affect the quality of the context information and the adaptiveness of context-aware applications (Manzoor et al., 2011).

The QoC notion was proposed in (Buchholz et al., 2003) as an essential aspect to take into account in context-aware software. QoC has since received a considerable amount of attention from the research community and several works have contributed proposals with which to represent, measure and evaluate QoC.

As explained in the previous chapter, MLContext is a domain-specific language that is used to model context information, the core element of our MDE approach with which manage context information in context-aware applications. In this chapter, we present an extension of MLContext whose purpose is to model the aspects of QoC. A review of literature allowed us to identify the main issues to be addressed in the design of such an extension. The result is a DSL that provides (1) constructs that can be used to model context situations and quality parameters, (2) mechanisms that allow the composition and parameterization of situations when they are modeled, and (3) support for external function calls and the possibility of defining quality requirements while taking into account the multidimensional aspect of the context quality information.

Our QoC modeling approach has been validated for two different context-aware middlewares that support context quality: COSMOS (Chabridon et al., 2013) and SAMURAI (Preuveneers et al., 2014). We use the same case study to show how software artifacts for QoC management can be automatically generated from MLContext models. We particularly demonstrate how MLContext models are able to generate the Spring descriptors for SAMURAI and the models managed in the COSMOS approach which are expressed at a lower level of abstraction.

This chapter is organized as follows: in Section 6.1 we perform a domain analysis in order to identify the main quality parameters used in literature, the requirements for our QoC DSL, and the design decisions made to define it; in Section 6.2 the Application metamodel for the abstract syntax of the QoC DSL is explained; Section 6.3 presents the concrete syntax of the QoC extension. Section 6.4 proposes a method for context ad QoC modeling with MLContext; Section 6.5 presents a case study based on a flash sale scenario, which is used to illustrate the modeling process and the automatic code generation. We also show how MLContext can be used to model the flash sale scenario with QoC; Section 6.6 explains how an MDE approach has been applied in order to automatically generate specific code for the COSMOS and SAMURAI frameworks; Finally, in Section 6.7, we evaluate our approach in terms of productivity, maintainability and reusability.

6.1 QoC extension design

Before describing our QoC modeling approach in detail, we shall identify and discuss the domain concepts and their properties related to QoC, such as quality parameters and context situation. Next, we will show the conclusions of the domain analysis and the DSL requirements elicited from it. Finally, we will present the choices that we have made in the design of the QoC extension for the language.

6.1.1 Domain analysis

Quality of context refers to the quality of information that is used as context information and not to processes or devices that possibly provide the information (Buchholz et al., 2003). Several works on information quality (Wang et al., 1996; Eppler et al., 2002; Knight et al., 2005; Helfert et al., 2013) have assumed the idea that quality is simply a "fitness for use" according to the definition of quality proposed by Juran (Juran, 1974). This definition entails that information quality is application-dependent. A user might consider that the quality of a piece of information is appropriate for one application but it is not for another. The "fitness for use" may depend on various factors such as accuracy, timeliness, relevancy or precision. Therefore, information quality is commonly conceived as a multidimensional concern (Wang et al., 1996; Eppler et al., 2002; Naumann, 2002; Knight et al., 2005), in the sense that it needs of several parameters (dimensions) to be defined.

As explained in Section 2.3, a number of quality parameters and different classifications have been proposed in the literature (Kim et al., 2006; Preuveneers et al., 2007; Lachica et al., 2008; Lee et al., 2002; Wang et al., 1996; Wand et al., 1996). We have identified the more common parameters used to measure the context quality and have classified these parameters in the following three categories, as shown in Table 6.1.

- **Data acquisition**: these are quality parameters directly related to the sensors that captured the information and must be defined at the sensor level.

- **Data representation**: these quality parameters are used to specify acceptable representations of information that can be correctly interpreted by the application.
- **Data usage**: these parameters define the context of the data acquisition itself to decide if the data has significance.

Table 6.1 Classification of quality parameters.

Data acquisition	Data representation	Data usage
resolution	units	believability (trustworthiness)
precision	format	completeness
accuracy	understandability	relevance
range	aliases	comparability
fressness		availability
location		
coverage		

A review of the literature evidences some confusion in the meaning of three parameters. The term *resolution* is often confused with *precision*, as well as *precision* is confused with *accuracy*. *Resolution* is the fineness to which an instrument can be read or the smallest change in the underlying physical quantity that produces a response in the measurement. *Precision* is the degree to which repeated measurements under unchanged conditions show the same result. The *accuracy* of the sensor is the maximum difference that will exist between the actual value and the indicated value at the output of the sensor.

We have taken the *resolution*, *precision* and *accuracy* definitions from the field of engineering and industry for instrumentation measuring (Carr et al., 2010). In addition to these definitions, we will also define the freshness and trustworthiness parameters because they will be used in the case study in Section 6.5.1. Freshness refers to how recent the provided information is at the time of delivery (Kahn et al., 2002). We must be aware that some information remains valid over time, while other information may become discredited or obsolete. Trustworthiness is the extent to which information is regarded as true and credible (Wang et al., 1996) and a context-aware application should disregard poorly reliable sources of context. A complete definition for the quality parameters of Table 6.1 can be consulted in Appendix A.

The quality parameters associated with data usage, normally, depends on the context of the information (e.g. to know if some data are relevant we need to know the context situation).

The quality parameters are not independent of each other and typically only a subset of them is relevant in a specific situation. Which quality parameters are relevant and which levels of quality are required is determined by the specific application (Naumann, 2002; Wang et al., 1996). On the other hand, Table 6.2 summarizes dependences between quality parameters.

Table 6.2 Quality parameters dependencies.

Quality parameter	Dependences
Believability (trustworthiness)	precision, accuracy, freshness, range, location, coverage, comparability
Comparability	units, format, understandability, aliases, coverage, freshness, accuracy, range, precision
Completeness	all required parameters (determined by the application)
Relevance	determined by the application or the user (e.g. freshness, coverage, location, accuracy, etc.)
Understandability	units, format, aliases

The column "Quality parameter" shows the quality parameters which have dependences with those of the second column, in the sense that for computing the value of the first may be necessary to know some of the values of the others. For example, we can only compare two values if they are expressed with the same units and format, and we can only believe that the actual current position of a moving target is the same reported by a sensor if the reported value is not too old (freshness).

Next, we define some basic concepts on context quality which are needed to understand our proposal.

A *provider* is typically a sensor or a software entity which provides the value for a context property. A *context situation* is a set of constraints on context information so that when these constraints are satisfied in the sensed environment, the situation is said to occur (Clear et al., 2007). This set of constraints constitutes a logic predicate. For example, the health status of a patient could be based on his/her context information about temperature, blood pressure and pulse so that the context situation "the patient is ill" could be formed of the "*temperature>37ºC*" constraint among others. Throughout this article, we will refer to *context facts* as such constraints on context information (e.g. *temperature > 37*, *pulse < 72*). The global quality of a context situation can therefore be computed by using the quality of the pieces of context information (i.e. *quality parameters*) on which it is based. This should be carried out by the context-aware middleware in order to achieve full QoC support.

In context modeling, there also exist some complex relationships between entities that can be modeled as *context situations*. For example, the meaning of the relationship "to be close to" can vary (e.g. from millimeters to meters) depending on the application domain. That is the case for a context situation which defines when a person is near from a shop by using the coordinates supplied by sensors.

Finally, we can specify constraints on the values for the quality parameters (e.g. *precision < 0.5*) and we refer to them as *quality requirements*. Therefore, a *context situation* occurs when the *context facts* and the *quality requirements* are satisfied.

From this analysis we can conclude the following requisites for the QoC extension:

- As QoC is application-dependent, it must be separated from the context information to favor reuse of models.
- The language constructs must correspond to significant domain concepts to achieve expressiveness.
- The language must offer constructs for expressing quality parameters for the context information.
- As quality parameters for data acquisition are related to providers of context information, the language must have constructs for defining such providers.
- The language must allow defining context situations.
- The language must allow defining quality requirements for the application.

6.1.2 Design choices

In this section we shall explain the main design choices that we have taken to satisfy the elicited requirements.

QoC is modeled into the application model

On the one hand, as noted in Section 6.1.1, the context quality information is application-dependent and the user might consider the quality of a piece of information to be appropriate for one application but not sufficient for another one. On the other hand, the quality of the raw data retrieved from the sensors depends on their quality parameters, and a sensor can be replaced for another one (e.g. with higher precision or accuracy). Therefore, all the information related to quality parameters, context situations and quality requirements will be modeled in the application model of MLContext. With regard to the context quality parameters we have taken into account those of Table 6.1.

QoC parameters expressiveness

The language constructs should correspond to significant domain concepts related to QoC. In this sense, MLContext has constructs for expressing quality parameters such as accuracy, precision, units, etc (see Figure 6.1). Each construct in the language is used to represent exactly one distinct concept in the domain, and the semantics of the domain is implicit in the language notation. This allows their users to easily create mappings between the syntax of the language and the objects of the problem domain.

Modeling of context situations with QoC

A context situation should be expressed by using a simple expression language which includes QoC operators to specify quality requirements. In order to express complex context situations the language should provide support to invoke external functions with the purpose of achieving a proper balance between expressiveness and simplicity. We have added such constructs to MLContext. To improve clarity and reuse, context situations can be parameterized in MLContext, and we can use composition when defining them.

No constructs for QoC parameters with dependencies

The language should not have constructs for quality parameters with dependencies on others (e.g. comparability) because they can not be expressed at design time as quality parameters. They are computed at runtime and can only be used as requirements when defining context situations. The form in which they are computed depends on the concrete middleware.

Dealing with the multidimensional nature of the QoC

As explained in Section 6.1.1, the quality of the context information commonly depends on various quality parameters and it is commonly conceived as a multidimensional concern. A QoC modeling language should therefore support the multidimensional aspects of context quality. In this sense, we have added to MLContext a construct to specify quality levels as an aggregation of quality requirements. These quality levels are identified by a unique label which can be used to specify those quality requirements when defining context situations. For example, we can define a *low_quality* "when the *freshness* of the context information is less than 0.8, its *accuracy* less than 10 and its *trustworthiness* less than 0.9".

Take decisions about the trustworthiness of a context source

Many times, a context-aware application must decide whether or not to trust a context source. This can be achieved in two ways: (1) the absence of contradictions, (2) the presence of consistency with other context information about the same context. MLContext allows us to associate several providers that provide the same information, to one context source, so that a context-aware application can use this information from the application model to compare the supplied information and detect contradictions. The way in which this information is used depends on the target middleware which should implement an algorithm for computing it.

Modeling of QoC must be platform independent

In order to follow an MDE approach, the QoC model must be platform-independent (PIM). Then we can obtain software artifacts or platform-specific models by means of model-to-text or model-to-model transformations. It is worth noting that the modeling of QoC, as an extension of MLContext, is platform-independent but it remains specific to a concrete application.

6.2 Abstract syntax

As explained in Section 4.1.2, the MLContext metamodel is really organized in two metamodels: Context and Application. This allows the separation of concerns in context modeling. Therefore, developers will be able to separate the context model from the models that represent application-dependent aspects such as quality of context and adaptation. An application model should import the context model for which is defined.

The Context metamodel was described in Section 5.2 and we will present here the Application metamodel which includes the QoC concepts. Figure 6.1 and Figure 6.2 show an excerpt of the QoC capabilities of the Application metamodel, which has been defined bearing in mind the design decisions introduced in Section 6.1.2. It is worth noting that the Application metamodel has references to elements of the context model (e.g. *Method metaclass*).

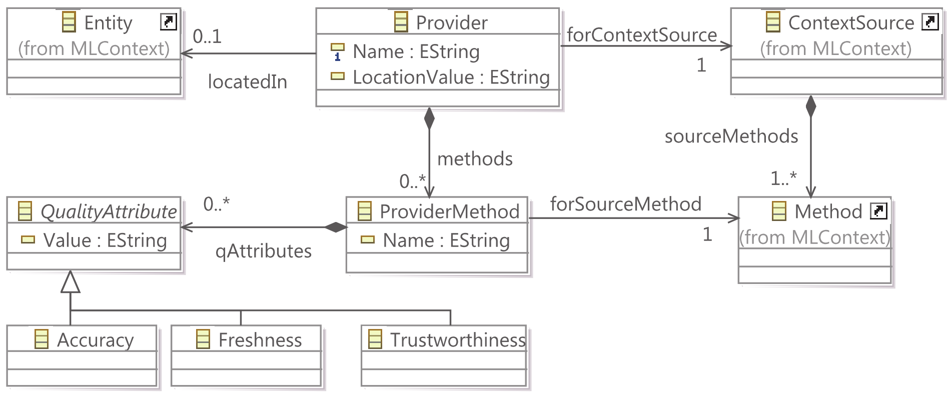


Figure 6.1 Excerpt of the Application metamodel. Providers definition.

Figure 6.1 shows the metaclasses involved in the provider definition. A *Provider* has a name and a value that indicates its location in the form of a textual description (*LocationValue* attribute). A provider can also be located into an *Entity* (metaclass from Context metamodel). Each provider can supply data for a *context source* (from Context metamodel) by means of a *ProviderMethod* which has a name that is the same as the *Method* referenced (from MLContext metamodel). A *ProviderMethod* can specify *Quality attributes* (i.e., quality parameters). Normally, the developer can obtain the values for these parameters from the technical specifications of the sensors (e.g. *Accuracy*).

Figure 6.2, shows an excerpt of the metamodel which includes some of the metaclasses involved in the definition of a context situation. A *context situation* (*Situation metaclass*), has a name and can be parameterized. Each parameter (*SParameter metaclass*) of the context situation is defined by its name and type.

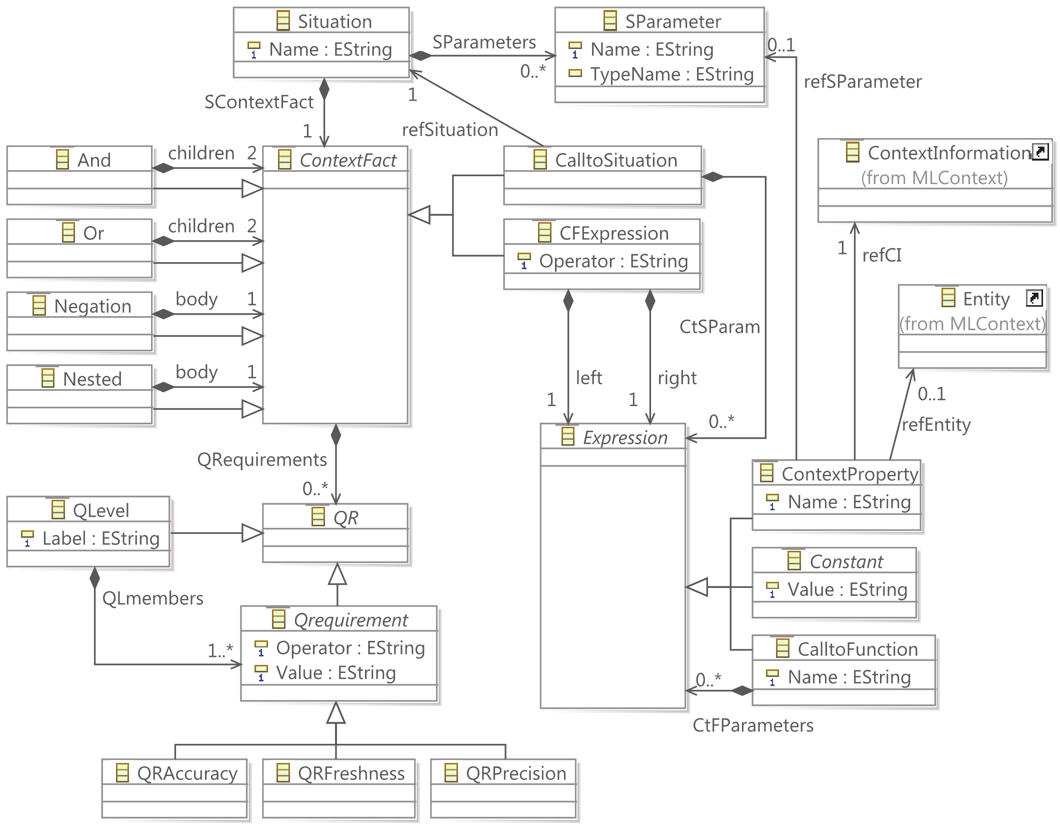


Figure 6.2 Metaclasses for a context situation definition.

A context situation, as explained in Section 6.1.1, consists of *context facts* (*ContextFact metaclass*) which are connected by a relational operator (e.g. *AND*, *OR metaclasses*). A context fact can be either a call to another context situation (*CalltoSituation metaclass*) or a context fact expression (*CFExpression metaclass*). Since a situation can call other ones, then a composition of context situations is possible. On the other hand, a context fact expression compares two values (*left* and *right*) to return a boolean result.

These values are represented by the *Expression metaclass* which can be a constant value (*Constant metaclass*), a call to an external function (*CalltoFunction metaclass*) or a context property (*ContextProperty metaclass*) from an entity of the context model. The context property can also be obtained from a parameter of the context situation (*refSParameter relationship*).

Quality requirements for each context fact (*Qrequirement metaclass*) may also be specified, either directly or by using a label which represents a *quality level* (*QLevel metaclass*) that can be defined in the model. A quality level is defined as a set of quality requirements (*QLmembers relationship*).

6.3 Concrete syntax

A textual concrete syntax for the QoC metamodel has been created, which can be used to create MLContext Application textual specifications (i.e. Application models). Figure 6.3 shows an excerpt of the EMFText specification that defines the syntax.

The first line of the header specifies an "mlctq" extension which will be used to identify the files containing an MLContext Application model. This header also specifies the parent metaclass that represents the model (*QualityModel metaclass*). An application model is defined by the "qualityModel" keyword, followed by the name of the model.

As mentioned above an Application model must import a context model. This is achieved by means of an "import" sentence which specifies the path to the file containing it.

```
SYNTAXDEF mlctq
FOR <http://www.modelum.es/MLContextQ>
START QualityModel

TOKENS {
DEFINE COMMENT $'//'(~('\n'|'\r'|'\uffff'))*$;
DEFINE INTEGER $('-')?('1'..'9')('0'..'9')*|'0'$;
DEFINE FLOAT $('-')?(('1'..'9') ('0'..'9')* | '0') '.' ('0'..'9')+ $;
DEFINE BOOLEAN $('true' | 'false')$;
DEFINE PREDEFINED_TYPE $('Int'|'Float'|'String'|'Boolean')$;
DEFINE RELOP $('<' | '>' | '<=' | '>=' | '==' | '!=')$;
}

RULES {
QualityModel ::= "qualityModel"  Name[]  "{" elements* "}" ;

ModelImport ::= "import" rootElement['"','"'] ;

Provider ::= "provider" forContextSource[] "{" ("name" ":" Name['"','"'])?
"location" ":" (LocationValue['"','"'] | locatedIn[]) methods+ "}" ;

ProviderMethod ::= "method" ":" forSourceMethod[] ("{" qAttributes+ "}")?;

//Quality parameters
Precision ::= "precision" ":" Value['"','"'] ;
Accuracy ::= "accuracy" ":" Value['"','"'] ;
Range ::= "range" ":" Value['"','"'] ;

...

//Situations
Situation ::= "situation" Name[] ("(" SParameters ("," SParameters)* ")")?
"{" SContextFact "}" ;

SParameter ::= Name[] ":" (TypeName[PREDEFINED_TYPE] | SPType[] );

CalltoSituation ::= refSituation[] "(" (CtSParameters (","
CtSParameters)*)? ")" ("[" QRequirements ("," QRequirements)* "]")?;

...

//Quality requirements
QRPrecision ::= "precision" Operator[RELOP] Value['"','"'];
QRAccuracy ::= "accuracy" Operator[RELOP] Value['"','"'];
QRFreshness ::= "freshness" Operator[RELOP] Value['"','"'];
QRTrustworthiness ::= "trustworthiness" Operator[RELOP] Value['"','"'];

...

}
```

Figure 6.3 Excerpt of the QoC concrete syntax.

Note that, as in the Context model, language keywords correspond to basic domain concepts. For example, providers are defined by using the "provider" keyword and quality parameters by using keywords such as "precision", "accuracy" and "range".

There are two different subsets of rules for quality parameters and quality requirements though they have a similar syntax. A value is assigned to quality parameters when they are defined and quality requirements are specified by means of a relational operator.

Figure 6.3 also shows the rules for defining a context situation (keyword "situation"), which can be parameterized and is formed of context facts. It can include a call to another situation (CalltoSituation rule).

6.4 A method for context modeling

Figure 6.4 shows an overview of our MDE approach which was outlined in Section 4.1. Firstly, the developer creates the MLContext model (which contains both the Context model and the Application model). Users could collaborate in the definition of these models since the modeling language reflects the domain concepts in an understandable way to final users. These models are then used to automatically generate software artifacts for different frameworks (SAMURAI and COSMOS in this case as shown in Section 6.6). Quality requirements can also be specified for the context. It is worth noting that MLContext models are kept separate from QoC models in order to achieve the separation of concerns discussed in Section 4.1.2 (i.e. a QoC model imports the MLContext model). This means that a new language with which to express QoC specifications has been created rather than modifying the original MLContext language.

This approach can be extended to other frameworks by adding a new transformation engine. The transformation engine creates a map from the MLContext model to the target framework through a model transformation chain (implemented as a one-step M2T transformation in our examples) by using the MOFScript language and the MLContext MOFScript API intended to help developers in the process of writing their own transformations. The models are created by using the MLContext editor implemented as an Eclipse plug-in.

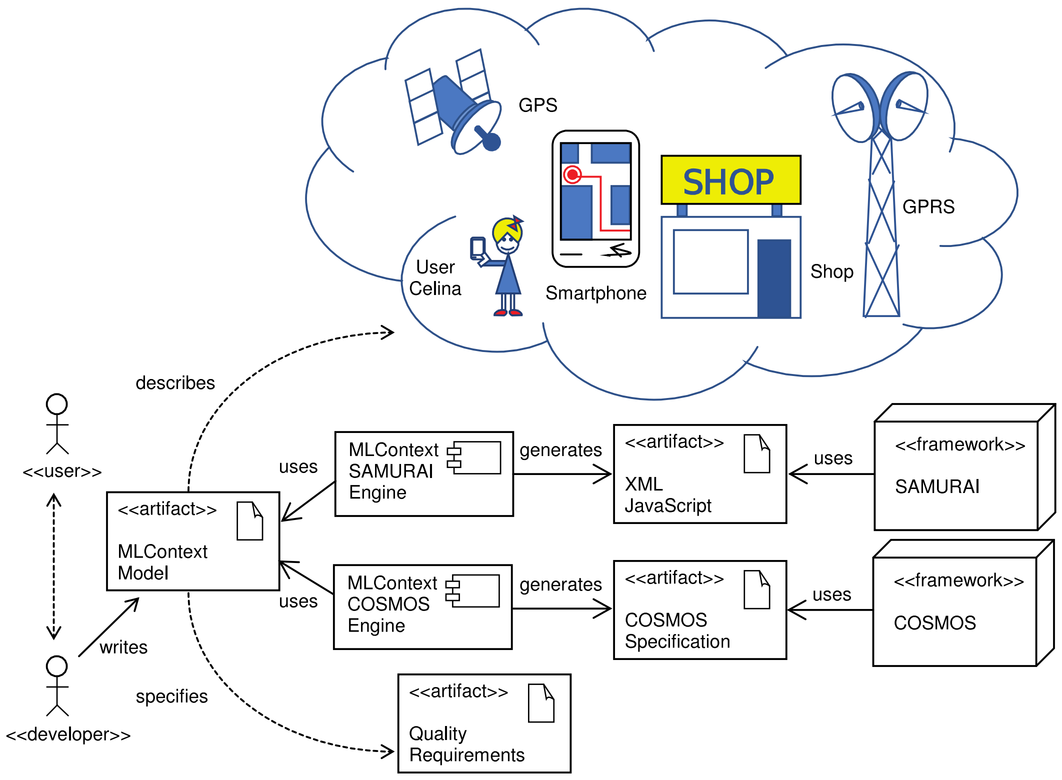


Figure 6.4 Overview of the MLContext MDE approach.

As mentioned above, the context information is represented by means of two models in MLContext: the context model and the application model. The context model represents the context information and includes the definition of the entities, their properties and the sources of context available to the context-aware application. On the other hand, the application model represents information related to providers and QoC, as well as those aspects related to context situations. The definition of an application model requires referring to elements included in the context model.

Figure 6.5 shows the steps that the developer must follow when modeling the context for a context-aware application. These steps can actually be performed in any order, but this is our recommendation based on the MLContext characteristics. Entities should be modeled prior to categories because if a category has only one entity the developer does not need to specify it. MLContext can automatically create the category for one entity, and the properties for the categories are also automatically inferred from their entities, as explained in Section 5.1.3. Sources of context are modeled after we have specified the entities because a source of context supplies a value for a property of an entity.

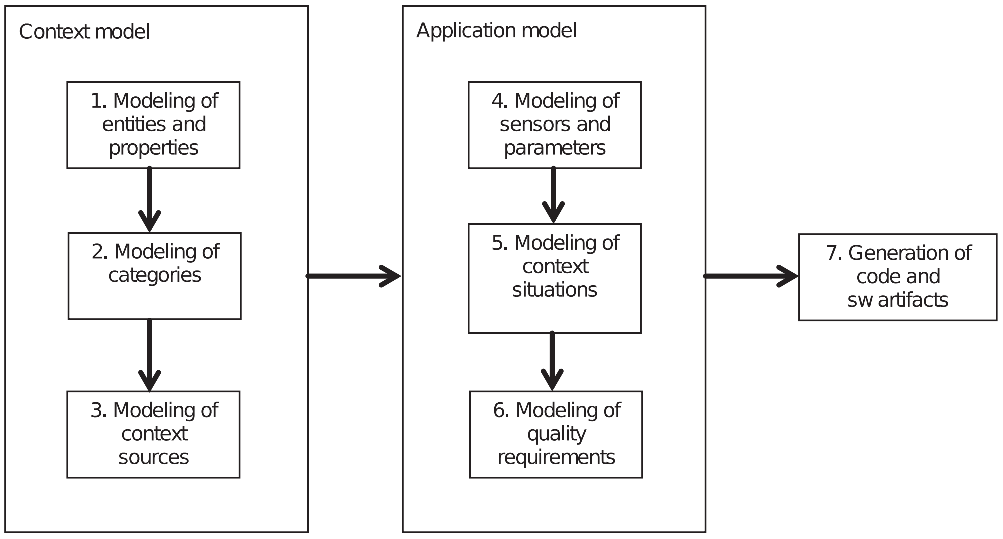

Context model
1. Modeling of entities and properties
2. Modeling of categories
3. Modeling of context sources
Application model
4. Modeling of sensors and parameters
5. Modeling of context situations
6. Modeling of quality requirements
7. Generation of code and sw artifacts

Figure 6.5 Steps in the modeling of context with MLContext.

The Context model should be created before the Application model since the latter includes references to the former. We propose therefore the following three steps:

1) **Specify entities**. The name and the properties of each entity must be specified. Simple relationships are also modeled in this step.
2) **Specify categories**. We must define the categories that entities belong to. We can also define a hierarchy of categories.
3) **Specify context sources**. The context sources which supply the values for those properties which are not constants must be specified. More than one context source for the same property can be defined. It is possible for a context source to supply values for different properties.

Once the developer has created the context model, he can specify the application model in the following three steps:

4) **Specify technical details and quality parameters for the sensors (providers)**. Each sensor should be attached to a source of context.

5) **Specify complex relationships and context situations**. The context information needed to determine if there exists a relationship between entities or if a context situation is occurring can be grouped in MLContext.

6) **Specify quality requirements**. Quality requirements and quality levels can be defined in MLContext. These requirements can be used later to determine if a context situation is occurring.

Steps 5 and 6 can be performed in parallel, and sensors modeled in step 4 can also be modeled in the end. Finally, in step 7, we can automatically generate code from the context model and the application model. Some examples of transformations have been already shown in Section 5.5 and two others are explained in Section 6.6.

6.5 Flash sale scenario modeling with MLContext

COSMOS is a context-aware framework with QoC support which can generate code from its models by applying MDE techniques. To show the capabilities of MLContext to model QoC we will demonstrate how COSMOS models can be automatically obtained from MLContext models which have a higher level of abstraction and are more readable. Therefore, we have chosen as case study the following scenario that is used in (Chabridon et al., 2013) to illustrate how QoC is managed with COSMOS. This scenario will be also used to validate our approach for a second middleware that supports QoC, in particular the SAMURAI framework (Preuveneers et al., 2014). Both middlewares were explained in detail in Section 3.1.3 and Section 3.2.6.

6.5.1 Case study: Flash sale scenario

A user (Celina) has a smarthphone so it is possible to use its location services based on not only GPS navigation but also 3G and Wifi communication. She has downloaded a flash sale application in her smartphone to receive a notification whenever a flash sale offer is near her that matches her preferences. Depending on the quality level of the reported location for the user, she receives more detailed indications.

In this scenario the location of the user can be provided by several sensors (GPS, GPRS and Wifi), which has different QoC properties. Three QoC parameters are considered for the location property (accuracy, freshness and trustworthiness) and three QoC levels are also defined (high, medium and low as depicted in Figure 6.6) depending on the values that might be taken by the quality parameters (see (Abid et al., 2009) for computation details).

quality high:

freshness >= 0.8;
accuracy < 10;
trustworthiness >= 0.9;

quality low:

freshness < 0.8;
accuracy < 10;
trustworthiness < 0.9;

quality medium:

freshness >= 0.8;
accuracy < 10;
trustworthiness < 0.9;

Figure 6.6 QoC levels defined for the flash sale application.

We shall follow the process described in Section 6.4 in order to model the flash sale scenario with MLContext.

6.5.2 Context model

The context model contains the entities, the categories to which they belong and the sources of context.

Specify entities

The context information that we need to know from the *Celina* user are her location, her preferences and the fact that she carries her smartphone. Therefore, this user is represented by an entity that has three context properties: location, preferences and carrying.

```
entity Celina context {
      environment  {"location" source historic}
      personal     {"preferences" source
                    "carrying" : smartphone}}
```

The "location" property is of environment type. As indicated in Section 5.4 the keyword *source* denotes that the value for the property "location" is obtained from one or more sources of context and the keyword *historic* tells the application that we want to keep a trace of the values over the time. Because the user is moving, this information can be used later to compute the speed of the user. The "preferences" and "carrying" properties are of *personal* type. The former is attached to a source but not the latter. The "carrying" property specifies a relationship with the *smartphone* entity, which will be modeled later. Next, the entities *smartphone* and *shop1* are specified.

```
entity shop1 context {
    environment{"location" : "37.607602, -0.975380" units : "WGS84"}
    computational{"saleOffer" source}}

entity smartphone context {}
```

Two context data are needed from the *shop1*: its location and the sale offer, whose types are *environment* and *computational*, respectively. The value for the property "location" is indicated and expressed in WGS84 units. However, note that we do not need context properties for the *smartphone* entity in this scenario, so we do not model them. The application knows that the location of the user *Celina* is the same of the *smartphone* only if the user carries it.

Specify sources of context

Now, we can model the context sources for the properties that have been attached to a source, that is, location and preferences for the *Celina* entity and saleOffer for the *shop1* entity. Next, the possible context sources for the location of *Celina* are specified.

```
contextSource GPRS {
      interfaceID : "gprs01"
      methodName : "getLocation" {
            supply : Celina.location
            returnValue : "WGS84"}
      }
contextSource GPS {
      interfaceID : "gps01"
      methodName : "getLocation" {
            supply : Celina.location
            returnValue : "WGS84"}
      }
contextSource WLAN {
      interfaceID : "wlan01"
      methodName : "getLocation" {
            supply : Celina.location
            returnValue : "WGS84"}
      }

contextSource Smartphone_Agent {
      interfaceID : "sm01"
      methodName : "getPreferences" {
            supply : Celina.preferences
            returnValue : "String"}
      }
```

Since Celina's smartphone has GPS navigation, 3G and Wifi communication, three sources of context have been specified to determine Celina's position: *GPRS*, *GPS* and *WLAN*. As indicated in Section 5.4, the keyword *interfaceID* specifies the name of the interface for the context source, and the keyword *methodName* specifies the name of the method to be invoked to obtain the value for the property. In each method we specify the property for which the value is supplied (*Celina.location* in this case) and the type of the value returned (WGS84 coordinates in this case). With regard to the Celina's preferences, we have defined the *Smartphone_Agent* source with a method "getPreferences". We will suppose that her preferences are stored in her smartphone and can be retrieved by a software agent.

A context source *Shop_scheduler* would also be defined in the same way to express how the sale offers are obtained from the *shop1* entity. The values supplied by *Smartphone_Agent* and *Shop_Scheduler* should be compared to check if the shops offer match to the Celina's preferences.

Specify categories

Once the entities are defined, the categories to which they belong could be indicated. In this case, we have defined the *User* category for *Celina* entity and the *Shop* category for the *shop1* entity, as shown below.

```
categories {
      User:Celina
      Shop:shop1}
```

We have not specified a category for the *smartphone* entity, so MLContext will generate automatically one category for it which would be called *C_smartphone*.

6.5.3 Application model

As indicated in Section 6.1.2, the Application model is specific for a concrete context-aware application and includes the following information: technical data about sensors, quality parameters, context situations, quality levels and quality requirements.

Specify technical details and quality parameters for sensors

The context sources are attached to physical adapters or providers. The following providers have been specified for the three context sources defined for the Celina's location (i.e. GPRS, GPS and WLAN).

```
provider gprs01 {
      name : "3G_cell_id"
      location : smartphone
      method : getLocation {
            units : "meters"
            accuracy : "500"}}

provider gps01 {
      name : "GPS_manager"
      location : smartphone
      method : getLocation {units : "GPRS"}}

provider wlan01 {
      name : "WIFI_manager"
      location : smartphone
      method : getLocation {units : "GPRS"}}

provider wlan01 {
      name : "Bluetooth_manager"
      location : smartphone
      method : getLocation {units : "GPRS"}}
```

A *provider* is defined by specifying the interface of the context source to which it is attached (e.g. gps01, wlan01). Note that several providers can be defined for a given interface. For instance, we have defined two providers for the interface "wlan01" of the *WLAN* context source in our Context model: "WIFI_manager" and "Bluetooth_manager". The Celina user position could therefore be obtained from two different providers through the context source attached to this entity.

The definition of a provider consists of three elements: *name*, *location* and *method*. The *name* is used to give a unique id to the provider. The *location* indicates where the provider is located and its value can be a textual description of the location or an entity from the Context model. In our scenario, the providers are located in Celina's smartphone. This information allows the application to know that Celina's location can be obtained from these providers only if she is carrying her smartphone, but not in other case. We can specify quality parameters for each of the method defined in the context source interface (identified by the keyword *method*). In our scenario, there is only one method defined in the interface (getLocation) and we have specified the units of the supplied values and an accuracy of 500 meters for the GPRS provider (this data is obtained from the manufacturer's specification). For simplicity, this example assumes that the controllers for the providers convert their units to GPRS units, but they can use any units. The context-aware middleware should be able to convert from one type of units to other in order to compare their values.

We also define the *User_profile_manager* and the *Shop_manager* providers in the same way (not shown here for the sake of space). These two providers do not have any quality parameters as they are not necessary in this example.

Specify quality levels

Like in COSMOS, quality levels can be defined in MLContext. Each level is identified by a unique label and specifies restrictions on the values that can take one or more quality parameters. When context situations are defined, these labels can be used to avoid writing too long expressions to specify the quality requirements. We have modeled the following three quality levels for our scenario as described in Section 6.5.1.

```
quality {
    QUALITY_HIGH : [freshness >= "0.8", accuracy <"10", trustworthiness >= "0.9"]
    QUALITY_MEDIUM : [freshness >= "0.8", accuracy <"10", trustworthiness < "0.9"]
    QUALITY_LOW : [freshness < "0.8", accuracy <"10", trustworthiness < "0.9"]
}
```

Specify context situations

The next step is to define the context situations for the application. As explained in Section 6.1.1, a context situation is a boolean expression that is based on context facts. We have defined the "flash_sale_alert" context situation that checks if the user is near the shop and this shop has a flash sale offer matching the user preferences. This situation has therefore two input parameters: the user and the shop.

```
situation flash_sale_alert (u:User, s:Shop) {
    user_near_shop(#u,#s)[QUALITY_HIGH]
    AND
    _flash_sale_offer_matching_user_info(#u.preferences,#s.saleOffer)==true
}

situation user_near_shop(u:User, s:Shop) {
    _AdjustedLocation(#u.location,_Speed(#u.location))==#s.location
}
```

As indicated in Section 6.1.2, MLContext provides a composition mechanism in order to favor the reuse and legibility of the context situations. We have taken advantage of this composition mechanism to define the "flash_sale_alert" situation by using the "user_near_shop" situation that checks if the adjusted location of the user is near the shop location. Note that we have expressed a quality requirement for the "user_near_shop" situation by using the "QUALITY_HIGH" label that we have previously defined.

These context situations also illustrate the use of external functions to implement complex relationships.

The "_flash_sale_offer_matching_user_info" function returns true if there is a sale offer matching the user preferences. Both, user preferences and sale offer, are parameters of this function and properties of the User and Shop categories in the context model. As the user is moving, the "_AdjustedLocation" function computes the value of the real position of the user based in the reported location and the speed of the user. The "_Speed" function computes the speed vector of the user using the position history of the user as stored on the smartphone (note that we specified the historic modifier for the user location property in the Context model). These three functions would be part of a library which would be used when generating code from the model.

In Section 6.1.1, we mentioned that some quality parameters have a dependency on others. MLContext does not provide specific constructs to express these kinds of parameters. However, they can be computed by the application using the context information from the model. For example, in the case of the completeness parameter, from the definition of the "user_near_shop" context situation defined above, the application knows that it can only be computed if the user location and the shop location are known, because these are properties that appear in the body of the definition. The application also knows the context sources that can supply those values. This way it does not try to compute the situation if some of the required context information is missing. With respect to the comparability parameter, the application can check if the user location and the shop location are expressed in the same units as, otherwise, they could not be compared (unless it can transform these units into the others). As the user location can be obtained from several sources of context, the application can also select those sources that supply the value in the required units. Finally, in this example, if the user position reported by one of the context sources differs significantly from the position reported by all of the other context sources, the trustworthiness for that context source should be decreased depending on the capabilities of the framework.

6.6 Generating code from the models

We have chosen the COSMOS and SAMURAI middlewares in order to assess the capability of MLContext to generate code for context quality management. We have chosen SAMURAI because we have a deep understanding of this middleware. Therefore we can check the result of the model-to-text transformation for optimum code generation. On the other hand, we have chosen COSMOS because we want to generate code for a third-party middleware with context quality support and it is the most complete and advanced approach we know. While SAMURAI is an event-based stream mining framework, COSMOS is a component-based framework where context information is modeled by components called context nodes.

In Sections 3.1.3 and 3.2.6, the main characteristics of the SAMURAI and COSMOS middleware platforms were explained. In this section we will show how the flash sale scenario can be modeled in these frameworks and how the artifacts for managing context can be automatically generated from the MLContext models.

6.6.1 Generating code for the COSMOS framework

As explained in Section 3.2.6, COSMOS is based on the notion of context node, i.e. context information modeled by a component. Context nodes are organized into hierarchies, and each node can receive information from their children nodes and send information to their parent nodes. Context nodes contain one or more operators to process the context information. Figure 6.7 from (Chabridon et al., 2013) shows an excerpt of the context nodes hierarchy for the flash sale scenario.

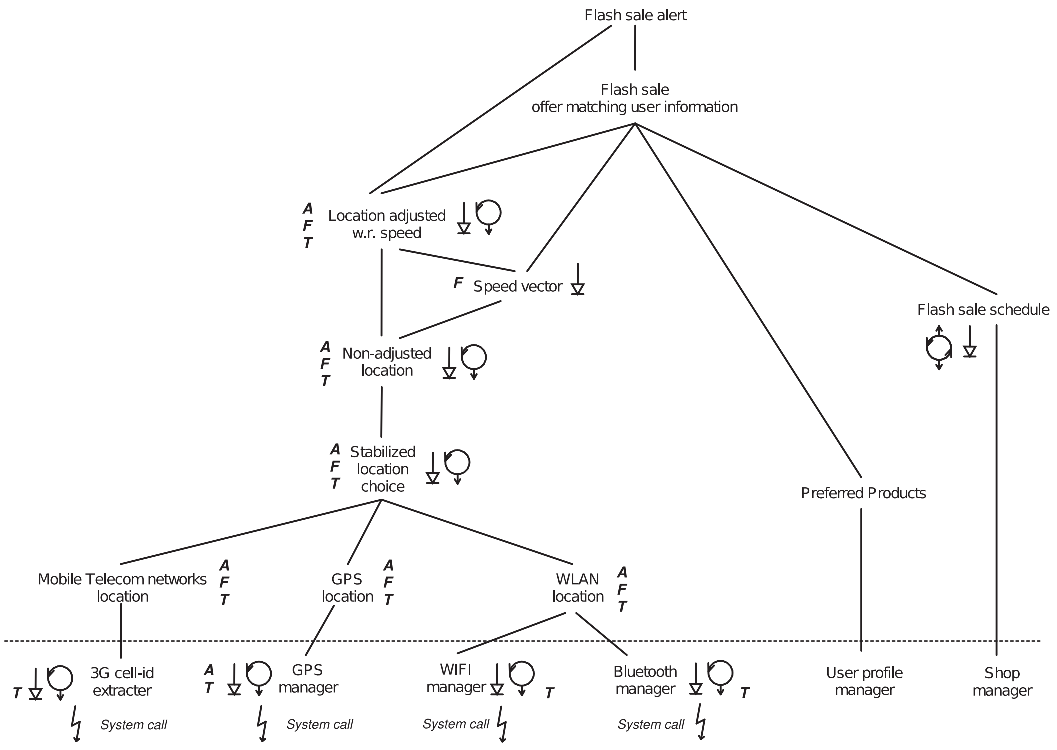


Figure 6.7 Context node hierarchy for the flash sale application.

There are three nodes which can obtain the user location: "Mobile Telecom networks location", "GPS location" and "WLAN location". These nodes take their context information from the leaves of the hierarchy, which are context collector nodes. Note that the "WLAN location" node has two collectors so it can obtain the location from the WIFI manager or the Bluetooth manager.

On the one hand, the "Stabilized location choice" node is the node responsible for choosing the best location value, based on the accuracy, freshness and trustworthiness values reported by its children (these quality parameters can be obtained directly from the collectors or can be computed by the other nodes). Then, the chosen location value is sent to the "Non-adjusted location" node. The reported value for the user location could be outdated, because the user is moving, so there is a "location adjusted w.r. speed" node which can compute the real position based on the location value and the speed vector from the user. The speed vector is calculated by the "Speed vector" node by using two values from the "Non-adjusted location" node and their timestamps.

On the other hand, the "Flash sale schedule" node can obtain both, the shop location and the sale offer product, from the "Shop manager" collector. The "Preferred products" node can obtain the user preferences from the "User profile manager" collector. Then, the "Flash sale offer matching user information" node compares the user preferences with the sale offer. If there is a match, the "Flash sale alert" node will report an alert if the user position is near the shop location.

Figure 6.8 from (Chabridon et al., 2013) shows some lines of the COSMOS DSL specification for the user location management subtree from the flash sale application example.

```
package every2mallqoc;
chunk position = {
   classname : every2mallqoc.PositionChunk,
   typeparam : every2mallqoc.PositionInfo
}
message location = {
   position, accuracy, freshness, trustworthiness
}
operator computeGprsLocation = {
   output : location,
   classname : every2mallqoc.ComputeGprsPosition,
   input : gprs
}
node gprsLocation = {
   operator : computeGprsLocation
   children : gprsExtracter
}
node stabilizedLocation = {
   operator : computeStabilization,
   children : gprsLocation gpsLocation wlanLocation
}
operator computeNonAdjustedLocation = {
   output : location,
   classname : every2mallqoc.ComputeNonAdjustedLocation,
   input : location
}
```

```
node nonAdjustedLocation = {
   operator : computeNonAdjustedLocation
   children : stabilizedLocation
}
operator computeAdjustedLocation = {
   output : location,
   classname : every2mallqoc.ComputeAdjustedLocation,
   input : location speed
}
node adjustedLocation = {
   operator : computeAdjustedLocation
   children : nonAdjustedLocation speedVector
}
configuration : gprsExtracter
   [observethrough=false] [periodobserver=10000]
configuration : stabilizedLocation
   [observethrough=false] [periodobserver=2000]
configuration : nonAdjustedLocation
   [observethrough=false] [periodobserver=3000]
configuration : adjustedLocation
   [observethrough=false] [periodobserver=5000]
```

Figure 6.8 Adjusted location management in COSMOS DSL.

The keyword "package" is used to specify the name of the Java package where the generated code will be included (*evry2mallqoc* in the example). A chunk definition (e.g. *position* in this example) specifies the Java class that implements the chunk (*PositionChunk* class) and its data type (*PositionInfo* class). If these classes do not exist, the COSMOS framework will generate a skeleton to be completed by the developer. The message definition specifies a message type for the exchange of information between nodes. For example, the "location" message contains the chunks: "position" (previously defined) and the "accuracy", "freshness" and "trustworthiness" chunks. The operator definition (e.g. *computeGprsLocation*) specifies an operator to be used by a node and contains the output message for the operator (e.g. location), the name for the Java class that implements the operator (e.g. *ComputeGprsPosition* class), and the input messages (e.g. gprs, that is the raw output for the gprs collector node in this example).

At the end of the specification there are some configuration parameters for the nodes not related with the context information, such as the period for notification in milliseconds. Finally, collector nodes can also be defined although such definitions are not shown in the example code.

We have applied an entirely different code generation approach due to the differences between SAMURAI and COSMOS. The context information is not defined in a separated model in COSMOS but instead is scattered through each of the context nodes. Since the COSMOS DSL is a language for describing the architecture for the context nodes hierarchy, our transformation engine must infer not only the COSMOS components but also how they are connected. The output of the MLContext transformation will be a COSMOS DSL specification like the one in Figure 6.8.

As we can see in Figure 6.8, each node of the hierarchy needs the context information from its children in order to compute the assigned operation, so we need to start our transformation by creating the leaf nodes (collectors) of the hierarchy (shown in Figure 6.7), and then create the other context nodes following a bottom-up strategy (i.e. children nodes are created prior to generate their parent node). The transformation is then defined as a non-tail recursive transformation, where each node generation calls a transformation for generating its children first.

For each node an operator is generated. The input for this operator is the combined output of the children nodes, and the output is the required context information for the parent node. For each context property that appears in any context situation in the MLContext model, the transformation generates a typed chunk. For example, for the user property "location" a "userLocation" chunk is generated (see Figure 6.10).

In the case of the flash sale scenario, the starting point for the transformation would be the context situation "flash_sale_alert" in the the MLContext application model.

```
situation flash_sale_alert (u:User, s:Shop) {
    user_near_shop(#u,#s)[QUALITY_HIGH]
    AND
    _flash_sale_offer_matching_user_info(#u.preferences,#s.saleOffer)==true}
```

The transformation will generate a "flash_sale_alert" context node for computing the situation. This node will be the root of the hierarchy like in Figure 6.7. The body of the context situation defines an AND operation between two values: the result of calling the "user_near_shop" context situation, and the result of calling the"flash_sale_offer_matching_user_info" external function, so two children nodes are generated for computing these values, as depicted in .

The "user_near_shop" node computes the following context situation:

```
situation user_near_shop(u:User, s:Shop) {
    _AdjustedLocation(#u.location,_Speed(#u.location))==#s.location
}
```

This context situation compares the return value of the call to the "AdjustedLocation" external function with the shop location (*#s.location*). To compute those values, two context nodes are generated, as children of the "user_near_shop" node (see Figure 6.9): "Adjusted_Location" and "shop_location".

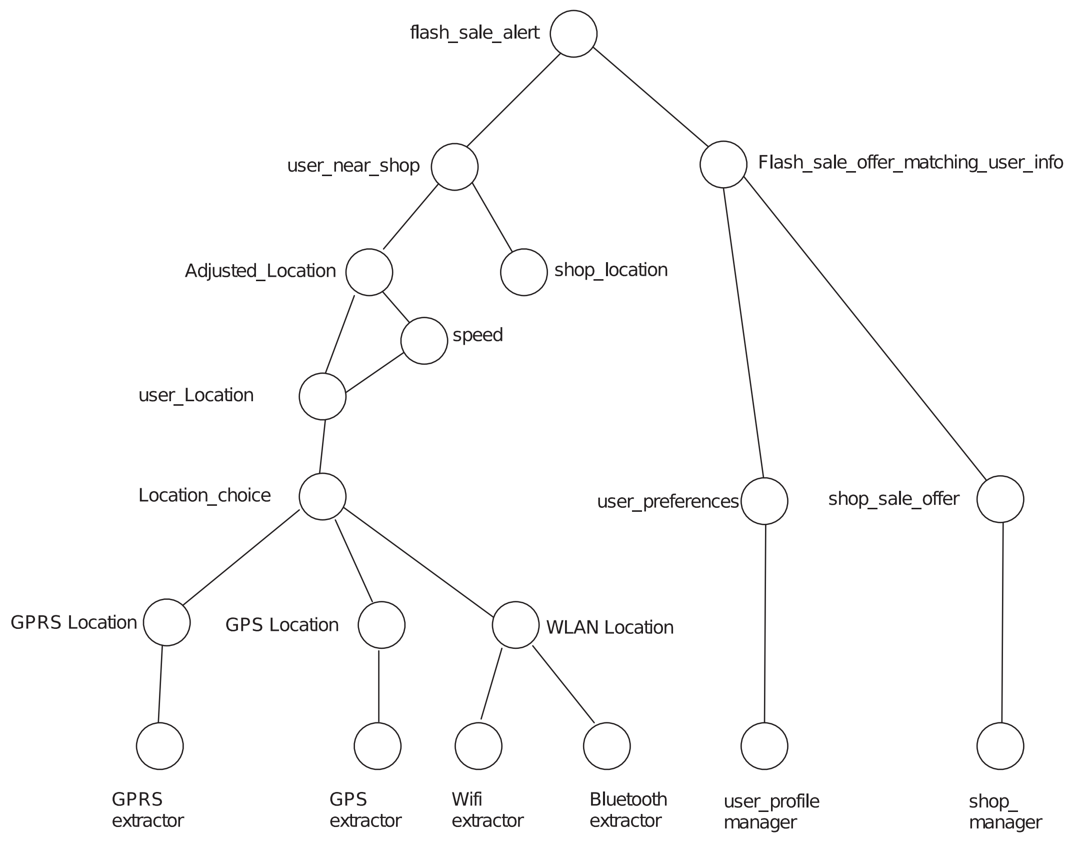


Figure 6.9 Partial view of the generated tree.

As shown above, the Adjusted_Location function has two parameters, the user location (*#u.location*) and the speed vector of the user (computed by the "Speed" function).Since these values are needed to compute the adjusted location of the user, the transformation generates two children for this node: "user_Location" and "speed" as observed in Figure 6.9. Note that both functions, the Adjusted_Location and the Speed function, take the user location as a parameter, so this node is shared by them.

The user_location node must get the user position value. This is a property defined in the Context model whose value can be supplied by several sources of context as shown in the Context model in Section 6.5.2. In this case, there are three sources of context (GPRS, GPS and WLAN) and the transformation generates a context node for each of them, plus an additional node "Location_choice". The "Location_choice" node is responsible for choosing the best (more suitable) source of context for the user position property (see Figure 6.9).

Finally, each source of context has one or more providers, so the transformation generates a collector node for each provider of the application model (see Section 6.5.3): "GPRS extracter", "GPS extracter", "Wifi extracter" and "Bluetooth extracter". Collector nodes are leaves of the hierarchy (see Figure 6.9).

```
package every2mallqoc;
chunk userLocation = {
   classname : every2mallqoc.UserLocationChunk,
   typeparam : every2mallqoc.UserLocationInfo
}
message userLocationMsg = {
   userLocation, freshness, accuracy, trustworthiness
}
operator computeGprsUserLocation = {
   output : userLocationMsg,
   classname : every2mallqoc.ComputeGprsUserLocation,
   input : {userLocation}
}
node gprsUserLocation = {
   operator : computeGprsUserLocation
   children : 3g_cell_id
}
node userLocationChoice = {
   operator : processUserLocation
   children : gprsUserLocation gpsUserLocation
              wlanUserLocation
}
operator computeUserLocation = {
   output : userLocationMsg,
   classname : every2mallqoc.ComputeUserLocation,
   input : userLocationMsg
}
```

```
node userLocationNode = {
   operator : computeUserLocation
   children : userLocationChoice
}
operator computeAdjustedLocation = {
   output : adjustedLocationMsg,
   classname : every2mallqoc.ComputeAdjustedLocation,
   input : userLocationMsg speedMsg
}
node adjustedLocation = {
   operator : computeAdjustedLocation
   children : userLocationNode speed
}
configuration : 3g_cell_id
   [observethrough=false] [periodobserve=xxxx]
configuration : userLocationChoice
   [observethrough=false] [periodobserve=xxxx]
configuration : userLocationNode
   [observethrough=false] [periodobserve=xxxx]
configuration : adjustedLocation
   [observethrough=false] [periodobserve=xxxx]
```

Figure 6.10 Partial view of the generated code.

Communication between nodes is carried out by means of messages. Generated messages contain the needed context information and the quality metrics (if any). For example, a "userLocationMsg" message is generated as shown in Figure 6.10. This message contains the "userLocation" chunk plus the quality metrics freshness, accuracy and trustworthiness. These quality metrics are included in all messages of the "user_near_shop" subtree, because we specified the "QUALITY_HIGH" requirement for this situation (see Section 6.5.3) which includes them.

Figure 6.10 shows an excerpt of the generated specification which can be compared with the original specification in Figure 6.8. Note that the generated tree in Figure 6.9 is equal to the original tree in Figure 6.7. The only difference (besides generated names) is that in Figure 6.7, the "flash_sale_offer_matching_user_information" function also computes if the user is near the shop. In our example we have kept it in a separated situation (user_near_shop) to improve clarity and legibility.

We have also generated a separate file with the quality levels required by the application.

Finally, we should mention that like in SAMURAI (see Section 6.6.2) there is not any module in COSMOS for computing the global quality of a node. If a developer is interested in computing a value for it, he/she must manually complete the generated Java code for the node operators.

6.6.2 Generating code for the SAMURAI framework

As mentioned in Section 3.1.3, SAMURAI uses: (1) bean classes, which are generated at runtime from a Spring descriptor (a XML file), to represent the context information, and (2) JavaScript functions to represent the context situations which can be evaluated at run-time by the embedded JavaScript interpreter. Our MLContext engine is able to generate both the Spring descriptor and the JavaScript functions.

For each entity in the MLContext context model the engine generates a bean description for a class of type "mlcontext.Entity", and a bean property for each of the properties of the entity. When a property obtains its value from a source of context, this property references the source of context in the Spring descriptor file. For example, in Figure 6.11, the "location" property for user "Celina" references a source called "Celina.location" of type "mlcontext.Source". This source also has a property named "history" whose value is set to "true" to keep a trace of the location values over the time.

Since a context source can have one or more providers, the engine also generates a bean description of type "mlcontext.ContextSource" for each of the providers. For example the code in Figure 6.12 shows the bean descriptors generated from the GPRS source and the corresponding provider. The provider description also has several entry keys which reference the methods defined for the source interface. These methods have their own description in the xml file. The properties specified for the methods in the provider definitions, such as quality parameters of the sensors, are included in the corresponding methods of the generated XML descriptors.

```
  <!-- Entity Celina -->
  <bean id="Celina" class="mlcontext.Entity">
      <property name="environment">
          <bean class="mlcontext.Environment">
              <property name="location" ref="Celina.location" />
         </bean>
     </property>
      <property name="personal">
          <bean class="mlcontext.Personal">
              <property name="preferences" ref="Celina.preferences" />
              <property name="carrying">
                  <map>
                      <entry key="smartphone" value-ref="smartphone" />
                  </map>
              </property>
         </bean>
     </property>
  </bean>

  <!-- Source Celina "location" -->
  <bean id="Celina.location" class="mlcontext.Source">
      <property name="history" value="true" />
  </bean>
```

Figure 6.11 Partial view of the generated code for user Celina.

```
<!-- ContextSource GPRS -->
  <bean id="GPRS" class="mlcontext.ContextSource">
      <property name="interfaceID" value="3G_cell_id" />
      <property name="methods">
          <map>
              <entry key="getLocation" value-ref="GPRS.getLocation" />
          </map>
      </property>
  </bean>

  <!-- Method GPRS "getLocation" -->
  <bean id="GPRS.getLocation" class="mlcontext.Method">
      <property name="returnValue" value="WGS84" />
      <property name="units" value="meters" />
      <property name="accuracy" value="500" />
      <property name="supply">
          <map>
              <entry key="Celina.location" value-ref="Celina.location"
/>
          </map>
      </property>
  </bean>
```

Figure 6.12 Generated code for the GPRS provider.

Finally, a Category section is generated for each of the categories in the Context model. When an entity does not belong to any category a new category is generated for it (e.g. the category "C_smartphone" in Figure 6.13 for the entity "smartphone").

```
<!-- Categories -->
  <bean id="Categories" class="mlcontext.Categories">
      <property name="User">
          <map>
              <entry key="Celina" value-ref="Celina" />
          </map>
      </property>
      <property name="Shop">
          <map>
              <entry key="shop1" value-ref="shop1" />
          </map>
      </property>
      <property name="C_smartphone">
          <map>
              <entry key="smartphone" value-ref="smartphone" />
          </map>
      </property>
  </bean>
```

Figure 6.13 Categories section.

Regarding the context situations in the model, the engine generates a JavaScript function for the starting situation "flash_sale_alert", and another one for each of the context situations it calls (see Figure 6.14). As each context situation can have its own quality requirements, they are combined with those of the context situations it calls. Quality requirements are included in the JavaScript function as conditional statements. An excerpt of the generated code is shown in Figure 6.14 where the JavaScript function "user_near_shop" includes the conditional statements for the quality requirements "QUALITY_HIGH". Those quality requirements were specified in the MLContext Application model.

As we have mention in Section 6.1.1, a context-aware middleware should have some module with the capability for computing the global quality of a context situation by using the quality of its pieces of context information. SAMURAI does not offer a generic building block to compute this global quality, but the developer can add the necessarily logic by making use of the complex event processing and machine learning building blocks embedded in SAMURAI, or by implementing a dedicated JavaScript function to manually compute this value. An example of this kind of module and a quality algorithm can be found in (Hoyos et al., 2011). However it is not necessary in our case study example.

```
function flash_sale_alert(u, s) {
  if (user_near_shop(u, s)&&
     (compareEqual(flash_sale_offer_matching_user_info(u.preferences, s.saleOffer)
      , "true"))) {
           return true;
  }
  return false;
}

function user_near_shop(u, s) {
  if ((compareEqual(AdjustedLocation(u.location, Speed(u.location)), s.location)
     && u.location.trustworthiness >="0.9"
     && u.location.freshness >="0.8"
     && u.location.accuracy <"10"
     && s.location.trustworthiness >="0.9"
     && s.location.freshness >="0.8"
     && s.location.accuracy <"10")) {
      return true;
  }
  return false;
}
```

Figure 6.14 Generated code for the context situations flash_sale_alert and user_near_shop.

6.7 Validation of the proposal

The advantages of using a DSL are well-known (Fowler, 2010; Völter et al., 2013). The main benefit is a gain of productivity motivated by an improvement in expressivity and legibility, and a reduction in the code size. Moreover, these improvements also ease the maintenance and portability, and can even facilitate a better communication with the final users. A DSL should only be built if the development cost is not high and a return of investment is feasible. It is therefore necessary to measure to what extension a productivity gain has been obtained, as well as the benefits achieved in other quality factors.

We evaluate the benefits of MLContext in Section 5.6. Now, we shall focus our evaluation in the MLContext extension for QoC, in particular discuss about expressiveness, portability and reusability and will measure the productivity gain. We demonstrate how a proper balance has been achieved by using the case study example.

Expresiveness

This is a typical benefit of using DSLs as they are simpler and more legible than general purpose languages. The syntax of the MLContext constructs for QoC has been taken from the context-aware QoC domain, therefore it has a higher level of abstraction than, for example, the XML code of SAMURAI. In the case of COSMOS, this framework also has a DSL but it is used to describe at a low level of abstraction a component-based software architecture, so it is difficult to figure out what is the context situation that is described and how it can be computed.

Portability and reusability

There are some context-aware middleware platforms with QoC support, and each of them uses a different format for representing the QoC information (e.g. a Spring descriptor or Java classes). MLContext QoC is based on a generic Ecore metamodel which represents the context information in a platform-independent way (see Section 6.2). In the case of SAMURAI and COSMOS we have again shown that platform-specific code can be generated from MLContext models, in particular code related to the QoC. MLContext can therefore be used in any middleware platform for which a mapping between the MLContext metamodel and the target platform can be defined (i.e. a transformation engine can be written). It should be noted that the code generation will be limited by the capabilities of the target platforms regarding the context management. With regard to the model reuse, the MLContext approach organizes the information typically included in a context model into two models: the context model and the application model. The context model does not contain application specific code, so it is easy to reuse this model within other context-aware applications. The application model (which contains the QoC information) however can not be reused in other applications because it has been written for a specific application though it remains platform-independent.

Productivity improvement

The gain in productivity is commonly measured by calculating the effort saved in writing code for the final application. We assume that writing code in

SAMURAI or COSMOS is not more difficult than writing code in MLContext because XML and COSMOS DSL have a short and well-defined grammar which makes easy to write a specification using them. MLContext grammar is short and easy to learn as well, as validated in Section 5.6. As we are specifying the same case study in MLContext that in the COSMOS and SAMURAI frameworks, the preliminary effort of abstracting and identifying the context elements which must be represented is the same for these two frameworks. Therefore, to calculate the gain in productivity we have measured the difference between the lines of code to write an MLContext specification and the lines of code (LOC) to write the artifacts which are automatically generated.

Table 6.3 Number of required LOC for defining each element.

COSMOS		MLContext	
Artifact	**LOC**	**Artifact**	**LOC**
Collector node	3	Provider	6 (1)
Context node	4	Context source	6
Operator	5	Entity	1+1 (2)
Chunk	4	Context situation	(3)
Message	3		
Node configuration	1		

Firstly we are going to calculate the gain in productivity in the case of COSMOS. Note that our code generation is a systematic process which generates the COSMOS code from the context situation in MLContext. This process allows us to estimate the lines of code that will be generated.

Table 6.3 shows the number of lines of code required to define each of the elements of COSMOS DSL and each of the elements of the MLContext model. The value of (1) is an average value, and depends on the number of quality parameters specified. The value of (2), must add one line for each context property of the entity. The value of (3) may vary depending on the specific context situation that is being modeled as explained below.

Knowing the number of LOC needed to define each element in COSMOS and MLContext, we can estimate the number of LOC that will be generated by the transformation engine and the gain in productivity, as we outline below.

As we know, from Section 6.6.1, a COSMOS nodes hierarchy is generated from an MLContext context situation. Therefore, we need to consider what elements are necessary from the MLContext model to compute the context situation (by inspecting the context facts in the body of the context situation) and how many LOC will be generated for each one. We will suppose that there is only one context fact per LOC in the body of the context situation. This is a reasonable assumption to improve the readability of the specification.

If we can estimate the number of LOC necessary to define in the MLContext model those elements involved in the computation of a context fact, and the number of LOC in the COSMOS specification generated from it, we will be able to estimate the gain in productivity for each LOC in the body of the context situation.

First, we must realize that each context fact in the body of the context situation makes use of, at least, one context property (two on average) to compare its value (e.g. Celina.location == shop.location). This property had to be previously defined in the body of an entity in the Context model (e.g. entity Celina), and it may obtain its value for one or more context sources (e.g. WLAN, GPS). These context sources may also have one or more providers attached (e.g. WIFI manager and Bluetooth manager for the context source WLAN).

To specify all those elements in the MLContext model, a different number of LOC have been necessary to write depending on the scenario. In the best-case scenario (only one context property is compared with a constant and this property obtains its value from only one source of context with one provider), the number of LOC is 14: 2 lines for defining the entity Celina with the property location plus 6 lines for defining the context source plus 6 lines for defining the provider (this total would be 15 LOC if we consider the line of code used to specify the context fact in the body of the context situation).

In the worst-case scenario (we suppose that there are two context properties in the context fact, and each property can obtain its values from two context sources, each one with one provider), the number of LOC is 40: 4 lines for defining the entities with the properties plus 24 lines for defining the four context sources plus 12 lines for defining the two providers (this total would be 41 considering the line used to specify the context fact).

Therefore we need to write from 15 to 41 LOC (according to the scenario is the best or worst case, respectively), to create an MLContext model which allows a context fact in the body of a context situation to be fully defined.

Next we will estimate the number of LOC of COSMOS that will be generated from one context fact.

Considering the best-case scenario previously defined, the transformation engine will generate a context node to compute the context fact and an operator and a configuration specification for the context node. With respect to the context property, a context node and a collector node are generated for their context source and provider, respectively, and also a chunk for the type of value of the context property. Finally, a message which encapsulates the chunk is generated. The total number of LOC generated would be 29: 4 (the context node for computing the context fact) + 4 (the operator) + 1 (node configuration) + 9 (the context node with operator and configuration for the context source) + 4 (collector node and configuration for the provider) + 4 (the chunk) + 3 (the message).

In the worst-case scenario, the transformation engine will generate a context node to compute the context fact with an operator and a configuration specification. A chunk and a message for each of the context properties, a context node (with operator and configuration) for each of the context sources and a collector node for each of the providers, will be generated. As there are properties which can obtain their values from more than one context source, a context node for choosing the best suitable context source will be also generated. If the context situation contains several context facts they will be connected by a relational operator (e.g. AND, OR) and another context node is generated to compute the relational operation.

Finally, if the context fact contains one or more call to external functions (e.g. Adjusted_location(u.location, Speed(u.location)) == s.location) a context node for computing each of them and a chunk and a message for each of the returned value will be generated.

The number of LOC generated for the worst-case scenario would therefore be: 10 (context node with operator and configuration for computing the context fact) + 2 x 7 (a chunk and a message for each of the context properties) + 4 x 10 (four context nodes with operator and configuration for the context sources) + 2 x 4 (a collector with configuration for each provider)

+ 2 x 10 (two context nodes with operator and configuration for choosing context sources) + 10 (context node for computing the relational operation) + 2 x 17 (context node for computing two external function call with a chunk and a message). This is a total of 136 LOC.

Therefore a total between 29 and 136 LOC (according the scenario is the best or worst case) will be generated in COSMOS from the context fact. This is a estimated gain in productivity between 48% and 70%. These percentage values have been calculated as 100 - (MLContext LOC x 100 / COSMOS LOC).

With regard to SAMURAI, we can do a similar reasoning that we had made with COSMOS, with the mapping from the MLContext model, as the generated elements always have a fixed number of lines in SAMURAI. The estimated productivity gain in this case is between 72% and 83%.

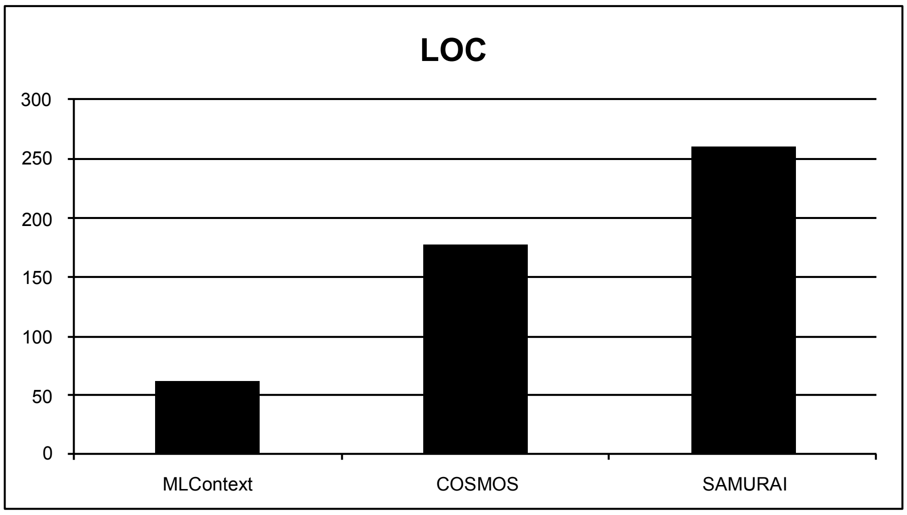


Figure 6.15 Comparison of LOC for the flash sale scenario example.

Figure 6.15 shows a LOC comparison for the case study. In the case of the flash sale scenario example, we have written 32 lines of code for the context model and 30 lines of code for the application model. Then, we have automatically generated 177 lines of code for the COSMOS framework and 260 for the SAMURAI framework. This is a gain in productivity of 65% and 76%, respectively.

7 Conclusions and future work

What now?

Tomorrow is often the
busiest day of the week.

Spanish Proverb

Context-awareness refers to systems that can both sense and react to their environment. Developing reliable context-aware applications continues to be a great challenge, even after a decade of research in this area, and one of the main difficulties that developers of this kind of applications must tackle is how to manage and represent the context information needed. Handling context is a crucial activity in the development of context-aware systems, which involves three main tasks: capturing the context, representing the context and defining adaptation behavior when the context changes. Moreover, the quality of the context information upon which these systems rely must be considered. Although sensors assume that this information is complete and accurate, their limitations may affect its quality and consequently lead to undesired behavior.

Various model-based approaches with which to tackle the complexity of context management have been proposed since the early years of context-aware computing. However, existing context modeling proposals have some limitations, as discussed in Chapter 3, and the quality of context aspects is normally neglected. In order to overcome these limitations, this thesis proposes an MDE approach that is based on a DSL whose purpose is context and context quality modeling. The capabilities of this language to generate software artifacts related to context management have been demonstrated by creating DSL engines for different context-aware middlewares. A simple context-modeling method has also been presented.

This is the final chapter of this manuscript. Some conclusions will be drawn and reflections presented (Section 7.1); contributions (Section 7.2) and cites to the contributions (Section 7.3) will be presented, and future work will be outlined (Section 7.4). In Section 7.5, the results of this work in terms of publications, projects, research stays and grants will be presented.

Next, we will discuss the level of achievement of the goals of this thesis that were presented in Chapter1.

7.1 Discussion

In Chapter 1, four objectives were defined: (1) to create a Domain-Specific Language for context modeling with a high-level of abstraction; (2) to extend the DSL in order to specify the quality of the context information and the quality requirements; (3) to automatically generate software artifacts for different context-aware middlewares; and (4) to define a method for context and QoC modeling.

A discussion on how these objectives have been achieved is shown in the following sub-sections.

7.1.1 Goal 1

The first goal was to create a DSL for context modeling with a high-level of abstraction.

As is explained in Section 1.3, we followed the design science research methodology (DSRM). Firstly, we made a review of the state of the art so as to attain comprehensive knowledge of the proposals published about context modeling (see Chapter 3). A domain analysis was then performed in order to elicit and describe the domain concepts and the relationships between them (see Section 5.1.1). This domain analysis allowed us to identify the requirements for a context modeling DSL with a high-level of abstraction, which were described in Section 5.1.2. All the requirements were met, as explained below.

Requirement: High-level abstraction

In order to achieve a high-level of abstraction, the language provides constructs that are close to the domain concepts (e.g. entity, context or type of context). In the design of MLContext, the main quality criteria and guidelines proposed as regards building a DSL have been considered (Kolovos et al, 2006; Hermans et al., 2009) (see Section 2.4.3). In this respect, the expressiveness of the language is achieved because all the concepts defined in MLContext correspond to relevant domain concepts, and each element of the language represents exactly one distinct concept in the domain (i.e., the language is orthogonal). Moreover, the semantics of the domain is implicit in the language notation, as explained in Section 5.6.

On the one hand, we have keep MLContext as simple as possible by defining its grammar with a low number of terminals (Table 5.2). On the other hand, MLContext is easy to learn and also to use, which are requirements related to a high-level abstraction that will be commented on later. The language is therefore capable of expressing the concepts of interest with ease.

The high-level of abstraction makes the development process simpler and more understandable with regard to a low-level of abstraction and encourages users who are not developers to participate in context modeling.

An MLContext specification could be written by a domain expert (i.e., a person with considerable knowledge on a given domain but, normally, with no or little expertise in programming). It therefore favors the interaction between the developer and the user, particularly when validating the context model.

Finally, MLContext promotes the writing of shorter context specifications. This has been measured by comparing the number of lines of code (LOC) necessary to write a context model specification using MLContext and using other context-aware frameworks (Sections 5.6 and 6.7).

Requirement: Support for modeling different types of context

When modeling context, it is necessary to distinguish among different types of context information. In this respect, a context taxonomy is useful to appropriately model the concepts, as explained in Section 5.1.1. Rather than having only a single type of context, MLContext allows different types of context to be modeled, such as physical, social or computational context, because this is closer to reality.

The state-of-the-art review allowed us to identify the most commonly used context taxonomies. We found that some authors do not consider certain types of context because their approaches are focused on specific domains, such as the management of web services or e-learning systems, which do not consider, for example, physical context. There is also a disagreement as to the type of information assigned to a category of context and, as is shown in Table 5.1, the same context information is assigned to different types of context depending on the approach considered. For example, some authors also consider environment information for social, computing, user or task types.

Our proposal covers all the types of context identified by classifying them in six categories (see Section 5.1.3). We have also explicitly defined what type of context information is included in each category. A context taxonomy allows the context information to be more precisely expressed in models. The definition of a context taxonomy has other benefits. Separating information into multiple contexts can also be exploited to achieve improvements such as more efficient storage and retrieval methods. For example, an application might only be interested in context information of the physical type.

Finally, it is worth mentioning that we have designed the abstract syntax of MLContext by bearing in mind the extensibility requirement for DSLs (Section 2.4.3). New types of context can therefore be easily incorporated into the DSL if necessary.

Requirement: Platform-independent and application-independent models

On the one hand, platform-independent models free the user from having to provide implementation details. On the other hand, context should be modeled regardless of the context-aware application in order to promote the reuse of models. The MLContext metamodel allows the context information to be represented in a platform-independent manner. As explained in Section 4.1.2, context models do not contain application-specific code details. In our approach the information typically included in a context model is organized into two models: the Application model and the Context model, as shown in Figure 4.2. The Context model thus represents the context information, which includes the entities, their properties and the sources of context, and the Application model represents those aspects related to activities and situations (e.g. QoC information), along with information related to sensors which are specific to a particular application. This separation of concerns keeps MLContext models simple and readable and also has some benefits as regards evolvability. For instance, changes at the application level (e.g. one of the sensors is exchanged for another with higher accuracy) would not affect the Context model. It is worth noting that the Context model still retains enough information to make it useful for generating software artifacts related to context management, as is shown in this work.

Since the Context model does not contain application specific code, it is easy to reuse this model in other context-aware applications. The Application model cannot, however, be reused in other applications because it has been written for a specific application, although it remains platform-independent. We have validated model reuse by generating software artifacts for different context-aware frameworks from the same MLContext models.

Requirement: Ease of use

If developers are to effectively write specifications, a DSL should be usable, that is, easy and intuitive to use. This ease of use is also related to learnability, another quality requirement for the design of a DSL (Hermans et al., 2009) (see Section 2.4.3). Learnability refers to concepts such as language comprehension or ease of learning. A language that can be easily learned facilitates its use.

In order to define a DSL that is easy to learn, we took into account some of the criteria and guidelines proposed in (Kolovos et al, 2006; Oliveira et al., 2009; Völter, 2009). As mentioned previously, the MLContext constructs correspond to relevant domain concepts and each construct is used to represent one distinct concept in the domain. The semantics of the domain is therefore implicit in the language notation which allows users to easily create mappings between the objects of the problem domain and the syntax of the language. The fact that MLContext notation only expresses concepts of the domain makes the language efficient as regards being read and learned by domain experts. Moreover, MLContext is expressive and simple owing to the low number of terminal symbols in its grammar.

In order to support our claims on the simplicity and readability of MLContext, we measured some classical metrics by analyzing its grammar structure. These measurements were used to estimate the complexity of the language. The results of our analysis are shown in Section 5.6, along with a comparison of MLContext with other reference languages (Java and C).

7.1.2 Goal 2

The second goal was to extend MLContext with constructors and notation in order to specify the quality of the context information and the quality requirements of a specific application.

As explained in Section 6.1, the lack of information about the QoC can result in a degraded performance of context-aware applications. A DSL for context modeling must therefore have some constructs for QoC modeling. We carried out a domain analysis (see Section 6.1.1) in order to identify the main concepts to be taken into account in our QoC extension of MLContext. This analysis allowed us to conclude that quality of context was a multidimensional concern in the sense that several quality parameters need to be defined (e.g. accuracy, precision or range of the sensor). We therefore identified the most common parameters used to measure context quality. Our review of literature evidenced some confusion in the meaning of some parameters, which had different meanings depending on the author. We consequently clearly defined each of the parameters (see Section 6.1.1 and Appendix A) and then went on to classify the quality parameters according to three points of view: data acquisition, data representation and data usage.

Our study also revealed that some quality parameters had dependencies with others and we identified these dependencies (see Table 6.2). We have defined constructs with which to express all the quality parameters identified except those that had dependencies on others, because the value of these parameters cannot be specified at design time, but must be computed at run-time. However, they can be used to specify quality requirements at design-time. As quality parameters for data acquisition are related to the providers of the context information, we also defined a construct with which to specify providers.

With regard to the quality requirements for a context-aware application, they are related to context situations which are specific to this application. We therefore provided MLContext with constructs that will allow a developer to define context situations.

In order to maintain a proper balance between expressiveness and complexity of the language, we defined constructs that can be used to specify simple context situations, but we also created constructs with which to include extern function calls and parameterized context situations. These constructs, together with the possibility of composition of context situations, allows a developer to define more complex context situations.

In order to improve legibility so as to specify context situations with quality requirements, MLContext also provides a construct with which to specify quality levels as an aggregation of quality requirements. These quality levels are identified with a unique label which can be used to specify those quality requirements when defining a context situation.

QoC is defined in the MLContext Application model since this aspect is application dependent. As indicated above, the reuse of Context models is favored by separating application-dependent concerns, such as QoC, into the Application model.

7.1.3 Goal 3

The third goal was to show how the software artifacts related to context management could be generated for different context-aware middlewares.

MLContext is the core element of an MDE generative architecture used to develop context-aware applications. If the context model is used as a basis, other context-aware aspects can also be modeled with the purpose of automatically generating code for these applications. In this thesis, we have addressed the issue of automatically generating software artifacts for context and QoC management. In this respect, we have created several transformation engines that are in charge of transforming the models expressed with MLContext into software artifacts. More specifically, we have validated our approach by generating software artifacts for the OCP, JCAF, SAMUARI and COSMOS frameworks. The artifacts generated include OWL ontologies, OCP producers and consumers, JCAF code, XMI Spring descriptors, JavaScript code, COSMOS DSL specifications and also Java code (Hoyos et al., 2014) and geoSPARQL specifications[3].

One of the main difficulties to be overcome when building such transformation engines is the lack of information and documentation on the existing context-aware frameworks. Both detailed information on context-representation and examples of source code are essential when defining a mapping between the MLContext model elements and the target middleware. Published approaches only offer small examples and excerpts of the context representation, or a brief description of the software artifacts they manage. In our case, we have built engines for two middlewares with which we were familiar (OCP and SAMURAI), a mature and well established and documented framework (JCAF), and the COSMOS framework. In the last case, we had

[3] available at http://www.modelum.es/MLContext

documentation on the COSMOS DSL and the case study to be analyzed, which allowed us to identify the code generated for each COSMOS element.

It should be noted that in the case of context management, the code generation is limited by the capabilities of the target platforms. For example it is not possible to generate native code for QoC management of the OCP or JCAF frameworks because they do not provide native QoC support.

7.1.4 Goal 4

The fourth goal was to define a method for context modeling that would take the quality of context into account. In order to take full advantage of the specificities of our approach, in this work we have presented a proposal of a simple method with which to model context and QoC in the earlier stages of a context-aware application development process. This method recommends the steps to be followed in context modeling and is based on the particular characteristics of our approach. It was presented in Section 6.4, and takes into account the separation of concerns of our approach by modeling first the Context model and then the Application model, which refers elements of the previous model. As explained in Section 6.4, some steps can be performed in parallel.

We have validated our approach by following the steps in the aforementioned method when modeling the case studies used in this thesis (see Sections 5.4 and 6.5).

As explained in Section 7.2, our method has been adopted in other research works in order to develop context-quality aware applications.

7.2 Contributions

In this thesis we have defined an MDE approach for context modeling in the development of context-aware systems. We have therefore made several contributions: (1) A domain analysis on context and quality of context; (2) The MLContext language, which is a Domain-specific language that has been specially tailored to model context at a high level of abstraction; (3) An extension of this DSL in order to model quality of context; and finally (4) a domain-specific approach with which to automate the management of context in context-aware middlewares.

A domain analysis of context and quality of context.

We have performed a domain analysis with the objective of identifying and describing the main concepts to be represented by our context model and their properties, along with defining the requirements for our context modeling DSL.

The domain knowledge was gathered from sources such as technical documents, source code from context-aware frameworks or domain experts. We have analyzed the most relevant approaches dealing with context-awareness and context quality representation. To the best of our knowledge, this is the first time a domain-analysis on context has been conducted prior to the design of a context modeling language.

The requirements identified from this analysis were taken into account when designing and implementing the MLContext language. These requirements have become a reference for other research works (Carvalho et al., 2012; Quintalleri et al., 2015; Duarte et al, 2015).

The main publication we have produced regarding this topic is the following.

- Hoyos, J.R., Garcia-Molina, J. Botia, J.A. MLContext: A context modeling language for context aware systems. 3rd DisCoTec Workshop on context-aware adaptation mechanism for pervasive and ubiquitous services. CAMPUS'10. Amsterdam, Netherlands, 2010.

A Domain-Specific Language for context modeling at a high level of abstraction

In order to help context-aware systems developers in their task of modeling context and implementing the artifacts required to manage context, we have created MLContext, a DSL with a high-level of abstraction that is specifically intended for context modeling and the automatic generation of the software artifacts involved in context management.

MLContext has been designed to be platform-independent, signifying that developers do not need to deal with implementation or specific framework details. It has also been designed to be easy to learn and to use, and to facilitate communication among domain experts and developers. MLContext improves the productivity involved in the development process since the development effort is reduced and maintainability is favored.

A review of the state of the art of context modeling was recently presented in (Koç et al., 2014), in which MLContext is classified as having been designed by following a design-oriented research method that includes a proper validation.

Our DSL meets all the requirements for a context modeling language identified in the domain analysis. MLContext can be downloaded and tested[4].

The most important publication regarding this topic is:

- Hoyos, J.R., Garcia-Molina, J., Botia, J.A. (2013) A Domain Specific Language for Context Modeling in Context-aware Systems. Journal of Systems and Software, 1.135, vol.86, 11, 2890-2905. ISSN: 0164-1212

A quality of context modeling extension

We have created a QoC modeling extension for the MLContext language. This extension allows developers to model quality parameters for the

[4] http://www.modelum.es/MLcontext

providers of the context information, in addition to expressing quality requirements for context-aware applications.

As the quality of the context information normally depends on several quality parameters, our proposal includes constructors with which to define quality levels that can be used to specify quality requirements related to a context situation.

Marie et al (Marie et al., 2014) highlight that "MLContext offers the benefits of considering the QoC in terms of guarantees for the producers of context and in terms of QoC requirements for the consumers of context".

The most important publications we have produced regarding this topic are:

- Hoyos, J.R., Preuveneers, D., Garcia-Molina, J., Berbers, Y. A DSL for Context Quality Modeling in Context-Aware Applications. International Symposium on Ambient Intelligence (ISAMI2011). Salamanca (Spain), 2011.

- Hoyos, J.R., Garcia-Molina, J., Botia, J.A., Preuveneers, D. (2015). A Model-Driven Approach for Quality of Context in Pervasive Systems. Special issue on Pervasive Computing. Computers & Electrical Engineering. Under review

A domain-specific approach for context managing

MLContext is the core element of a domain specific approach used to automate the management of context in pervasive systems. One of the main characteristics of this approach is that it promotes a separation of concerns in order to favor the reuse and evolution of context models. Peinado et al (Peinado et al., 2015) explain that the main advantage of the MLContext separation of concerns is that it proposes the use of generic context models that can be reused for different applications.

Duarte et al (Duarte et al, 2015) point out that, MLContext separates the definition of the contextual elements from details concerning the sources, which is a desirable characteristic for a context modeling language. They also present a study for existing DSLs for context-aware mobile application development, which shows that MLContext is the only one of the DSLs compared that has middleware platform independence.

By creating several transformation engines, MLContext models can be used to create software artifacts for different context-aware middlewares.To the best of our knowledge, MLContext is the first context modeling language to have been validated by generating code for different middlewares.

Our proposal is accompanied by a method which guides developers in the modeling of context and quality of context for context-aware applications, as explained in Sections 6.4 and 7.1.4. Marie et al (Marie et al., 2014) comment on the MLContext method for modeling context and conclude that it is a process that greatly facilitates the modeling of context and quality of context, thus helping developers build context quality aware applications.

The publications we have produced regarding this topic are:

- Hoyos, J.R., Garcia-Molina, J., Botia, J.A. (2013) A Domain Specific Language for Context Modeling in Context-aware Systems. Journal of Systems and Software, 1.135, vol.86, 11, 2890-2905. ISSN: 0164-1212

- Hoyos, J.R., Garcia-Molina, J., Botia, J.A., Preuveneers, D. (2015). A Model-Driven Approach for Quality of Context in Pervasive Systems. Special issue on Pervasive Computing. Computers & Electrical Engineering. Under review

- Hoyos, J.R., GarciaMolina, J., Botia, J.A. Modelado de calidad de contexto con MLContext. XIX Jornadas de Ingeniería del Software y Bases de Datos (JISBD'2014). Cádiz (España), 2014. ISBN: 978-84-697-1152-1

7.3 Cites to the thesis contributions

In order to determine the impact of the results of this thesis published to date, we consulted the Web of Science and Google Scholar Citations, and the search revealed a total of 29 cites (excluding self-citations) of the articles published from this thesis. Some of these references are commented on below.

Peinado et al (Peinado et al., 2015) state that the separation of concerns is one of the main advantages of the MLContext approach because generic context models can be reused for different applications. They observe that adaptation-to-context code cannot be generated from MLContext models.

Quintalleri et al (Quintalleri et al., 2015) refer to our domain analysis and highlight some of the requirements we have identified for a context model such as support for different types of context or a high-level abstraction and they state that these requirements are fundamental features for modern context models.

Carvalho et al (Carvalho et al., 2012) present a computational context modeling framework for the development of context-aware web applications. They have used some of the requirements we have identified in the design of MLContext in addition to some of its characteristics in the creation of their framework (e.g. in order to categorize the information in context dimensions).

Marie et al (Marie et al., 2014) present the QoCIM meta-model which is intended for use in the design and representation of heterogeneous QoC criteria. They refer to some of the benefits of the MLContext approach: "MLContext offers the benefits of considering the QoC in terms of guarantees for the producers of context and in terms of QoC requirements for the consumers of context".

They also comment that the MLContext method for context modeling is a process that greatly facilitates the modeling of context and quality of context, thus helping developers when developing context quality aware applications. In order to extend the usability of QoCIM and improve its expressiveness so as to develop QoC-aware applications, they are planning to integrate QoCIM into MLContext in order to express QoC requirements and guarantees with a generic model of QoC criteria, rather than using the pre-defined list of criteria of MLContext.

Zhang et al (Zhang et al., 2014) refer to the design process of MLContext. They propose the MLContext design and development process as an example to be considered when constructing a modeling environment oriented toward Emergency Management (EM).

Koç et al (Koç et al., 2014) present a systematic literature review carried out to find answers to four research questions: (1) How much activity in the field of context modeling has there been since 2005 and who is active in this area; (2) What research topics are being investigated; (3) What research paradigms or methods are being used; and (4) Which topics need further research according to the authors. They classify MLContext as a DSL that has been designed by following a design-oriented research method and which has a proper validation. They also mention that the majority of context modeling research adopts a conceptual research paradigm and that those approaches suffer from a lack of proper validation of their structure and applicability.

Machara et al (Machara et al., 2013) propose metamodels for use in designing context contracts that define privacy and QoC agreements. They state that MLContext is a promising work, although it relates more to code generation than to exploiting the context model at runtime.

Duarte et al (Duarte et al, 2015) present an MDE approach for the modeling and generation of Context Aware Mobile (CAM) applications. Their approach includes ContextRuleML,a DSL for the modeling of contextual information and rule-based behavior. ContextRuleML is based in the MLContext metamodel and supports context rule definition. It has been built on the basis of the set of requirements we proposed when creating MLContext as regards context modeling and on requirements from a DSL survey of CAM applications related to adaptation behavior. The authors compare their approach with several DSLs proposed for context modeling by using a desired set of requirements as comparison criteria. As ContextRuleML is based on the MLContext metamodel, both approaches fulfill all the requirements related to context modeling. ContextRuleML also meets the requirements for application adaptation and context rules. It should be noted that MLContext is the only one of the approaches considered that has middleware-platform independence.

Maran et al (Maran et al., 2014) carried out a survey on context representations which includes the metamodel for the abstract syntax of MLContext. They state that MLContext representation is easy to understand and facilitate the development process of ubiquitous systems.

张鹏et al (张 鹏 et al, 2015) present a computational experiment for emergency management model building, in which a generic modeling environment and a discussion about metamodel design is presented. The authors explain the advantages of the MLContext approach.

Finally, some works (Al-Bashayreh et al., 2013) mention that MLContext is suitable for use in the biomedical informatics domain and can be helpful in the development of mobile patient monitoring systems.

7.4 Future work

The research we have presented in this thesis is not closed. There are several interesting directions that can be taken to provide our proposal with a wider spectrum of application. The research activities we are planning in order to continue our work are therefore the following.

To reuse MLContext models in different applications for the same framework.

MLContext has been designed to be platform-independent and to promote context model reuse. In this respect, we have applied a separation of concerns in order to keep the Context model free from specific-application or platform details. Our proposal has been validated by generating software artifacts for different context-aware frameworks. Currently, we plan to validate model reuse in different applications for the same framework by modeling and generating code for the following scenario.

A context model for an archeological museum will be specified by using MLContext. The context model will contain the different rooms of the museum, along with the different objects exhibited. Each object will be accompanied by a description of its history. The specification of the rooms will include environment information regarding the shape of the rooms, how they are connected and existing exits. The museum would have different kinds of sensors (e.g., presence detectors and fire detectors) that would be also modeled.

This context model will be reused to generate code for different applications, and we will create an application model for each of them: Examples of possible applications would be the following.

1) **A tourist guide system**. A context-aware application which can guide visitors through the museum and provide a description of the object the visitor is in front of. The application model will define a "visitor_in_front_object" context situation, which will use objects locations, visitor location and presence detectors.

2) **An emergency evacuation system**. This system will be in charge of evacuating visitors in the case of fire. If a fire is detected, it will trigger an alarm and alert the visitors. It can also let the visitors know where the fire is located and will guide them to the nearest exist. The application model will define an "alert_of_fire" context situation, and the application will use the fire detectors and the environment information to locate the nearest exist without crossing the fire.

3) **A surveillance system**. The museum is closed during thr night, and this application will be in charge of triggering an alarm if an intruder is detected. We will specify an "intruder_detected" context situation in the application model. The application will make use of the presence detectors and the system time information to detect nightly hours.

To add run-time support to the MLContext language.

We are planning to add run-time support to MLContext, thus allowing context models defined at design time to be used at run-time. At the present, context models can be used to generate software artifacts, but adding run-time functionality will allow us to define dynamic context models that can evolve. One obstacle to us achieving this is the fact that MLContext is integrated into the Eclipse environment. We are therefore planning to create an MLContext Java library which will allow Java programs to read and manage MLContext models outside the Eclipse environment.

A run-time model approach, involves several challenges: (1) development of the models, (2) evolution of the models, (3) supporting adaptation based on run-time models, (4) maintenance of fidelity of the run-time model with respect to the system and its environment.

In order to adapt its behavior, the system needs to reason about itself and its environment, which involves manipulating the models at run-time. The need to adapt can be raised not only based on current status and environment but also on predicted violations of functional properties. In this sense, changes would be planned by using the run-time model.

Our approach proposes a separation of concerns. However, this raises the issue of maintaining multiple models at runtime and keeping them consistent with each other. This requires some mechanims to manage relationships between models.

Finally, it should be noted that changes in the run-time model should be propagated to the running system. This requires mechanisms to map model changes to changes at the system level and vice versa.

To research how to handle scalability.

We are interested in researching how to handle scalability from an MDE perspective when the number of users, sensors, mobile devices and other context sources will grow orders of magnitudes as envisioned by the Internet of Things paradigm.

The growth of the context model would require additional resources such as computation power or memory. Moreover, data fusion, which is a technology (and methods) to enable combining information from several sources, would become a challengin task when the number of context sources increases exponentially.

In this respect, a small specification written in MLContext could be used by devices to interchange information in order to create and maintain the context model dynamically. A new device, e.g. a smarthphone, which is within the scope of the system, will therefore send a context specification with information about its user. This information will be incorporated into the MLContext model at run-time (i.e., a new entity will be added to the model). Technical details (in the form of services provided for the smartphone) will also be incorporated into the application model. These specifications could be expressed in MLContext and the context model could be distributed among several nodes.

To explore MDE strategies coping with specification and development of adaptive context-aware systems

MLContext promotes the separation of concerns. Therefore it would be possible to extend it to address other context-aware aspects like adaptation.

Adaptation is an essential feature in context-aware systems which are designed to assist the human in the long term. Think for example on a companion robot or an intelligent workplace. These systems need to model the user or groups of users to specifically tailor themselves to their particular habits and interests. Adaptive systems are enabled with basic machine learning features which act on building user models that change during time. Thus, such models are on-line learning models. The specification of an adaptive context-aware system through a DSL should allow to specify, for each service offered, what are the adaptive capabilities to enable, its reactiveness to change (i.e. should a 24/7 monitoring system at home adapt its behaviour to the fact that the user is in bed because of a cold?), the level of accuracy required in predictions, the tolerated level of disruption of the system into the user tasks among others. Nowadays, the specification of such properties by a DSL is easier than enabling the technologies provided in the substrate (i.e. effective and accurate user model learning). Thus, this line of research is particularly challenging.

In order to start paving the way for such a cutting-edge technology, we should start by defining basic mechanisms for simple but effective user modelling on basic scenarios and add new building blocks on the basis of basic but reliable software components. Thus, it is not only about machine learning but also about specification of scenarios and identification of all the possible situations in which the system can be really effective.

7.5 Publications

The research activity of this thesis has produced various innovative contributions that have been presented and discussed on several peer-review forums. The articles in which the research from this thesis has been published are presented below.

7.5.1 International journals indexed in the JCR

- Hoyos, J.R., Garcia-Molina, J., Botia, J.A. (2013) A Domain Specific Language for Context Modeling in Context-aware Systems. Journal of Systems and Software, 1.135, vol.86, 11, 2890-2905. ISSN: 0164-1212

- Hoyos, J.R., Garcia-Molina, J., Botia, J.A., Preuveneers, D. (2015). A Model-Driven Approach for Quality of Context in Pervasive Systems. Special issue on Pervasive Computing. Computers & Electrical Engineering. Under review

7.5.2 International conferences

- Hoyos, J.R., Preuveneers, D., Garcia-Molina, J., Berbers, Y. A DSL for Context Quality Modeling in Context-Aware Applications. International Symposium on Ambient Intelligence (ISAMI2011). Salamanca (Spain), 2011.

- Hoyos, J.R., Garcia-Molina, J. Botia, J.A. MLContext: A context modeling language for context aware systems. 3rd DisCoTec Workshop on context-aware adaptation mechanism for pervasive and ubiquitous services. CAMPUS'10. Amsterdam, Netherlands, 2010.

  This paper was ranked among the best papers of Context-aware Adaptation Mechanisms for Pervasive and Ubiquitous Services (CAMPUS 2010) and requested for an extension to be included in the special issue of Scientific Annals of Computer Science.

7.5.3 Other journals

- Hoyos, J.R., Preuveneers, D., García-Molina, J., Berbers, Y. (2011) A DSL for context quality modeling in context-aware applications. Advances in Intelligent and Soft Computing, vol 92, 41-49. Springer. ISSN: 1867-5662. This is a publication of the article from ISAMI2011.

- Hoyos, J.R., García-Molina, J., Botia, J.A. (2010). MLContext: A Context-Modeling Language for Context-Aware Systems. Electronic Communications of the EASST, vol 28, 2010. ISSN 1863-2122. This is a publication of the article from CAMPUS'10

7.5.4 National conferences

- Hoyos, J.R., GarciaMolina, J., Botia, J.A. Modelado de calidad de contexto con MLContext. XIX Jornadas de Ingeniería del Software y Bases de Datos (JISBD'2014). Cádiz (España), 2014. ISBN: 978-84-697-1152-1

7.6 Projects related to this thesis

- **Foundation for the development of AAL services and applications** (Fundamentos para el desarrollo de Servicios y Aplicaciones AAL). Ministerio de Ciencia e Innovación. From 01/01/2012 to 12/31/2014. TIN2011-28335-C02-02.

- **CADUCEO: Creation and Validation of e-Careservices in Hospitals** (CADUCEO-Sistema de creación y validación de servicios hospitalarios sobre la plataforma Cardea). From 09/01/2011 to 12/31/2013. (IPT-2011-1080-900000), INNPACTO Program, 2011 founded by Ministry of Economy and Competitiveness (Spain) and the European Regional Development Fund (ERDF).

7.7 Research stays

- **Research stay at the Katholieke Universiteit Leuven (Belgium) for 3 months** (07/01/2010 – 09/30/2010) collaborating with the DistriNet research group. During the stay we researched context quality modeling topics from an MDE perspective. As a result of this collaboration, we have created a quality context extension for the MLContext language (Hoyos et al., 2011).

7.8 Grants

- **Scholarship for student mobility stays to obtain the European Mention** in doctorate degrees with a quality mention. Ministry of Education. Spain. EDU/2933/2009. TME2009-00278

A
Quality parameters definitions

This section explains in detail each of the quality parameters defined in Section 6.1.1. These parameters are the most common parameters used to measure the context quality in the literature. We have classified them in three categories: data acquisition, data representation and data usage.

Data acquisition

The quality parameters associated with data acquisition are directly related with the sensors which provide the information. Therefore, they can be defined at the sensor level.

Resolution

It is "the fineness to which an instrument can be read or the smallest change in the underlying physical quantity that produces a response in the measurement" (Carr et al., 2010). For example, a sensor with a resolution of 0.005 grams can obtain measures like 1.130 gr, 1.135 gr, etc.; and a sensor with a resolution of 0.001 grams can obtain measures like 1.125 gr, 1.126 gr, etc. Note that measurements could be imprecise (inaccuracy) in both cases as we will see later. An application interested on a granularity of milligrams should choose the second sensor, but it should also take into account other factors like the accuracy or precision of the sensor.

Precision

Precision is "the degree to which repeated measurements under unchanged conditions show the same results" (Carr et al., 2010). Precision it is also called repeatability or reproducibility. An ideal sensor would output the same value every time the value is measured, but real sensors output a range of values relative to the actual correct value. For example, let suppose a sensor with a resolution of 0.001 gr. If we apply a weight of exactly 1.110 gr. the output values from the sensor will vary considerably (e.g. 1.121 gr., 1.100 gr., 1.107 gr., etc.). As we can see, successive measures differ from the second decimal position. This is because, in this case, the precision of the sensor is 0.1 gr.

While a weather forecast system does not need to know a precise location of the user to provide a weather prediction for his/her area, a navigation system does, in order to supply a precise route. The difference between resolution and precision is repeatability, but we must also take into account that a very precise measurement (e.g. a repeatable measurement which always give us a value of 1.128 gr.) could be a wrong measurement (e.g. when the true value is 1.120 gr.).

Accuracy

The accuracy of a measurement is "the degree of closeness of a quantity to its actual (true) value" (Carr et al., 2010). Therefore the accuracy of a sensor is the maximum difference between the true value and the supplied value at the output of the sensor. Actually, despite its name, this parameter expresses inaccuracy, and it can be expressed as a percentage or in absolute terms. The difference between precision and accuracy is correctness. For example, a precise measurement of 1.128 gr. (with a resolution of 0.001 grams) in our previous example could be wrong if the true value is 1.120 gr.

Accuracy is a very important quality parameter for context-aware systems though, sometimes, a source of context can supply biased information to protect the user privacy (e.g. not sending his/her exact location to some service).

Range

The range of the sensor is "the maximum and minimum values of applied parameter that can be measured" (Carr et al., 2010). For example, the output of a weight sensor will be wrong when we apply a weight of 1020 grams if it has a range from 0 to 1000 grams, because it exceeds the maximum weight. A context-aware application should take this into account. The range of a sensor can be expressed by means of its maximum and minimum values.

Freshness (Timeliness)

Freshness refers to "how current the provided information is at the time of delivery" (Kahn et al., 2002). A source of context should indicate the time of the data acquisition. For example, a navigation system needs updated information about the user's location). While some information remains valid over time, other may become discredited or obsolete. The frequency of the sensor readings determines the freshness of the measurement.

Location

This parameter refers to the physical location of the sensor. For example, a context-aware application might be interested on identifying all the devices a user is carrying. This parameter must not be confused with coverage.

Coverage

The coverage of the sensor defines "the scope of the information it provides" (Preuveneers et al., 2007). Normally, it is a geographic scope which coincides with the location of the context source, but not necessary (e.g. an adapter, that is inside a room, which can supply information about the outside temperature).

Data representation

The quality parameters concerning data representation can be used to specify quality requirements related to the representation of information within context-aware systems.

Units

A unit is "a quantity chosen as a standard in terms of which other quantities may be expressed" (Oxford dictionary). It can be used to express the value of data information. On the one hand, sources of context may provide a value using different units from those required by an application. On the other hand, the application could not know the units in which a value is expressed. Therefore, it can not use the information (without units is ambiguous information). A context-aware framework should be able of converting the value supplied by the source to the units required by the application.

Format

Data format refers to the data structures used to represent the information. The format of the context information depends on the context source which supplies it (e.g. string, boolean, float, or more complex structures). A context-aware application can not manage information which is in an unknown format.

Understandability

Understandability is "the extent to which data is easily comprehended by the information consumer" (Pipino et al., 2002). Understandability is related to data format and semantics. Format determines whether information is represented using an appropriate notation or not. Understandability refers to the capability of the information consumer to comprehend the semantics of the information. For example, an application needs to know what "temperature" is in order to use this information or, at a higher level of abstraction, the meaning of a context situation like "the user is sleeping".

Aliases

Some context providers may use semantically related terms or synonyms (e.g. Corridor and Hallway) to refer to the same value. If a context-aware application does not know a specific term, it can not use the associate information, even if it knows the meaning of other terms which are synonyms (Preuveneers et al., 2007). A context-aware framework might provide some aliases for the application. This is not a frequent issue because most of the sensors supply numerical values instead of semantic terms.

Data usage

Quality parameters associated with data usage, normally, depends on the context of the information (i.e. to know if some data are relevant we need to know the context situation).

Believability (Trustworthiness)

Believability is "the extent to which information is regarded as true and credible" (Wang et al., 1996). Sometimes, it is seen as expected accuracy, but there are some differences. While accuracy refers to the verifiable precision of a value supplied by a sensor, believability refers to trusting the value of the data without checking it. This is an important quality parameter for forecasts, which can not be verified in advance.

Believability is a subjective parameter, as there are different subjective procedures to decide whether the supplied information is true or not. The believability of a data value depends on two factors: the origin (the context source which supply the value) and the subsequent processing history.

Reliability is part of the believability because it captures the error rate of a context source (the error rate can be measured objectively by training or observation).

Completeness

Completeness is "the degree to which information is not missing" (Pipino et al., 2002). Generally, the literature views a set of data as complete if all necessary values are include (Ballou et al., 1985). This parameter is related to ambiguity, and applies not only to context data but also to quality context metadata. For example, if an application needs the temperature and pulse of a patient but the sources only provides the pulse, probably, it can not determine his/her state of health (context data are incomplete). Moreover, even if the temperature and pulse are available, if the application does not know the accuracy of the sources or the units of the measurements it can not use the context information (can not determine the quality of the information or it is ambiguous).

Relevance

Relevance is "the extent to which information is applicable and helpful for the task at hand" (Borlund, 2003) and is generally divided into two categories: topical relevance and user-centered relevance (Kagolovsky et al., 2001). Topical relevance is objective and it is concerned with terminology (e.g. temperature is relevant to determine the health's status of a patient). It can be judged by domain experts. User-centered relevance is subjective to the user (e.g. information about a new shopping center in the city).

Relevance can be seen as the relation between the situation, task, or problem at hand, and the resource. This is an important quality parameter for some context-aware systems (like web-based systems), as information consumers must deal with an overflow of potentially relevant information. This parameter could be used to sort or filter the information according to their relevance.

Comparability

Context information is normally provided by a multitude of different devices and sensors. Different measuring and coding systems, used by different manufacturer, result in a heterogeneous set of values describing the same entities (Krummenacher et al., 2007). Therefore, it would be necessary to provide some mechanism to compare values (e.g. the middleware could make a unit conversion). This is an important quality parameter because, for example, if we can compare the same context information, supplied by different sources, then we can choose the most accurate and precise source. We can also use it to measure the believability of a source, by comparing it with other trusted sources.

This parameter is essential to ensure the quality of context information, because we can only discriminate context information from different sources, based on its quality, if they are comparable.

Availability

Availability is "the extent to which information is accessible or easily and quickly retrievable" (Pipino et al., 2002). Sometimes, the accessibility of the context information is influenced by practical factors related to communications (e.g. time-of-the-day, network congestion, worldwide distribution of servers, denial-of-service-attacks, etc.) (Naumann, 2002). In these cases, these accessibility problems can be tackled by replicating or caching the information.

Bibliography

[Aarts et al. 2002] Aarts, E., Harwig, R., and Schuurmans, M. (2002). Ambient intelligence.The invisible future: the seamless integration of technology into everyday life, pages 235-250.

[Abid et al., 2009] Abid, Z., Chabridon, S., Conan, D., (2009). A framework for quality of context management. In: Rothermel, K., Fritsch, D., Blochinger, W., Drr, F. (Eds.), Quality of Context. Vol. 5786 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 120–131.

[Acceleo] Model to Text Language Acceleo. http://www.acceleo.org/pages/home/en

[Al-Bashayreh et al., 2013] Al-Bashayreh, M. G., Hashim, N. L., & Khorma, O. T. (2013). Context-Aware Mobile Patient Monitoring Frameworks: A Systematic Review and Research Agenda. Journal of Software, 8(7), 1604-1612.

[Alarcón et al., 2005] Alarcón, R., Guerrero, L. A., Ochoa, S., Pino, J. A. "Context in Collaborative Mobile Scenarios". Proc. of the Fifth International and Interdisciplinary Conference on Modelling and Using Context (Context 2005), Workshop 10: Context and Groupware. Paris, France. July 2005, Vol. 133

[Ardissono et al., 2007]. Ardissono, L., Furnari, R., Goy, A., Petrone, G., Segnan, M. "Context-Aware Workflow Management" Lecture Notes in Computer Science n. 4607, Web Engineering. (ICWE 2007), pp. 47-52, Springer Verlag.

[AtlanMod] AtlanMod Transformation Language (ATL). http://www.eclipse.org/atl/.

[Ayed et al., 2007] Ayed, D., Delanote, D., Berbers, Y., (2007). MDD approach for the development of context-aware applications. In: Kokinov, B., et al. (Eds.), CONTEXT 2007. Vol. 4635. LNAI. , pp. 15–28.

[Baldauf et al., 2007] Baldauf, M., Dustdar, S., Rosenberg, F., (2007). A survey on context-aware systems. International Journal Ad Hoc and Ubiquitous Computing 2 (4), 263–277.

[Ballou et al., 1985] Ballou, D.P., Pazer, H.L. (1985).Modeling data and process quality in multi-input, multi-output information systems. Manage. Sci. 31, 2 (1985), pp. 150–162.

[Bardram, 2005] Bardram, J.E., (2005). The Java Context Awareness Framework (JCAF) – a service infrastructure and programming framework for context-aware applications. In: Pervasive 2005. Vol. 3468. LNCS.

[Bolchini et al., 2007] Cristiana Bolchini, Carlo A. Curino, Elisa Quintarelli, Fabio A. Schreiber, and Letizia Tanca. 2007. A data-oriented survey of context models. SIGMOD Rec. 36, 4 (December 2007), 19-26.

[Bone et al., 2010] Bone, M., Cloutier, R. (2010): The current state of model based system engineering: results from the OMGTM SysML request for information 2009. In: Proceedings of the 8th Conference on Systems Engineering Research (CSER), March 2010. Hoboken, NJ.

[Borlund, 2003] Borlund, P. (2003): The concept of relevance in IR. Journal of the American Society for Information Science and Technology, 54(10), (Aug. 2003) 913-925

[Brambilla et al., 2012] Brambilla, M., Cabot, J., Wimmer, M. Model-Driven Software Engineering in Practice. Morgan & Claypool, 2012.

[Brezillon, 1999] Brezillon, P. (1999). Context in Artificial Intelligence: I. A survey of the literature. Computers and artificial intelligence, 18, 321-340.

[Brezillon et al., 2004] Brezillon, Borges, Pino, and Pomerol. (2004). "Context-Awareness in Group Work: Three Case Studies", IFIP Int. Conference on Decision Support Systems, Prato, Italy.

[Brown, 1996] Brown, P.J. (1996) 'The stick-e document: a framework for creating context-aware applications', Proceedings of the Electronic Publishing, Palo Alto, pp.259–272.

[Brown et al., 1997] Brown, P.J., Bovey, J.D. Chen, X., (1997). Context-Aware Applications: From the Laboratory to the Marketplace. IEEE Personal Communications. 4(5): p. 58-64.

[Bruneton et al., 2006] Bruneton, E., Coupaye, T., Leclercq, M., Quéma, V., Stefani, J.-B., 2006. The fractal component model and its support in java. Software-Practice and Experience 36 (11), 1257–1284.

[Buchholz et al., 2003] Buchholz, T., Küpper, A., Schiffer, M. (2003). Quality of context: What it is and why we need it. In Proc. of the 10th Workshop of the OpenView Univeristy Association (OVUA'03), Geneva, Switzerland, July 2003.

[Buchholz et al., 2004] S. Buchholz, T. Hamann, and G. Hübsch. Comprehensive structured context profiles (CSCP): Design and experiences. In Proc. 2nd IEEE Conf. on Pervasive Computing and Communications Workshops, pages 43–47, 2004

[Buchholz et al., 2004b] Buchholz, T., Krause, M., Linnhoff-Popien, C., Schiffers, M., 2004. CoCo: dynamic composition of context information. In: Proceedings of the First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'04), pp. 335–343.

[Campagne, 2014] Campagne, F. (2014). The MPS Language Workbench: Volume I (Vol. 1). Fabien Campagne.

[Carr et al., 2010] Carr, J. J., Brown, J. M., (2010). Introduction to biomedical equipment technology, 3rd Edition. Prentice Hall.

[Carvalho et al., 2012] Carvalho, L. P., da Silva, P. C. (2012). CCMF, Computational Context Modeling Framework–An Ontological Approach to Develop Context-Aware Web Applications. SEMANTICS IN ACTION–APPLICATIONS AND SCENARIOS, 63.

[Ceri et al., 2002] Ceri, S., Fraternali, P., Matera, M., July–August 2002. Conceptual modeling of dataintensive web applications. IEEE Internet Computing 6 (4), 20–30.

[Chabridon et al., 2013] Chabridon, S.,Conan, D.,Abid, Z.,Taconet, C. (2013). Building ubiquitous QoC-aware applications through model-driven software engineering. Science of Computer Programming. 78(10): 1912-1929 (2013). doi:10.1016/j.scico.2012.07.019

[Chen et al., 2000] Chen G, Kotz D. (2000). A survey of context-aware mobile computing research. Technical Report TR2000-381. Dartmouth College; November 2000.

[Chen et al., 2003] Chen, H., Finin, T., Joshi, A. Using OWL in a Pervasive Computing Broker. In Proceedings of Workshop on Ontologies in Open Agent Systems (AAMAS 2003) (2003).

[Chen, 2004] Chen, H. An Intelligent Broker Architecture for Pervasive Context-aware Systems. PhD Thesis University of Maryland, Baltimore Country, USA 2004.

[Chen et al., 2004] Chen,H., Finin, T., Joshi, A.,(2004) "An Ontology for Context-Aware Pervasive Computing Environments", Article, Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review, May 2004

[Chen et al., 2005] Chen, H., Finin, T. and Josh, A. 2005. The SOUPA Ontology for Pervasive Computing. In Ontologies for Agents: Theory and Experiences, pp. 233-258.

[Cheverst et al., 1998] Cheverst, K., Mitchell, K., Davies, N., 1998, Design of an Object Model for a Context Sensitive Tourist GUIDE. Proceedings of the International Workshop on Interactive Applications of Mobile Computing.(IMC 98), Rostock, Germany, pp 883-891.

[Clark et al., 2008] Clark, T., Evans, A., Sammut, P., Williams, J. Applied Metamodelling: A Foundation for Language Driven Development. Ceteva, 2008.

[Clear et al., 2007] Clear, A. K., Dobson, S., Nixon, P., (2007). An approach to dealing with uncertainty in context-aware pervasive systems. In: UK/IE IEEE SMC Cybernetic Systems Conference.

[Conan et al., 2007] Conan, D., Rouvoy, R., Seinturier, L., (2007). Scalable processing of context information with cosmos. In: Distributed Applications and Interoperable Systems. Springer, pp. 210–224.

[Coutaz et al., 2005] Coutaz, J., Crowley, J.L., Dobson, S., Garlan, D.. CONTEXT is KEY. Communications of the ACM, 48(3):49–53, 2005.

[Crepinsek et al., 2010] Crepinsek, M., Kosar, T., Mernik, M., Cervelle, J., Forax, R., Roussel, G., 2010. On automata and language based grammar metrics. Journal on Computer Science and Information Systems 7, 310–329.

[Czarnecki et al., 2006] Czarnecki, K., Helsen, S. Feature-based survey of model transformation approaches. IBM Systems Journal, 45(3):621–645, July 2006.

[Dey, 2000] Dey, A. K., (2000). Providing Architectural Support for Building Context-Aware Applications. Georgia Institute of Technology, Ph.D. Thesis.

[Dey et al., 2000] Dey, A.K., Abowd, G.D., 2000. Towards a better understanding of context and context awareness. Proceedings of the workshop on the What, Who, Where, When and how of Context Awareness, ACM Press, New York. 2000.

[Dey, 2001] Dey, A. K. (2001). Understanding and using context. Personal and ubiquitous computing, 5(1), 4-7.

[Duarte et al, 2015] Duarte, P. A. D. S., Barreto, F. M., Gomes, F. A. D. A., Carvalho, W. V. D., & Trinta, F. A. M. (2015, July). CRITiCAL: A Configuration Tool for Context Aware and mobiLe Applications. In Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual (Vol. 2, pp. 159-168). IEEE.

[EBNF, 1996] Extended BNF ISO/IEC 14977:1996 international standard. http://www.iso.org/iso/catalogue_detail?csnumber=26153

[Ejigu et al., 2007] D. Ejigu, M. Scuturici, L. Brunie (2007), ‛CoCA: A Collaborative Context-Aware Service Platform for Pervasive Computing′, In Proceedings of the Fourth International Conference on Information Technology. ITNG '07.

[EMFText] EMFText Concrete syntax mapper. http://www.emftext.org/

[EMOF, 2004] The Object Management Group: "Meta Object Facility (MOF) 2.0 Core Final Adopted Specification"; (Mar. 2004) http://www.omg.org/cgi-bin/doc?ptc/03-10-04.

[Eppler et al., 2002] Eppler, M. J., Muenzenmayer, P., November (2002). Measuring information quality in the web context: A survey of stateof-the-art instruments and an application methodology. In: International Conference on Information Quality (ICIQ02), MIT. pp. 187–196.

[Eysholdt et al., 2010] Eysholdt, M., Behrens, H. (2010, October). Xtext: implement your language faster than the quick and dirty way. In Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion (pp. 307-309). ACM.

[Filho et al.,2010] Filho, J.B., Miron, A.D., Satoh, I. (2010). Modeling and measuring quality of context information in pervasive environments. In: The IEEE International Conference on Advanced Information Networking and Applications, pp. 690-697.

[Fischer, 2012] Fischer, G.. Context-aware systems - the 'right' information, at the 'right' time, in the 'right' place, in the 'right' way, to the 'right' person. In AVI'12. ACM, 2012.

[Fisher et al., 2001] Fisher, C. W., Kingma, B. R. (2001). Criticality of data quality as exemplified in two disasters. Information & Management, 39(2), 109-116.

[Fowler, 2010] Fowler, M. (2010). Domain-specific languages. Pearson Education.

[García-Molina et al., 2013] García-Molina, J., García, F.O., Pelechano, V. , Vallecillo, A., Vara, J.M., Vicente-Chicote, C., 2013. Desarrollo de Software Dirigido por Modelos: Conceptos, Métodos y Herramientas. RA-MA editorial. ISBN: 978-8499642154

[Göker et al, 2002] Göker, A., Myrhaug, H.I.: User context and personalisation. In: Workshop proceedings for the 6th European Conference on Case Based Reasoning. (2002)

[Gómez-Romero et al., 2011] Gómez-Romero, J., Bobillo, F., Delgado, M. (2011, August). Context representation and reasoning with formal ontologies.

In Workshops at the Twenty-Fifth AAAI Conference on Artificial Intelligence.

[Gray et al., 2001] Gray, P.D., Salber, D. "Modelling and using sensed context information in the design of interactive applications". In Proceedings of the 8th IFIP International Conference on Engineering for Human-Computer Interaction (EHCI 2001), Reed Little, M., Nigay, L. (eds) Engineering for Human-Computer Interaction. LNCS n. 2254, 317-336. Berlin Springer-Verlag, 2001

[Greenfield, 2006] Greenfield, A. (2006). Everyware: the dawning age of ubiquitous computing. New Riders. pp. 11–12. ISBN 0-321-38401-6.

[Gu et al., 2005] Gu, T., Pung, H., Zhang, D. (2005) A service-oriented middleware for building context-aware services. Journal of Network and Computer Applications, Volume 28, Issue 1, January 2005, Pages 1-18

[Halpin, 2010] Halpin, T. A. Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design. Morgan Kaufman Publishers, San Francisco, 2010

[Held et al., 2002] Held, A., Buchholz, S., Schill, A., (2002). Modeling of context information for pervasive computing applications. In: Proc. of the 6th World Multiconference on Systemics, Cybernetics and Informatics.

[Helfert et al., 2013] Helfert, M., Walshe, M. R., Gurrin, C. (2013). The Impact of Information Quality on Quality of Life: An Information Quality Oriented Framework. IEICE TRANSACTIONS on Communications Vol.E96-B No.2 pp.404-409

[Henricksen et al., 2002] Henricksen, K., Indulska, J., Rakotonirainy, A. (2002). Modeling context information in pervasive computing systems. In Pervasive Computing (pp. 167-180). Springer Berlin Heidelberg.

[Henricksen et al., 2004] Karen Henricksen and Jadwiga Indulska. (2004). Modelling and using imperfect context information. In PERCOMW′04: Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops, page 33. IEEE Computer Society.

[Henricksen et al., 2004b] Henricksen, K. , Indulska, J. "A software engineering framework for context-aware pervasive computing". In Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04, Orlando, Florida, March 14-17, 2004) pp. 77-86.

[Hermans et al., 2009] Hermans, F., Pinzger, M., & Van Deursen, A. (2009). Domain-specific languages in practice: A user study on the success factors. In Model driven engineering languages and systems (pp. 423-437). Springer Berlin Heidelberg.

[Hisazumi et al., 2003] Hisazumi, K., Nakanishi, T., Kitasuka, T., Fukuda, A., et al., 2003. Campus: A context-aware middleware. In: The 2nd CREST Workshop on Advanced Computing and Communicating Techniques for Wearable Information Playing.

[Hofer et al., 2002] Hofer, T., Schwinger, W., Pichler, M., Leonhartsberger, G., and Altmann, J. (2002). Context-awareness on mobile devices – the hydrogen approach. In Proceedings of the 36th Annual Hawaii International Conference on System Sciences, pages 292–302.

[Hong et al., 2004] Hong, J.I., Landay, J.A (2004) 'An architecture for privacy-sensitive ubiquitous computing', In Proceedings of: 2nd International Conference on Mobile Systems, Applications, and Services (MobiSys), Boston.

[Hong et al., 2009] Hong, J., Suh, E., Kim, J., Kim, S., 2009. Context-aware systems for proactive personalized service based on context history. Expert Systems with Applications: An International Journal 36 (4), 7448–7457.

[Hoyos et al., 2011] Hoyos, J. R., Preuveneers, D., García-Molina, J. J., Berbers, Y., (2011). A dsl for context quality modeling in context-aware applications. In: Novais, P., Preuveneers, D., Corchado, J. (Eds.), Ambient Intelligence - Software and Applications. Vol. 92 of Advances in Intelligent and Soft Computing. Springer Berlin Heidelberg, pp. 41–49

[Hoyos et al., 2014] Hoyos, J.R., GarciaMolina, J., Botia, J.A. Modelado de calidad de contexto con MLContext. XIX Jornadas de Ingeniería del Software y Bases de Datos (JISBD'2014). Cádiz (España), 2014. ISBN: 978-84-697-1152-1

[Hutchinson et al., 2011] Hutchinson, J., Whittle, J., Rouncefield, M., Kristofferson, S. (2011): Empirical assessment of MDE in industry. In:

Proceedings 33rd International Conference on Software Engineering (ICSE′ 11), pp. 471–480. ACM, New York.

[ITU-T, 2008] ITU-T, "E.800: Terms and definitions related to quality of service and network performance including dependability," 2008. Available at http://www.itu.int/rec/T-REC-E.800-200809-I

[Izquierdo et al., 2008] Izquierdo, J. L. C., Cuadrado, J. S., Molina, J. G. (2008). Gra2MoL: A domain specific transformation language for bridging grammarware to modelware in software modernization. In Workshop on Model-Driven Software Evolution.

[Johnson et al., 1981] Johnson, J.R.; Leitch, R.A.; and Neter, J. Characteristics of errors in accounts receivable and inventory audits. Accounting Review, 56, 2 (1981), 270-293

[Juran, 1974] Juran, J., (1974). The Quality Control Handbook. McGraw-Hill.

[Kagolovsky et al., 2001] Kagolovsky Y, Mohr JR.(2001): A new approach to the concept of relevance in information retrieval (IR). In: Patel V, Rogers R and Haux R (editors). Proceedings of the 10th World Congress on Medical Informatics (Medinfo 2001). Amsterdam, The Netherlands: IOS Press, 2001 Sep;10(Pt 1):348-52

[Kahn et al., 2002] Kahn, B. K., Strong, D. M., Wang, R. Y., (2002). Information quality benchmarks: product and service performance. Communications of the ACM 45 (4), 184–192.

[Kan, 2002] Kan, S.H. Metrics and Models in Software Quality Engineering (2nd ed.). (528 pages). Addison-Wesley. ISBN 0-201-72915-6

[Kelly et al., 2008] Kelly, S., Tolvanen, J.P.: Domain Specific Languages, John Wiley, 2008.

[Kelly et al., 2009] Kelly, S. Pohjonen, R. (2009). Worst Practices for Domain-Specic Modeling. IEEE Software, 26:4, 2009.

[Khattak et al., 2009] Khattak, A. M., Latif, K., Lee, S., & Lee, Y. K. (2009). Ontology evolution: a survey and future challenges. In U-and E-service, science and technology (pp. 68-75). Springer Berlin Heidelberg.

[Kiama] Kiama project. https://bitbucket.org/inkytonik/kiama

[Kim et al., 2006] Kim, Y., Lee, K., (2006). A quality measurement method of context information in ubiquitous environments. In: Hybrid

Information Technology, 2006. ICHIT′06. International Conference on. Vol. 2. IEEE, pp. 576–581.

[Klatt, 2008] B. Klatt, "Xpand: A Closer Look at the model2text Tranformation Language" 12th European Conference on Software Maintenance and Reengineering, (2008).

[Kleppe et al., 2008] Kleppe, A.; Warmer, J., Bast, W.; MDA Explained, Addison-Wesley, 2003.

[Knight et al., 2005] Knight, S., Burn, J. (2005). Developing a Framework for Assessing Information Quality on the World Wide Web. Informing Science Journal, 8:160–172, 2005

[Koç et al., 2014] Koç, H., Hennig, E., Jastram, S., & Starke, C. (2014, January). State of the Art in Context Modelling–A Systematic Literature Review. In Advanced Information Systems Engineering Workshops (pp. 53-64). Springer International Publishing.

[Kofod-Petersen et al., 2005] Kofod-Petersen, A., Mikalsen, M. (2005, June). An architecture supporting implementation of context-aware services. In Workshop on Context Awareness for Proactive Systems (CAPS 2005), Helsinki, Finland, HIIT Publications (pp. 31-42).

[Kolovos et al, 2006] Kolovos, D., Paige, R., Kelly, T. Polack, F. Requirements for Domain-Specific Languages. In Domain-Specific Program Development workshop, 2006.

[Kolovos et al., 2008] Kolovos, D. S., Paige, R. F., & Polack, F. A. (2008). The epsilon transformation language. In Theory and practice of model transformations (pp. 46-60). Springer Berlin Heidelberg.

[Korpipää et al., 2003] Korpipää, P., Jani, M., Kela, J., & Malm, E. J. (2003). Managing context information in mobile devices. IEEE pervasive computing, (3), 42-51.

[Krause et al., 2005] Krause, M., Hochstatter, I.: Challenges in modeling and using quality of context (qoc). In: Magedanz, T., Karmouch, A., Pierre, S., Venieris, I.S. (eds.) MATA 2005. LNCS, vol. 3744, pp. 324–333. Springer, Heidelberg (2005)

[Krummenacher et al., 2007] Krummenacher, R., Strang, T. (2007). Ontology-Based Context Modeling. In Workshop on Context-Aware Proactive Systems, 2007

[Lachica et al., 2008] Lachica, R., Karabeg, D., Rudan, S., (2008). Quality, relevance and importance in information retrieval with fuzzy semantic networks. Proc. TMRA.

[Laudon, 1986] Laudon, K.C. Data quality and due process in large interorganizational record systems. Communications of the ACM, 29, 1 (1986), 4-11.

[Leclercq et al., 2007] Leclercq, M., Özcan, A. E., Quema, V., Stefani, J.-B., 2007. Supporting heterogeneous architecture descriptions in an extensible toolset. In: Software Engineering, 2007. ICSE 2007. 29th International Conference on. IEEE, pp. 209–219.

[Lee et al., 2002] Lee, Y. W., Strong, D. M., Kahn, B. K., Wang, R. Y., 2002. Aimq: a methodology for information quality assessment. Information & management 40 (2), 133–146.

[Lee et al., 2011] Lee, S., Chang, J., Lee, S.-g. (2011). Survey and Trend Analysis of Context-Aware Systems. Information-An International Interdisciplinary Journal. Vol. 14(2). pp. 527-548.

[Lenat, 1995] Lenat, D. B. 1995 CYC: A large-scale investment in knowledge infrastructure. Communications of the ACM 38(11), 33–38.

[Lillrank, 2003] Lillrank, P. (2003). The quality of information. International Journal of Quality & Reliability Management, 20(6), 691-703.

[Machara et al., 2013] Machara, S., Chabridon, S., & Taconet, C. (2013, December). Trust-based Context Contract Models for the Internet of Things. In Ubiquitous Intelligence and Computing, 2013 IEEE 10th International Conference on and 10th International Conference on Autonomic and Trusted Computing (UIC/ATC) (pp. 557-562). IEEE.

[Maedche et al, 2003] Maedche, A., Motik, B., Stojanovic, L., Studer, R., Volz, R. (2003). Ontologies for Enterprise Knowledge Management, IEEE Intelligent Systems, vol. 18, no. 2, pp. 26-33.

[Manzoor et al., 2011] Manzoor, A., Truong, H.L., Dustdar, S. (2011). Quality of context: models and applications for context-aware systems in pervasive environments, in: Web and Mobile Information Services, The Knowledge Engineering Review (2011) (special issue).

[Marie et al., 2014] Marie, P., Desprats, T., Chabridon, S., Sibilla, M. (2014). The QoCIM Framework: Concepts and Tools for Quality of Context Management. In Context in Computing (pp. 155-172). Springer New York.

[McCarthy , 1993] McCarthy, J. "Notes on formalizing contexts". In Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (San Mateo, California, 1993), R. Bajcsy, Ed., Morgan Kaufmann, pp. 555–560.

[McFadden et al., 2004] McFadden, T., Henricksen, K., Indulska, J. (2004). Automating context-aware application development. In Jadwiga Indulska & David De Roure (eds) 1st International Workshop on Advanced Context Modelling, Reasoning and Management, in conjunction with the Sixth International Conference on Ubiquitous Computing, Tokyo, Japan, 2004. (pp. 90-95)

[McKeever et al., 2009] McKeever, S., Ye, J., Coyle, L., Dobson, S. (2009). A Context Quality Model to Support Transparent Reasoning with Uncertain Context. QuaCon, 9, 65-75.

[Mens, 2013] Mens, T. (2013) Model Transformation: A Survey of the State of the Art, in Model-Driven Engineering for Distributed Real-Time Systems: MARTE Modeling, Model Transformations and their Usages (eds J.-P. Babau, M. Blay-Fornarino, J. Champeau, S. Robert and A. Sabetta), John Wiley & Sons, Inc., Hoboken, NJ, USA. doi: 10.1002/9781118558096.ch1

[Mernik et al., 2005] Mernik, M., Heering, J., Sloane, A.M. When and How to Develop Domain-Specific Languages. ACM Computer Survey, 37(4):316-344, 2005.

[MOF, 2006] MOF. The Meta-Object Facility. Specification documents: http://www.omg.org/spec/MOF/2.0/, 2006.

[MOFScript] MOFScript (Meta-Object Facility) Model-to-Text Transformation Tool. http://www.eclipse.org/gmt/mofscript

[Moran et al., 2001] Moran, T.P., Dourish, P. (2001) Introduction to This Special Issue on Context-Aware Computing, Human–Computer Interaction, 16:2-4, 87-95

[Mühlhäuser, 2008] Mühlhäuser, M. (Ed.). (2008). Handbook of research on ubiquitous computing technology for real time enterprises. IGI Global.

[Naumann, 2002] Naumann, F., (2002). Quality-driven query answering for integrated information systems. Vol. 2261. Springer Science & Business Media.

[Navarro et al., 2009] Navarro, S., Alcarria, R., Botia, J.A., Platas, S., Robles, T., 2009, April. CARDEA: service platform for monitoring patients and medicines based on SIP-OSGi and RFID technologies in hospital environment. In: Primer simposio OpenHealth, Spain.

[Nieto et al., 2006] Nieto, I., Botia, J.A., Gómez-Skarmeta, A. F. (2006). Information and Hybrid Architecture Model of the OCP Contextual Information Management System Journal of Universal Computer Science, vol. 12, no. 3 (2006), 357-366

[Oldevik et al., 2005] Oldevik, J., Neple, T., Grønmo, R., Aagedal, J., & Berre, A. J. (2005, January). Toward standardised model to text transformations. In Model Driven Architecture–Foundations and Applications (pp. 239-253). Springer Berlin Heidelberg.

[Oliveira et al., 2009] Oliveira, N., Pereira, M.J.V., Henriques, P.R., Cruz, D., 2009. Domain specific languages: a theoretical survey. In: 3rd Compilers, Programming Languages, Related Technologies and Applications (CoRTA′ 2009).

[OMG, 2000] Object Management Group (OMG), 2000. Unified Modeling Language Specification. http://www.omg.org/spec/UML/

[OSGI, 2012] Open Services Gateway Initiative. OSGi Alliance Home Page.

http://www.osgi.org/Main/HomePage

[OWL, 2004] World Wide Web Consortium (W3C). OWL Web Ontology Language Guide. http://www.w3.org/TR/owl-guide/

[Parr, 2013] Parr, T. The Definitive ANTLR 4 Reference. Pragmatic Bookshelf, 2013. ISBN: 978-1-93435-699-9

[Peffers et al., 2008] Peffers K., Tuunanen T., Rothenberger MA., Chatterjee S. (2008). A design science research methodology for information systems research. Journal of Management Information Systems 2008;24(3):45–77.

[Peinado et al., 2015] Peinado, S., Ortiz, G., & Dodero, J. M. (2015). A metamodel and taxonomy to facilitate context-aware service adaptation. Computers & Electrical Engineering, vol 44, 262–279

[Pipino et al., 2002] Pipino, L. L., Lee, Y. W., Wang, R. Y. (2002). Data quality assessment. Communications of the ACM, 45(4), 211-218.

[Pires et al., 2005] Pires, L. F., Sinderen, M. van, Munthe-Kaas, E., Pokraev, S., Hutschemaekers, M. and Plas, D. J., ″Techniques for describing and manipulating context information″, Lucent Technologies, Freeband A-Muse Project, Technical report D3.5v2.0, October, 2005

[Power et al., 2004] Power, J., Malloy, J., 2004. A metrics suite for grammar-based software. Journal of Software Maintenance and Evolution: Research and Practice 16, 405–426.

[Preuveneers et al., 2005] Preuveneers, D., Berbers, Y., (2005). Semantic and syntactic modeling of componentbased services for context-aware pervasive systems using owl-s. In: MCMP-05. First International Workshop on Managing Context Information in Mobile and Pervasive Environments, pp. 30–39.

[Preuveneers et al., 2007] Preuveneers, D., Berbers, Y., (2007). Architectural backpropagation support for managing ambiguous context in smart environments. In: Universal Access in Human-Computer Interaction. Ambient Interaction. Springer, pp. 178–187.

[Preuveneers et al., 2014] Preuveneers, D., Berbers, Y., June 2014. Samurai: A streaming multi-tenant context-management architecture for intelligent and scalable internet of things applications. In: Intelligent Environments (IE), 2014 International Conference on. pp. 226–233.

[Quintalleri et al., 2015] Quintarelli, E., Rabosio, E., Tanca, L. (2015). A principled approach to context schema evolution in a data management perspective. Information Systems, 49, 65-101.

[QVT20] Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT). http://www.omg.org/spec/QVT/.

[Ranganathan et al., 2004] Ranganathan, A., Al-Muhtadi, J., Campbell, R. H. (2004). Reasoning about uncertain contexts in pervasive computing environments. IEEE Pervasive Computing, (2), 62-70.

[Ranganathan,et al., 2004b] Ranganathan, A., Al-Muhtadi, J., Chetan, S., Campbell, R., Mickunas, M., 2004. Middlewhere: A middleware for location awareness in ubiquitous computing applications. In: Jacobsen, H.-A. (Ed.), Middleware 2004. Vol. 3231 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 397–416.

[Rittenbruch, 2002] Rittenbruch. (2002). "ATMOSPHERE: A Framework for Contextual Awareness", International Journal of Human-Computer Interaction 14(2), 159-180.

[Riva, 2006] Riva, O. (2006). Contory: A Middleware for the Provisioning of Context Information on Smart Phones. Middleware 2006, LNCS 4290, pp. 219–239. Springer Berlin Heidelberg

[Ryan, 1999] Ryan, N. "ConteXtML: Exchanging Contextual Information between a Mobile Client and the FieldNote Server", August 1999. Available at
http://www.cs.kent.ac.uk/projects/mobicomp/fnc/ConteXtML.html

[Ryan et al., 1997] Ryan, N., J. Pascoe, J., Morse, D. (1997). Enhanced Reality Fieldwork: the Context Aware Archaeological Assistant, in: Dingwall, L., S. Exon, V. Gaffney, S. Laflin and M. van Leusen (eds.), Archaeology in the Age of the Internet. CAA97. Computer Applications and Quantitative Methods in Archaeology. Proceedings of the 25th Anniversary Conference, University of Birmingham, (BAR International Series 750). Archaeopress, Oxford, pp. 269-274.

[Sadalage et al., 2012] Sadalage, P.J., Fowler, M. NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley, 2012

[Salber et al, 1999] Salber, D., Dey, A.K., Abowd, G.D. (1999). The Context Toolkit: Aiding the Development of Context-Enabled Applications. In: Proceedings of Conf. Human Factors in Computing Systems, CHI′99. pp 434-441

[Sánchez et al., 2006] Sánchez, J., García-Molina, J., Menárguez, M. RubyTL: A practical, extensible transformation language. In 2nd European

Conference on Model-Driven Architecture, volume 4066 of LNCS, pages 158–172. Springer, 2006.

[Schilit et al., 1994] Schilit, B.N. and Theimer, M.M. (1994). "Disseminating Active Map Information to Mobile Hosts". IEEE Network 8 (5): 22–32. doi:10.1109/65.313011

[Schilit et al., 1994b] Schilit, B.N., Adams, N.L., Want, R., (1994). Context-aware computing applications. In: IEEE Workshop on Mobile Computing Systems and Applications. IEEE Computer Society, pp. 85–90. Santa Cruz, CA, US.

[Schmidt, 2005] Schmidt, A., 2005. The knowledge maturing process as a unifying concept for Elearning and knowledge management. In: Proceedings of the 5th International Conference on Knowledge Management (I-KNOW 2005).

[Selic, 2007] Selic, B.,(2007), May. A systematic approach to domain-specific language design using UML. In: Tenth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2007). IEEE Computer Society, pp. 2–9.

[Selic, 2012] Selic, B. (2012). What will it take? A view on adoption of model-based methods in practice. Software & Systems Modeling, 11(4), 513-526.

[Serral et al., 2007] Serral, E., Valderas, P., Muñoz, J., Pelechano, V.: Towards a Model Driven Development of Context-aware Systems for AmI Environments. AmI.d, Nize, France (2007)

[Serral et al., 2008] Serral, E., Valderas, P., Pelechano, V., (2008). A model driven development method for developing context-aware pervasive systems. In: UIC 2008. Vol. 5061. LNCS, pp. 662–676.

[Sheng et al., 2005] Sheng, Q.Z., Benatallah, B., (2005). ContextUML: a UML-based modeling language for model-driven development of context-aware web services. In: 2005 International Conference on Mobile Business (ICMB 2005), 11–13 July 2005, Sydney, Australia. IEEE Computer Society, pp. 206–212.

[Sheng et al., 2009] Sheng, Q. Z., Pohlenz, S., Yu, J., Wong, H. S., Ngu, A. H., Maamar, Z. (2009, May). ContextServ: A platform for rapid and flexible development of context-aware Web services. In Proceedings of the

31st International Conference on Software Engineering (pp. 619-622). IEEE Computer Society.

[Sidi et al., 2012] Sidi, F., Shariat Panahy, P. H., Affendey, L. S., Jabar, M., Ibrahim, H., & Mustapha, A. (2012, March). Data quality: A survey of data quality dimensions. In Information Retrieval & Knowledge Management (CAMP), 2012 International Conference on (pp. 300-304). IEEE.

[Sindico et al., 2009] Sindico, A., Grassi, V., (2009). Model driven development of context aware software systems. In: COP'09: International Workshop on Context-Oriented Programming, Genova, Italy. ACM, pp. 1–5.

[Stahl et al., 2006] Stahl, T., Volter, M.; Model-driven Software Development. John Wiley, 2006.

[Steimberg et al., 2008] Steinberg, D., Budinsky, F., Paternostro, M., Merks, E. EMF: Eclipse Modeling Framework. Addison-Wesley Professional, 2008.

[Strang et al., 2004] Strang, T., Linnhoff-Popien, C. (2004). A Context Modeling Survey. In In: Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004-The Sixth International Conference on Ubiquitous Computing, Nottingham/England.

[Tesoriero et al., 2010] Tesoriero, R., Gallud, J. A., Lozano, M. D., & Penichet, V. M. R. (2010). CAUCE: Model-driven Development of Context-aware Applications for Ubiquitous Computing Environments. J. UCS, 16(15), 2111-2138.

[Topcased, 2005] Toolkit in Open Source for Critical Applications & Systems Development (Topcased) (2005).
https://marketplace.eclipse.org/content/topcased

[Vaishnavi et al, 2013] Vaishnavi, V. and Kuechler, W., (2013). Design science research in information systems.
http://desrist.org/desrist/content/design-science-research-in-information-systems.pdf

[Vanathi et al., 2010] Vanathi, B., Uthariaraj, V. R. (2010, January). Context Representation and Management in a Pervasive Environment. In

Information and Communication Technologies (pp. 543-548). Springer Berlin Heidelberg.

[Vildjiounaite et al., 2007] Vildjiounaite, E., Kallio, S., 2007. A layered approach to context-dependent user modelling.In: Advances in Information Retrieval ECIR 2007. Vol. 4425. LNCS, pp. 749–752.

[Völter, 2009] Völter, M., (2009). MD* best practices. Journal of Object Technology 8 (6), 79–102. (updated March 2011)

[Völter et al., 2013] Völter, M., Benz, S., Dietrich, C., Engelmann, B., Helander, M., Kats, L.C.L., Visser, E., Wachsmuth, G., 2013. DSL Engineering - Designing, Implementing and Using Domain-Specific Languages. dslbook.org

[Wand et al., 1996] Wand, Y., Wang, R. Y., (1996). Anchoring data quality dimensions in ontological foundations. Communications of the ACM 39 (11), 86–95.

[Wang et al., 1996] Wang, R., Strong, D. (1996). Beyond accuracy: what data quality means to data consumers, Journal of Management Information Systems 12 (4) (1996) 5–33.

[Want et al., 1992] Want, R., Hopper, A., Falcao, V. and Gibbons, J. (1992) 'The active badge location system', ACM Transactions on Information Systems, Vol. 10, No. 1, pp.91–102

[Wei et al., 2007] Wei, E. J. Y., Chan, A. T. S. (2007). Towards context-awareness in ubiquitous computing. In Proceedings of the 2007 international conference on Embedded and ubiquitous computing (pp. 706–717). Berlin, Heidelberg: Springer-Verlag.

[Weiser, 1988] Mark Weiser personal web page.
http://www.ubiq.com/hypertext/weiser/UbiHome.html

[Weiser, 1991] Weiser, M. (1991). The computer for the twenty-first century. Scientific American. Newsletter. ACM SIGMOBILE Mobile Computing and Communications Review - Special issue dedicated to Mark Weiser. Volume 3 Issue 3, July 1999. Pages 3-11. ACM New York, NY, USA

[Weisser et al., 1997] Weiser, M., & Brown, J. S. (1997). The coming age of calm technology. In Beyond calculation (pp. 75-85). Springer New York.

[Whittle et al., 2014] Whittle, J., Hutchinson, J., and Rouncefield, M. (2014). The State of Practice in Model-Driven Engineering. Software, IEEE, 31(3):79–85.

[Wooldridge et al., 1995] Wooldridge, M. J., Jennings, N. R. Intelligent agents: Theory and practice. Knowledge Engineering Review, 10(2):115–152, June 1995.

[Zhang et al., 2014] Zhang, P., Zhang, L., Fan, Z., & Qiu, X. (2014). Knowledge Based Modeling Method of Artificial Society Oriented to Emergency Management. In Life System Modeling and Simulation (pp. 278-287). Springer Berlin Heidelberg.

[张 鹏 et al, 2015] 张鹏, 陈彬, 孟荣清, 张烙兵, & 邱晓刚. (2015). 面向应急管理计算实验的模型构建和模型管理. 国防科技大学学报, 37(3), 173-178.