EXPLOITING METADATA, ONTOLOGIES AND SEMANTICS TO DESIGN/ENHANCE NEW END-USER EXPERIENCES FOR ADAPTIVE PERVERSIVE COMPUTING ENVIRONMENTS

Ahmet SOYLU

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Science

May 2012
EXPLOITING METADATA, ONTOLOGIES AND SEMANTICS TO DESIGN/ENHANCE NEW END-USER EXPERIENCES FOR ADAPTIVE PERVERSIVE COMPUTING ENVIRONMENTS

Ahmet SOYLU

Examination Committee:
Prof. dr. Paul Igodt (chair)
Prof. dr. Patrick De Causmaecker (promotor)
Prof. dr. ir. Erik Duval (co-supervisor)
Prof. dr. Piet Desmet (co-supervisor)
Prof. dr. ir. Yolande Berbers
Prof. dr. Miguel-Angel Sicilia (University of Alcalá)
dr. Katrien Verbert
Fridolin Wild (The Open University)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Science

May 2012
Abstract

Adaptive Systems and Pervasive Computing change the face of computing and redefine the way people interact with the technology. Pioneers pursue a vision that technology is seamlessly situated in people’s life and adapts itself to the characteristics, requirements, and needs of the users and the environment without any distraction at the user side. Adaptive Systems research mostly focuses on individual applications that can alter their interface, behavior, presentation etc. mainly with respect to the user characteristics and needs. The pervasive computing vision considers adaptivity as a relation between the computing setting and the context rather than a one-to-one relation between user and application. Therefore, it enlarges the source of adaptation from user characteristics to a broader notion, which is context. In this respect, emerging context-aware systems try to utilize the characteristics, requirements etc. of entities relevant to the computing setting while including the user as a core entity.

Context is an open concept and collected contextual information is often imperfect. Therefore, the development and management of large scale pervasive and adaptive systems and adaptation logic is complex and challenging. Several researchers tried to remedy this situation by providing middleware support, approaches, and methods for dealing with imperfectness, context modeling, management, reasoning etc. However, due to the openness of contextual information, on the one hand, it is almost impossible to enumerate every possible scenario and to define or mine adaptation rules. On the other hand, absolute machine control is not always desirable considering the intellectual characteristics of the end-users. The aforementioned criticism suggests that efficient software development approaches and means to enable end-users to reflect on their own state of affairs are required. In this thesis, we address these issues at an individual application level and at a collective level. The former deals with development and end-user interaction issues on the basis of individual applications providing adaptive experiences while the concern of the latter is on the basis of distributed applications serving end-users in concert. For each level, we provide a number of conceptual and practical contributions, mainly applied to the e-learning domain.

Regarding the individual application level, we utilize an approach based on high level abstractions, particularly ontologies, aiming at facilitating the development and management of adaptive and pervasive systems. Ontologies are used for the acquisition of domain knowledge and semantics at the first stage. Afterwards, the goal is to use the resulting ontology for dynamic adaptations, end-user awareness, intelligibility, software self-expressiveness, user control, and automated software development with a Model Driven Development perspective. Our contribution is mainly conceptual at this level. We first review the main notions and characteristics of Pervasive Computing and Adaptive Systems along end-user considerations. We criticize the pervasive computing vision with a user-centric perspective and elaborate on the practical body of existing literature, intersecting Knowledge Representation,
Logic, and the Semantic Web, to arrive at a uniform development approach meeting the aforementioned concerns.

Regarding the collective level, we focus on personal and pervasive environments and investigate how high level abstractions and semantics, varying from generic vocabularies and metadata approaches to ontologies, can be exploited for the creation of such environments and to enrich and augment the end-user experience. Our contribution is built on the conceptual and practical approach that we derived earlier. We envision encapsulating digital and physical entities having digital presence in the form of widgets and enabling the creation of web-based personal environments through widget-based user interface mashups. For this purpose, we first address the widgetization of existing applications, in a broader sense in terms of ubiquitous web navigation, through harvesting semantic in-content annotations from application interfaces. An ontology-driven development approach allows automated annotation and generation of user interfaces. We introduce specifications and mechanisms for annotation, extraction, and presentation of embedded data. We introduce a set of heuristics to exploit domain knowledge and ontology metadata, with ontological reasoning support, for generating user-friendly navigation experiences. Thereafter, we introduce an open and standard widget platform, an interoperability framework, and methods for manual and automated widget orchestration. We introduce public widget interfaces and employ the semantic web technologies, particularly embedded semantics and ontologies, to address data and application interoperability challenges. We build an end-user data mobility facility on top of the proposed interoperability framework for user-driven manual orchestration. We propose a method for mining user behavioral patterns from the user logs. The method is based on the adoption of workflow mining techniques, for extracting topology, and multi-label classification techniques based on a label combination approach, for learning the routing criteria. We exploit harvested patterns for demand-driven automated widget orchestration.

We compare our approaches and methods with a broad interdisciplinary literature. We provide prototypes for each practical contribution and conduct end-user experiments and usability assessments to prove the computational feasibility and usability of the proposed approaches and methods.
Samenvatting

Adaptieve systemen en pervasive computing veranderen de aanblik van computersystemen en herdefiniëren de manier waarop mensen met de technologie omgaan. Volgens de inzichten van de pioniers is technologie naadloos in het leven van de mensen ingebed en past ze zich aan de eigenschappen, verwachtingen en noden van de gebruikers aan zonder deze gebruiker hierbij af te leiden. Onderzoek naar adaptieve systemen concentreert zich vooral op individuele toepassingen die in staat zijn om hun interface, gedrag, presentatie enz. te veranderen, hierbij meestal rekening houdend met de eigenschappen en de noden van de gebruiker. De visie van pervasive computing beschouwt adaptiviteit als een relatie tussen computing en context, veelal dan als een één op één relatie tussen gebruiker en toepassing. De bron van de adaptiviteit wordt daarom uitgebreid van de gebruikerseigenschappen naar het bredere begrip context. In dit verband pogen de opkomende ‘context-aware’ systemen om eigenschappen, vereisten e.d. van de voor computing relevante entiteiten te gebruiken, met de gebruiker als een centrale entiteit.

Context is een open concept en verzamelde contextuele informatie is vaak niet perfect. De ontwikkeling en het beheer van grootschalige pervasive en adaptieve systemen en de adaptie-logica is dan ook complex en uitdagend. Verschillende onderzoekers benaderen de problematiek via middleware ondersteuning en ze ontwikkelden methodes voor de behandeling van imperfecties, context modellering, redeneren enz. Het is evenwel vanwege de openheid en de contextuele informatie bijna onmogelijk om voor elk mogelijk scenario adaptatierregels te definiëren of via mining te bepalen. Anderzijds is volledige controle door de machine niet altijd wenselijk, gezien de intellectuele mogelijkheden van de eindgebruikers.

Deze kritiek suggereert dat efficiënte benaderingen voor software ontwikkeling nodig zijn, evenals manieren die eindgebruikers toe laten na te denken over hun eigen situatie. In deze thesis behandelen we deze onderwerpen zowel op het niveau van de individuele toepassing als collectief. Het eerste betreft de ontwikkeling en eindgebruiker-interactie met individuele toepassingen voor adaptieve ervaringen. Het laatste betreft een verzameling van gedistribueerde toepassingen die de eindgebruiker gezamenlijk bedienen. Voor elk niveau leveren we een aantal conceptuele en praktische bijdragen, voornamelijk toegepast op het e-learning domein.

Wat het individuele toepassingsgebied betreft gebruiken we een benadering gebaseerd op hoog niveau abstracties, in het bijzonder otologieën, gericht op het faciliteren van ontwikkeling en beheer van adaptive pervasive systems. Ontologieën worden gebruikt voor het verwerven van domeinkennis en semantiek in de eerste stap. Vervolgens wenden we de resulterende ontologie aan bij het at run-time redeneren over dynamische aanpassingen, observatie van de eindgebruiker, begrijpelijkheid, expressiviteit en controle van de gebruiker. We genereren en hergenereren de toepassingscode automatisch via dezelfde ontologie door Model Driven Development. Op dit niveau is onze bijdrage vooral conceptueel. We herbekijken eerst de
belangrijkste begrippen en concepten van pervasieve computing en adaptive systems vanuit het perspectief van de eindgebruiker. We herdefiniëren de visie van pervasive computing met een perspectief waarin de gebruiker centraal staat en werken verder op een praktisch geheel van bestaand werk, op de doorsnede van kennisrepresentatie, logica, en het semantisch web in een uniforme ontwikkelingsstrategie.

Op het collectieve niveau concentreren we ons op gepersonaliseerde en pervasieve omgevingen vanuit een gebruikersstandpunt, en we onderzoeken hoe hoog niveau abstracties en semantiek, van generische vocabulaire en metadata methodes tot ontologieën, kunnen gebruikt worden voor het creëren van zulke omgevingen zowel als ter verrijking van de ervaring van de eindgebruiker. Onze bijdrage steunt op de conceptuele en praktische benadering die we voordien afleidden. We voorzien de inpakking van digitale en fysische entiteiten met een digitale aanwezigheid in de vorm van widgets en het creëren van webgebaseerde gepersonaliseerde omgevingen via widget-gebaseerde mashups voor de gebruikersinterface. Daartoe beschouwen we eerst ‘widgetization’ van bestaande toepassingen, in bredere zin op het gebied van alomtegenwoordige web navigeren, door het oogsten van semantische in-content annotaties van gebruikersinterfaces. Een ontologiegedreven ontwikkelingsbenadering laat geautomatiseerde annotatie en het genereren van interfaces toe. We introduceren specificaties en mechanismen voor annotatie, extractie en presentatie van ingebedde data. We introduceren een verzameling van heuristieken die domeinkennis gebruiken en ontologie metadata om, met ondersteuning voor het redeneren op basis van de ontologie, gebruikersvriendelijke navigatie te realiseren. Vervolgens introduceren we een open gestandaardiseerd widget platform, een raamwerk voor interoperabiliteit en methodes voor manuele en automatische widget orchestratie. We behandelen de uitdagingen van de interoperabiliteit van gegevens en toepassingen door het invoeren van publieke widget interfaces en semantisch web technieken, in het bijzonder ingebedde semantiek en ontologieën. We faciliteren eindgebruiker datamobiliteit bovenop het voorgestelde raamwerk voor interoperabiliteit voor gebruikersgedreven manuele widget orchestratie. We stellen een datamining methode voor om de patronen van de gebruikers te leren uit de log files. De methode steunt op workflow mining technieken die de topologie kunnen afleiden en op multi-label classificatietechnieken met een label combinatie benadering om de routing criteria te leren. We benutten de geoogste patronen in een vraaggestuurd automatische widget orchestratie.

We vergelijken onze benaderingen en methodes met een brede, interdisciplinaire literatuur. We presenteren prototypes voor elke praktische bijdrage en voeren experimenten met eindgebruikers uit zowel als bruikbaarheids assesments om de computationele haalbaarheid en de bruikbaarheid van de benaderingen en methodes aan te tonen.
Computers are incredibly fast, accurate and stupid; humans are incredibly slow, inaccurate and brilliant; together they are powerful beyond imagination.

Albert Einstein
Acknowledgments

It is a pleasure to thank the many people who made this thesis possible. This thesis would not have been written without the support of many people that have affected my work and life in one way or another.

I am heartily thankful to my supervisor, Prof. dr. Patrick De Causmaecker, whose encouragement, guidance and support from the initial to the final level enabled me to complete this thesis. I wish to thank Prof. dr. Piet Desmet and Prof. dr. ir. Erik Duval for being in my PhD supervisory committee and for taking an interest in my work. I wish to thank in addition Prof. dr. ir. Yolande Berbers, Prof. dr. Miguel-Angel Sicilia, Fridolin Wild, and dr. Katrien Verbert for serving as the members of my jury and Prof. dr. Paul Igodt for chairing the jury.

This thesis is based on research funded by the Industrial Research Fund (IOF) and conducted within the IOF Knowledge platform ‘Harnessing collective intelligence in order to make e-learning environments adaptive’ (IOF KP/07/006). Partially, it is also funded by the European Community's 7th Framework Programme (IST-FP7) under grant agreement no 231396 (ROLE project), Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, and by the Research Fund KU Leuven. I would like to thank everybody who was involved in designing these projects.

I am indebted to dr. Felix Mödritscher and dr. ir. Davy Preuveneers for their insightful comments and constructive criticisms for my research. I am grateful to my colleagues Srikeerthana Kuchi, Francisco Bonachela Capdevila, dr. Bidzina Shergelashvili, and all the members of ITEC-IBBT and CODeS research groups for providing a stimulating and fun environment.

I owe my deepest gratitude to my friends in Ghent, Mustafa Emirik, Selim Körügli, Sacid Örnegül, İqbol Qoraboyev, Halbay Turumtay, Erhan Ilhan Bozkurt, Musa Keleş, Yavuz Soytürk, Mehmet Bayrak, Gezim Bala, Özgür Ceylan, DemirAli Köse, Talha Gökmen, Çengiz Demirkundak, Mehmet Kuşcuoğlu, Sabahattin Sümer, Murat Uzun, Mustafa Dişli, Enes Altan, and many others, who I forgot to mention here, for providing support and friendship that I needed. I have learned a lot from each of them. May God rest my friend Erhan Ilhan’s soul who we recently lost.

I am most grateful to my parents, Sakine and Mustafa, my sister Duygu, and family. The best outcome from these past four years is finding my best friend and wife. I would like to thank Emel for always being with me, sharing my good and bad times, and sticking by my side even when I was irritable and depressed.

Ahmet Soylu
# Contents

**ABSTRACT** .......................................................................................................................... I

**SAMENVATTING** ................................................................................................................ III

**ACKNOWLEDGMENTS** .......................................................................................................... VII

**CONTENTS** ........................................................................................................................... IX

**LIST OF FIGURES** ............................................................................................................... XI

## 1 INTRODUCTION .................................................................................................................. 1

1.1 **PROBLEM STATEMENT AND CHALLENGES** ....................................................... 3

1.2 **SCOPE AND REQUIREMENTS** ................................................................................ 4

   1.2.1 *Development of Context-Aware Software* ......................................................... 4

   1.2.2 *End-User Involvement and Awareness* ............................................................ 5

   1.2.3 *Personal and Pervasive Environments* ............................................................ 5

1.3 **STATE OF THE ART AND DISCUSSION** ................................................................. 6

   1.3.1 *Definition and Characteristics of Context* ....................................................... 7

   1.3.2 *Context Modeling and Reasoning* .................................................................... 7

   1.3.3 *Frameworks for Context-Aware Systems* ....................................................... 9

   1.3.4 *Problems with Context* ................................................................................... 9

   1.3.5 *Discussion and Directions* ............................................................................... 10

   1.3.6 *Ontologies and MDD* ...................................................................................... 11

   1.3.7 *Pervasive Systems and End-users* ................................................................... 11

   1.3.8 *Widgets and Personal Environments* .............................................................. 12

1.4 **APPROACH AND METHODOLOGY** ........................................................................ 14

   1.4.1 *Developing User-centric Pervasive Software* ................................................... 14

   1.4.2 *Web-based Personal Environments* ................................................................. 15

   1.4.3 *Putting Pieces Together* ................................................................................ 18

1.5 **CONTRIBUTIONS** ...................................................................................................... 19

   1.5.1 *Pervasive Computing Revisited* ....................................................................... 19

   1.5.2 *End-user Awareness and Control* .................................................................. 20

   1.5.3 *The Two-use of Ontologies* ............................................................................ 21

   1.5.4 *Ubiquitous Web Navigation* ........................................................................... 23

   1.5.5 *Widget-based Personal Environments* ............................................................ 27

1.6 **STRUCTURE OF THE TEXT** ..................................................................................... 30

## 2 SELECTION OF PUBLISHED ARTICLES ........................................................................ 33

2.1 **CONTEXT AND ADAPTIVITY IN PERVERSIVE COMPUTING ENVIRONMENTS: LINKS WITH SOFTWARE ENGINEERING AND ONTOLOGICAL ENGINEERING** ................................................. 35
2.2 **FORMAL MODELLING, KNOWLEDGE REPRESENTATION AND REASONING FOR DESIGN AND DEVELOPMENT OF USER-CENTRIC PERSVATIVE SOFTWARE: A META-REVIEW** .............................................. 75
2.3 **UBQUITOUS WEB NAVIGATION THROUGH HARVESTING EMBEDDED SEMANTIC DATA: A MOBILE SCENARIO** ................................................................. 127
2.4 **MASHUPS BY ORCHESTRATION AND WIDGET-BASED PERSONAL ENVIRONMENTS: KEY CHALLENGES, SOLUTION STRATEGIES, AND AN APPLICATION** ........................................... 155

3 **CONCLUSIONS AND FUTURE RESEARCH** ................................................................. 201

3.1 **CONTRIBUTIONS** ........................................................................................................ 201
3.2 **DISCUSSION AND OPEN PROBLEMS** ........................................................................ 204
3.3 **FUTURE RESEARCH** .................................................................................................... 206
3.4 **CONCLUDING THOUGHTS AND TRENDS** ................................................................. 207

**BIBLIOGRAPHY** ............................................................................................................. 209

**LIST OF PUBLICATIONS** ............................................................................................... 215

**BIOGRAPHY** .................................................................................................................. 219
List of Figures

Figure 1.1: Abstractions as medium of adaptation and interoperability. 16
Figure 1.2: The Web as a pervasive computing framework. 17
Figure 1.3: Overall research framework. 19
Figure 1.4: An example site consisting of three annotated HTML pages. 24
Figure 1.5: A fragment of an annotated HTML page. 25
Figure 1.6: An example demonstrating SWC. 26
Chapter 1

Introduction

Pervasive Computing (a.k.a. Ubiquitous Computing) [1] and Adaptive Systems [2] are two important interlinked research domains driving the current shift in technology. On the one hand, Pervasive Computing envisions a new user ecosystem that integrates the human layer of earth and the digital space. The vision manifests that devices and applications in this ecosystem should collectively realize unobtrusive, any-where and any-time user experiences by intelligently meeting the needs of people. The pervasive computing research has strong focus on the seamlessness of technology, i.e., the technology should be immersed into the life so that people do not even notice it. As a response, the concept of context-awareness [3] has emerged; context-aware systems perceive characteristics and situations of the entities relevant to the computing setting (e.g., people, devices etc.), i.e., context [4], to tailor themselves accordingly. On the other hand, Adaptive Systems research, which has a longer history, mainly focuses on the user, through user models (e.g., [5]), to personalize the end-user experience by adapting content, interface, application behavior etc. One can say that context-aware systems consider adaptation in a broader scope; nevertheless, the experience gained in personalized systems has been crucial for realization of context-aware pervasive environments.

Adaptivity is the key notion for both research domains; adaptivity can be seen as a solid form of software intelligence from a user perspective. In a pervasive environment, the notion of intelligence, as well as the environment itself, can be considered at two levels. The first one is at the individual application level, that is, an application in a user environment can provide adaptive services to the users. The second one is at the collective level, that is, a set of applications can serve to the users by combining their functionalities in concert, while each application might or might not maintain adaptivity at the individual level. The latter also necessitates heterogeneous and distributed applications to be able to interoperate. Moreover, the traditional nature and means of interaction between users and applications need to evolve, since the technology becomes more invasive. Accordingly, we consider the followings among the main research directions, at a higher level: software intelligence, interoperability, and end-users interaction. There has been considerable research in each fold such as sensor networks for the aggregation of contextual information (e.g., [6]), context modeling and representation for context abstraction and reasoning (cf. [7]), middleware support for sharing contextual information and enabling interoperability between physical appliances and applications (e.g., [8]), multi-modal interfaces for enabling natural user interaction etc. (e.g., [9]).
The work presented in this thesis is driven by a skeptical perspective towards full machine control, manifested by the mainstream pervasive computing vision, due to increasing development complexity, the narrow borders of software intelligence, and the intellectual characteristics of the end-users. Accordingly, in general, we focus on approaches, methods, techniques, and software support to facilitate design and development of user-centric adaptive and (personal) pervasive applications and environments. We mainly investigate how high level abstractions and semantics, varying from generic vocabularies and metadata approaches to ontologies, can be exploited to create such applications and environments and to enrich and augment the end-user experience.

There are two complementary directions to meet the aforementioned concerns. The first direction focuses on the development aspects of adaptive and pervasive computing systems while the second direction focuses on aspects regarding the interaction between these systems and the end-users. In this thesis, we are mainly interested in the following high level challenges, some of which we tackle with at conceptual level only:

1. providing developers with abstract development approaches, methods, and tools to facilitate development and management of complex adaptive pervasive applications and systems,
2. enabling end-users to be aware of the relevant context, to conceive the reasoning behind the behaviors of adaptive systems, and to be involved in adaptation process accordingly,
3. enabling end-users to generate their own personal (digital) and pervasive environments by populating distributed applications and appliances and to blend the functionalities of these entities in order to realize own experiences.

The first two challenges, in this thesis, mainly take place at the individual application level. They constitute the conceptual body of our research and guide our practical studies. The notion of context is in the very core of these two challenges. In pervasive computing domain, context modeling and representation is long studied and aimed for making better use of contextual information, possibly with high level semantics and domain knowledge, (i.e., inference and reasoning) to provide adaptive experiences. Ontologies (cf. [7, 10]) have been one of the key instruments for this purpose while other approaches remained proprietary. Nevertheless, the use of ontologies has been mostly limited with reasoning purposes; however, firstly, ontologies carry a considerable potential both as development-time and run-time artifacts, and secondly ontologies can act as a medium of communication and control between applications and the end-users, with ability to inform the end-users about relevant context and to explain the reasoning behind adaptations. In this thesis, we first review main characteristics of Pervasive Computing from development and end-user perspectives, elaborate on main challenges, and explore the use of high level abstractions to meet these challenges. This allows us to criticize the pervasive computing vision, to grab the overall picture of the domain, and to define a realistic research trajectory.

The third challenge is at collective level. It constitutes the practical body of our research and follows the perspective and approach derived by our conceptual study in terms of end-user considerations and high level abstractions. We work on several complementary tracks to realize the third challenge: (1) a platform, which is open,
generic and standard-oriented, as a basis of the user environment, (2) means to represent entities (i.e., applications, data sources, and devices with digital presence) in the user environment, (3) an interoperability framework to enable interplay between these entities, (3) means to orchestrate entities in personal environments manually or automatically (with respect to user interaction patterns). We evaluate each practical contribution with respect to feasibility, performance, usability, and broad literature. The approach based on high level abstractions and semantics is in continuum while addressing all these challenges. Particularly, ontologies, at representation level, act as a medium of interoperability between distributed and heterogeneous applications. The semantic power of ontologies facilitates the extraction of interaction patterns of the users. Last but not least, the abstraction power of ontologies enables access to applications from variety of platforms and environments.

This thesis is based on international publications that emerged as a result of our studies; these publications are overviewed in this introductory chapter which places the research within a broader perspective, outlines the theme of our research, and establishes links between the individual articles. The remainder of this chapter is organized as follows. We first present the problems, that we address, and state the main context and challenges in Section 1.1. We describe scope and requirements in Section 1.2. In Section 1.3, we address the main state of the art to situate our work in a broad context. In Section 1.4, we describe our approach and methodology. We present our contributions and related literature, to justify our contributions, in Section 1.5. Finally, Section 1.6 presents an outline of the following chapters which take place in the rest of this thesis.

1.1 Problem Statement and Challenges

Adaptivity is the main pillar of both Adaptive Systems and Pervasive Computing. The aim is to enhance the end-user experience by enabling entities in the user’s environment to intelligently adapt to the active situation without any end-user involvement. Indeed, adaptation can be considered from two complementary perspectives, namely a development-time perspective and a run-time perspective. Regarding the former, it is called static adaptation or requirement adaptability. It addresses the development-time adaptations to the software for a specific context, preferably, when adaptations cannot be realized at the run-time. Regarding the latter, it addresses dynamic changes in the behavior, interface etc. of the software at the run-time. However, there are two notable problems regarding the implementation and the use of this notion from the perspective of two main stakeholders, namely developers and naive (inexperienced) end-users.

On the one hand, from a developer or development perspective, adaptation logic is mainly hard-coded into the application, domain knowledge mostly remains unexploited, and application knowledge is mostly implicit. Firstly, considering ever-increasing context space and the application complexity, current approaches remain inefficient for the development and management of large scale software systems. Secondly, due to imperfectness of the contextual information, it becomes challenging to ensure the appropriateness of the adaptations and the consistency of the system. On the other hand, from a naive end-user perspective, adaptivity alone remains deficient
for creating successful pervasive and adaptive end-user experiences. Firstly, these systems mainly follow a black-box approach with absolute machine control where the context and reasoning logic behind the adaptations are totally unknown to the end-users. This results in decreased user engagement, trust, and acceptance (cf. [11]) and this negative affect is propagated with inappropriate adaptations. Secondly, it is virtually impossible to identify the variety of context dimensions, situations etc. and to define (or mine) adaptation rules for every possible scenario. Thirdly, the end-user environment is considered to be a mere input for the user experience where the entities in the user environment are pre-populated and/or interactions between these entities are pre-designed by skilled users or developers. Such an approach limits the end-users and the end-user experience with the imagination of developers. In short, the domain lacks sustainable and efficient development approaches, means for end-user awareness and control, scaffolding support for the construction of personal and pervasive environments.

In this respect, our goal is to support design and development of user-centric personal and pervasive applications and environments in which the end-users: (1) can interfere with the adaptive behaviors of the applications (at individual application level), (2) can acquire information regarding relevant execution context and conceive the reasoning behind adaptive behaviors (at individual application level), (3) can gather, organize, and blend the functionalities of distributed and heterogeneous applications (i.e., orchestrate, at collective level). These challenges require: (a) efficient and systematic development approaches for the rapid and sustainable development of pervasive and adaptive applications; (b) applications to be able to communicate reasoning logic to the end-users; (c) applications to be able to communicate relevant context to the end-users; (d) end-users to be able communicate their requirements/needs to the applications (a and b are prerequisite); (e) generic platforms in order to enable end-users to aggregate applications, data sources, and appliances, to form personal and pervasive environments; (f) standard frameworks to enable the interoperability between the member entities; (g) approaches and encapsulation mechanisms to enable ubiquitous access to member entities from variety of platforms; and (h) facilities and algorithms to enable manual and/or automated orchestration of the member entities. The work presented in this thesis addresses (1) and (2) (i.e., a to d) only at conceptual level and (3) (i.e., e to h) at practical level.

1.2 Scope and Requirements

In this section we narrow down the scope of our research, specify the extent to which we address each of the aforementioned challenges, and translate these challenges into requirements.

1.2.1 Development of Context-Aware Software

Existing development approaches for traditional software fall short for the development and management of complex pervasive and adaptive systems and applications. This is because, on the one hand, existing approaches are not able to
tackle with ever-increasing application knowledge. On the other hand, it is inflexible and error prone to extend and alter adaptation logic, embedded into application code, for new scenarios. Abstract approaches allow automated development and preserve the application knowledge for further changes (i.e., incremental development). It is also equally important to be able to use abstract knowledge and semantics of the application and the domain as a run-time artifact for dynamic adaptations. This will facilitate the management of adaptation logic and ensure the consistency of adaptations and contextual information.

Developers need to be supported with approaches, methods, and techniques, based on higher level abstractions and semantics, for the efficient and sustainable development. Abstractions have been used for automated software development in terms of models, and for run-time adaptation in terms of ontologies. However, both uses have been considered in isolation to each other. In this thesis, at conceptual level, we explore a unified approach, which can exploit expressive abstractions (i.e., ontologies) for development-time and run-time.

1.2.2 End-User Involvement and Awareness

The pervasive computing vision promises seamless end-user experiences. This necessitates isolating end-users from the operational context. Mostly, little or no evidence is provided to the end-users regarding the reasoning behind adaptive behaviors. Our perspective is that end-user involvement is inevitable for the successful realization of adaptive and pervasive systems. This is not only because that software intelligence remains insufficient to enumerate and address all eventualities; it is also because of the intellectual presence of the human-beings. It is not possible to consider human beings as a piece of software that will function efficiently when fed by appropriate data; therefore more effort should be put on the end-user aspects. Therefore, this requires enabling context-aware applications to communicate relevant contextual information and reasoning logic to end-users and allowing end-users to interfere with the adaptation logic.

Formalized abstractions (i.e., ontologies) have potential as a communication medium between the end-users and applications. This is because; contextual information and reasoning logic are preserved explicitly as run-time artifacts. Accordingly, we explore the main characteristics of Pervasive Computing and identify prominent end-user aspects. At conceptual level, we criticize the pervasive computing vision from a user-centric perspective and explore how the power of ontologies and semantics can be exploited for the end-user considerations.

1.2.3 Personal and Pervasive Environments

End-users must be supported with appropriate means to reflect on their own state of affairs, including the environment, while extending their physical, sensory, and mental abilities. In this respect, the practical body of our work focuses on personal and pervasive environments from an end-user perspective. This puts the user environment as a whole under focus, rather than individual applications, in terms of end-user involvement.
We would like to stress that the borders of one’s environment is not set by the physical vicinity anymore; thanks to today’s communication and Internet technologies, the human being can be multi-present. Therefore, the notion of connectedness is the key. A personal environment consists of physical and digital entities which the user is directly or indirectly connected to and thus has potential impact on. The goal is to provide end-users with a unified interaction experience, over a collection of distributed digital entities, and necessary affordances to blend functionalities of these entities and to stimulate the exploration of new possibilities.

We consider the Web as a main medium of pervasive computing environments; therefore, we opt for exploring, using, and extending the standards and specifications of the World Wide Web Consortium (W3C), when available and necessary. We particularly aim for web-based personal environments due to the broad ubiquity of the web technologies.

The very first requirement is a common form of representation and access for/to digital and physical entities (with digital/web presence). Secondly, generic and standard platforms along means for the aggregation and management of user entities (i.e., applications, data sources, appliances) are required. An interoperability framework is necessary to enable user entities to operate collectively. End-users should be enabled to manually orchestrate the entities in their environments (i.e., by being able to copy data from one entity to another). Entities, in a user environment, should be able to automatically react to relevant events happening in each other, as a result of user interactions at the interface level. The integration of entities should be seamless while the automation of interplay (i.e., orchestration) between entities should carry a demand-driven characteristic rather than being pre-defined. The automated interplay between widgets can be realized through learning user interactional patterns.

We employ web-based widgets for the representation of digital and physical entities. Widgets act as the building blocks of personal and pervasive environments. In this respect, we look for means for the automated widgetization of applications. Ontologies are in the very core of proposed solution strategies, particularly as medium of interoperability, widgetization (ubiquitous access in a broader sense), and automated/manual orchestration.

### 1.3 State of the Art and Discussion

In this section, we provide an overview of the main state of the art since a thorough review of the domain, that we address, is presented in two review publications that take place in Chapter 2 (see Section 2.1 and Section 2.2 for the corresponding publications). In addition to that, each publication, in this thesis, elaborates on the relevant literature in a comparative manner. In this section, our focus is on the main characteristics of context and context-aware systems, design and development of pervasive systems, end-user considerations, and widget-based personal environments. We indentify and situate our research directions with respect to the review presented in what follows.
1.3.1 Definition and Characteristics of Context

Context is the key concept for pervasive systems. The initial definitions of context include enumeration of types and entities relevant to the user such as location, nearby people, and objects etc. (e.g., [12-13]). After elaborating on different definitions and enumerations of context, Dey et al. [4] provide a more general and widely accepted definition of the context:

*Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between the user and application, including the user and applications themselves* [4].

The aforementioned definition emphasizes the openness of the notion. Greenberg [14] points out that it is not always possible to enumerate a priori a limited set of context that matches the real world context. Dey et al. [4] refer to the same point and remark that it is not possible to enumerate all important aspects of a situation. Winograd [15] discusses how one can decide whether a piece of information is context or not and states the following:

*...something is context because of the way it is used in interpretation, not due to its inherent properties. The voltage on the power lines is a context if there is some action by the user and/or computer whose interpretation is dependent on it, but otherwise is just part of the environment* [15].

Context dimensions (i.e., atomic context elements) are mostly dynamic, i.e., dynamic construct, where static context dimensions (e.g., gender of a user) also exist [14, 16]. Greenberg [14] states that context is indeed dynamically constructed, that is, it evolves by time (such as the knowledge of a user) and suggests not supporting a particular context but evolution of the context. Context is relational, in other words, different context dimensions are interrelated [17]. In this respect, the perception of context is not only limited with the acquisition of a set of context dimensions but also the relationships in between. Last but not least, context is imperfect [18] due to ambiguity, irrelevance, and incompleteness of the context dimensions (e.g., multiple sensors providing different readings for the same context dimension).

1.3.2 Context Modeling and Reasoning

Modeling and representation of acquired contextual information and reasoning over it are of crucial to identify relevant situations and to adjust application behaviors accordingly. There exist various approaches for context modeling and representation which particularly differ in the level of expressiveness. Availability and the level of reasoning support are mostly attached to modeling and representation paradigm used.

Strang and Linnhoff-popien [18] analyze several approaches in the literature with respect to data scheme used. The authors categorize existing applications into (1) key-value models, (2) markup scheme models, (3) graphical models, (4) object oriented models, (5) logic based models, and (6) ontology based models. Key-value pairs are the simplest approach for the modeling and representation of contextual information (e.g., [19]). They are very limited in expressivity; interpretation and reasoning are application dependent and mostly embedded in application code. They are inefficient for the creation of complex information structures and are mainly based on exact key-
value matches. Markup approaches are based on a fixed hierarchical structure consisting of tags, with attributes, associated with content (e.g., [20]). Compared to the key-value pairs, they allow expression of more complex relationships and are often serialized in form of RDF or XML. Similar to the key-value approaches, their interpretations and reasoning are application dependent. The wide acceptance and use of Unified Modeling Language (UML) in the software engineering domain is one of the main motivations for using graphical approaches for context modeling and representation. One of the most prominent advantages of graphical approaches is their strength in structuring based on human-friendly visual constructs (e.g., [17]). Object Constraint Language (OCL) is used to provide reasoning support. Nevertheless, the expressivity and reasoning support of UML and OCL are limited due to lack of a formal ground (cf. [21]). The driving force behind the use of object oriented approaches is the intention to employ benefits of the object oriented paradigm such as encapsulation and reusability (e.g., [22]). Typically a compiler is employed to validate the structure and a run-time system is used to validate the instances. Object oriented approaches usually suffer from lack of automated inference. In logic based approaches, context is modeled in terms of facts, expressions, and rules (e.g., [23]). Logic based approaches focus on form to arrive at logical conclusions rather than content representation. Therefore, in such systems, contextual information added and updated from the system and new facts are inferred respectively. Although logic based approaches are quite strong in terms of expressivity and reasoning, existing approaches miss partial validation and knowledge share. Ontologies [10], as a Knowledge Representation (KR) paradigm, focus on content. Gruber and Borst [24] define ontology as a formal and explicit specification of a shared conceptualization where a conceptualization refers to an abstract model of a phenomenon in the world by identifying the relevant concepts of that phenomenon. Formal refers to the fact that the ontology is formulated in an artificial machine readable language which is based on some logical system like First Order Logic (FOL) [25]. Ontologies are favorable for context modeling (cf. [26]) due to their expressiveness and explicit support for context reasoning and for dealing with context ambiguity. Strang and Linnhoff-popien [18] conclude that the use of ontologies is promising for context modeling. Wang et al. [27] list the following reasons to use ontologies for context modeling: knowledge share, logical inference, and knowledge re-use. Held [20] lists following requirements for the context modeling: structured, interchangeable, extensible, standardized, and composable / decomposable. Apparently, ontologies are promising to meet the main characteristics of context, i.e., open, dynamic, and imperfect, and the aforementioned requirements for context modeling and representation.

Ontologies have been widely used for context modeling and representation. Gomez-Perez et al. [10] categorize techniques to develop ontologies: AI based, software engineering oriented (UML), database oriented (ER, EER), and application oriented (e.g., key-value pairs). Ontologies can be considered in terms of heavyweight and lightweight ontologies. A lightweight ontology includes concepts, concept taxonomies, properties, and relationships between concepts, and in the simplest case an ontology describes a hierarchy of concepts. A heavyweight ontology requires suitable axioms in order to express more complicated relationships and constraints. In other words, heavyweight ontologies describe a domain with more constrains and higher expressiveness. Gomez-Perez et al. [10] point out that, AI based techniques,
such as Description Logics (DL), are more expressive and allow development of heavyweight ontologies. Among other AI based approaches such as Bayesian networks, fuzzy logic etc. (e.g., [28-29]), the use of ontologies are widely common due to their strength in knowledge share and re-use. Various context ontologies have been proposed (e.g., [26-27, 30]), mostly based on OWL-DL (Web Ontology Language of W3C). SOUPA [26] and CONON [27] are among the most prominent early examples.

1.3.3 Frameworks for Context-Aware Systems

Several architectures, employing ontologies, have been developed particularly for smart spaces. SOCAM [8], CoBrA [31], and GAIA [32] are notable among other ontology/non-ontology based approaches (e.g., [33-36]). Chen et al. [31] propose a system, named Context Broker Architecture (CoBrA), which is based on a central context broker and utilizes SOUPA ontology, to support context-aware systems in smart spaces. The context broker is in the core of the CoBrA and is responsible of: (1) providing a centralized context model to be shared with agents, services, and devices in the space, (2) acquiring contextual information, (3) reasoning about acquired contextual information, (4) detecting and resolving inconsistent knowledge, and (5) protecting user privacy. Gu et al. [8] propose a service oriented context-aware middleware, named SOCAM which employs ontology based context models (adopts SOUPA and CONON ontologies), for building context-aware services. The SOCAM architecture includes context providers (acquiring contextual information from heterogeneous sources), context interpreter (providing reasoning services), context database (storing contextual information), context-aware services (providing tailored services with respect to context), and service locating service (providing a mechanism to enable context interpreter and context-aware services to announce their presence). Ranganathan et al. [32] propose a system named GAIA, which is based on CORBA (Common Object Request Broker Architecture) to enable distributed entities to communicate, for smart spaces. These systems mainly provide components for collecting, processing, storing etc. of contextual information. They also provide support for programming pervasive spaces to deliver adaptive services. In such systems, logic rules are defined and executed over a context ontology.

1.3.4 Problems with Context

One of the biggest challenges for the context-aware systems is the imperfectness of the contextual information. In a pervasive computing environment, it is quite crucial to operate adequately when conflicting, imprecise, uncertain, and ambiguous contextual information is available, since pathological adaptations might have severe consequences over the end-user experience. There exist several methods for dealing with the imperfectness of contextual information. A quite common approach is to annotate contextual information with quality parameters such as precision, confidence, updtodateness, trust level, granularity, certainty, accuracy, freshness etc. (e.g., [37-39]). Truong et al. [40] remark that related approaches in the literature for reasoning about uncertainty with various metadata terms, such as confidence and accuracy, are not expressive enough to capture the rich types of context information
and to support the reasoning mechanism. The authors combine Bayesian networks and ontologies due to the expressiveness of ontologies and the probabilistic strength of Bayesian networks. Although different AI techniques have been used to alleviate the imperfectness of contextual information such as Fuzzy Logic, Bayesian Networks, Hidden Markov Models, and hybrid approaches etc. (cf. [41-42]), it is not possible to reach full success. For this reason, end-user involvement emerges as an important paradigm. Dey et al. [43], by building on the past work of Mankoff et al. [44], propose an approach for involving end-users to address context ambiguity through mediation. We refer to the publication presented in Section 2.1 for a more detailed review.

1.3.5 Discussion and Directions

The state of the art presented up to now suggests that the use of ontologies is preferable over other ad-hoc approaches which lack expressivity and extensibility. The state of the art also reveals several shortcomings and problems. Existing context-aware systems are mostly small scale. Considerable effort has been put towards collection and management of contextual information through middleware and framework support; however, development approaches for large scale pervasive applications are not truly investigated. Separate and substantial efforts are required for the development of application, application logic, and application interfaces suitable for various platforms and devices. Existing approaches and systems leave no room for the end-user considerations and hinder user trust, acceptance, engagement etc. Adaptation logic is mostly pre-designed and generic means for enabling end-users to have awareness of their own context and to define and control adaptation policies are not provided. The work regarding the end-user involvement for alleviating context imperfectness is very limited. Therefore, it is not guaranteed that the adaptations reflect the demands of end-users and that the emerging adaptations are sound. There is a substantive focus on fully automated adaptations leaving almost no control to the end-users. Existing context-aware systems are limited with smart environments based on specific locations (e.g., meeting room, conference hall etc.). Borders of one’s environment are mostly drawn by physical proximity and covers nearby objects. Such an approach omits applications available in the digital space and the fact that users indeed can be digitally multi-present in different locations. Smart environments are pre-designed; hence, end-users do not have any control over the design of their own environments. Last but not least, end-users are not provided with means to control entities in their environments with ability to blend and orchestrate their functionalities for their own needs. In this respect, our goal is to investigate unified development approaches for efficient development of large scale pervasive applications and systems, to identify relevant concepts and criteria for the end-user involvement in terms of end-user control and awareness, to provide means to enable end-users to design and orchestrate their own personal and pervasive environments. In what follows, we continue our review with a specific focus on these matters.
1.3.6 Ontologies and MDD

Regarding development of large scale pervasive applications and systems, the most tempting direction is to use Model Driven Development (MDD) approaches and tools. Several researchers indeed have employed UML for modeling contextual information (e.g., [17, 45]). Knublauch [46] points out that, models can be used not only for automated code generation but also as executable software artifacts. This explains why, in the literature, there are efforts employing UML for context modeling (due to simplicity of UML) and employing ontologies as a modeling formalism (due to its expressivity) for MDD (cf. [47]). Recently, Serreal et al. [48] employed MDD for automated code generation and ontologies for run-time reasoning. The modeling is based on UML meta-model and ontologies are derived from the initial models. The problem with UML based approaches is that the resulting models not expressive enough and UML do not provide explicit reasoning support (although that can be realized up to some extend with Object Constraint Language – OCL – it lacks a formal ground). Therefore, the use of ontologies as a modeling paradigm for automated development and as a run-time artifact for adaptation logic is more promising. Ruiz et al. [47] elaborate on two uses of ontologies as a development-time artifact and run-time artifact; however, both uses are considered in isolation. From a practical perspective, several researchers try to establish mappings between software artifacts (SQL, Java etc.) and ontologies (OWL based) to realize transformations from ontologies to software artifacts (e.g., [49-50]). However, the main challenge is completeness at the moment, that is transforming an ontology to a software artifact without any loss of information. The semantic web technologies, particularly OWL, are commonly used in the current literature for ontology development due to their integration with the Web. However, the main challenge is the immaturity of the logic layer (where logic rules are employed); there is a considerable effort towards maturing the logic layer of OWL in the current literature (e.g., [51-52]). In short, substantial amount of work is required for the realization of a unified approach. We refer to the publication presented in Section 2.2 for a more detailed review.

1.3.7 Pervasive Systems and End-users

Regarding the end-user considerations, the existing work is very limited, disconnected, and is not built on a solid conceptual framework. Mankoff et al. [44] and Dey et al. [43], in their early connected studies, provide mechanisms for end-user mediation. They address situations when an ambiguity in contextual information is detected. The proposed mediation mechanism and the grounding context model are rather ad-hoc. Dey et al. [53], in their later study, elaborate on the concept of intelligibility, that is allowing end-users to understand why an application behaves in a particular manner, and introduce situations within their context model. An adaptation rule is associated with a specific (set of) situation(s). The context-aware system discloses the situations and associated rules to the end-user through the end-user interface. Their approach is based on their early framework and on an ad-hoc context model. Ontologies maintain contextual information in a form that is ready to be communicated with the end-users; moreover the reasoning chain of adaptation rules can be disclosed to the end-users. Niu and Kay [54] employ ontologies for...
context modeling and adaptation logic and communicate the reasoning behind adaptations to the end-users by disclosing the reasoning chain. However, a complete methodology or framework is missing. The conceptual body of the literature suggests that user engagement (cf. [55]), user trust (cf. [56]), and user acceptance (cf. [11]) are crucial elements for realizing successful systems. For this purpose, end-user situation awareness, perceived user control, software intelligibility, and self expressiveness hold a crucial role (cf. [11, 53, 57]). The literature reveals the potential of ontologies for addressing the end-user considerations due to its strength in knowledge share, logic based reasoning and ability to act as a communication medium between end-users and applications; yet, a systematic approach is missing. We refer to the publication presented in Section 2.2 for a more detailed review and discussion.

Regarding the end-user environment, the existing work is mostly based on pre-designed environments. Entities in a user-environment exist in terms of services and these services act based on the contextual information aggregated through sensors. Adaptations are mostly in first-order (cf. [58]) (i.e., pre-programmed by a skilled user), approaches based on second-order adaptivity (i.e., adaptation rules are learned by the system) rely on AI techniques where adaptive actions are probabilistically selected. Firstly, such approaches often give no opportunity to the end-users to aggregate their own environments and blend functionalities of the involved entities. Secondly, there exists only limited work on enabling traditional web applications to be utilized in user environments as entities; an approach for this purpose requires access to traditional applications from a variety of platforms. There exist a few studies on personal environments, not directly associated with the pervasive computing domain. We aim at adopting and extending the work that is available on personal environments for construction of personal and pervasive environments. Existing work mainly employs user interface mashups based on widgets. In this respect, we consider widgets as a prominent paradigm for the encapsulation of digital and physical entities (cf. [59-60]). This becomes possible by widgetizing traditional applications and associating widgets with the functionalities of digital appliances (cf. [61]). Such an approach will enable end-users to aggregate distributed entities into a common space, to synchronize their behaviors for their needs, and even to program their own spaces. In this respect, a set of challenges do arise such as the widgetization of existing applications, widget interoperability, availability of standard widget platforms, and widget orchestration (i.e., how different widgets react to each other’s state changes).

1.3.8 Widgets and Personal Environments

Widgetization of existing applications is not directly addressed in the literature yet. However, there exist studies towards ubiquitous access and user adaptive interface generation. There exists a considerable work on model-based interface generation in which formal models are used to automatically generate user interfaces (cf. [62-64]). Among other approaches for pervasive interfaces, Lei et al. [64] propose a pattern based approach in which interfaces are generated based on context-dependent patterns; Leichtenstern and Andre [65] propose a rule-based approach in which interfaces are adapted with respect to the different contextual situations; and Paterno et al. [66] propose an approach, built on XML-based languages, for authoring multimodal interfaces. Regarding the user adaptive interface generation, Anderson et
al. [67] report on website personalizers which observe the browsing behavior of website visitors and automatically adapt the pages to the users; Buyukkokten et al. [68] examine methods to summarize websites for handheld devices. However, the main problem with these approaches is that they are either based on substantial authoring efforts or require complex AI processing of content; hence they are not integrated to a uniform development process and require separate efforts. The existing work based on semantics usually focuses on linked data browsers with a data-oriented perspective (e.g., Tabulator, Sig.ma [69], Disco, Dipper etc.). However, these approaches are not tailored for the end-users and do not address the end-user consumption. There exist some domain specific linked data viewers (cf. [70]). Fallucchi et al. [71] propose a semantic web portal for supporting domain users in organizing, browsing and visualizing relevant semantic data. Auer et al. [72] follow a template-based approach for improving the syndication and the use of linked data sources. The domain-specific nature of these approaches allows them to present data in a form suited to its characteristics (e.g., for instance geographical data can be presented in a map). However, prior need for the domain knowledge restricts these approaches to content producers and mashups for specific presentation environments. Our aim is, by following a model-based approach, to harvest semantically annotated information from the automatically generated and annotated interfaces, and to re-generate the application interface in a much simpler form. We refer to the publication presented in Section 2.3 for a more detailed review of the relevant literature.

Early widget-based approaches such as Yahoo widgets, Google gadgets are ad-hoc and do not support interaction between widgets (indeed between entities). More advanced approaches, prominently Intel Mash Maker [73], mashArt [74], and Mashlight [75], allow designing user interface mashups. However, since such approaches are highly design driven (in terms of data and event/behavior mappings), they are not appropriate for naive end-users. Apart from other approaches, which utilize widgets for the composition of services by means of visual programming support (e.g., [75]), Friedrich et al. [76] and Govaerts et al. [77], in their interlinked studies, propose an interoperability framework for personal environments employed in e-learning domain. However, they utilize an approach based on inter-widget communication in which widgets subscribe to each other and act according to the received events. Each widget decides on its behavior with respect to content of the received events. Similarly, Wilson et al. [78] propose several approaches for widget orchestration, the proposed approaches are mainly design oriented and distributed (based on inter-widget communication). In a design oriented approach, widgets disclose their functionalities and a skilled user programs the behavior of the widgets with respect to possible events. In the distributed approach, widgets subscribe to each other with respect to a topic ontology and react accordingly. Firstly, a distributed approach or pre-design driven approach does not reflect the end-user demand. Secondly, a distributed orchestration approach based on syntactic or even semantic event – interest match does not guarantee emergence of a sound orchestration. When several widgets react autonomously to the same events, chaotic situations might also arise.

Wilson et al. [78] employ W3C widget specification (cf. [79]) and propose extensions. We do agree with the need for extensions. There remains major room for extensions towards realizing mashups by orchestration (and other approaches), for
instance, for communication, event delivery, functional integration, end-user data mobility (to allow end-user copy data from one widget to another effortlessly – particularly important for user-driven orchestration) etc. Regarding the architecture, existing work is mainly repository centric (e.g., Wookie), that is most of the services (communication, preference management etc.) are aggregated into the widget repositories which serve the widgets. Such a centralized approach is inflexible and overloads repositories by aggregating services and tasks, which should normally be provided by a client-side run-time system, to itself (e.g., widgets coming from different repositories cannot communicate).

Finally, literature lacks a appropriate means for representing and modeling adaptive behaviors. The existing approaches which try to exploit user footprints mostly for different purposes (e.g., programming by demonstration – cf. [60]). These approaches miss a formal ground for harvesting and representing user behavioral patterns. This makes validation, verification, and share of patterns almost impossible.

We aim at employing ontologies to annotate events, widget functional interfaces, and content to enable interoperability and to enhance behavioral pattern mining. A unified ontology-driven development approach allows automated annotation and development of widgets. We refer to the publication presented in Section 2.4 for a more detailed review and discussion of the relevant literature.

1.4 Approach and Methodology

The literature presented leads us to a uniform approach where ontologies play a crucial role for the design and development of user-centric pervasive applications; end-user considerations in terms of intelligibility, end-user situation awareness, user control; widgetization of existing applications; widget interoperability; and orchestration in widget-based personal environments. In what follows, we clarify our approach and methodology in more detail.

1.4.1 Developing User-centric Pervasive Software

Regarding the development and management of pervasive software, the use of high-level abstractions (i.e., ontologies), merged with a MDD [80] approach, can exploit ontologies both as a development-time and run-time artifact at the same time. Ontologies enable explicit preservation of the adaptation logic and software knowledge. Software can make use of the domain knowledge and semantics through ontologies as executable artifacts. This leads to efficient development and management of pervasive software systems and applications. Considering the imperfectness of the contextual information, ontological reasoning allows checking the consistency of the context; therefore, it ensures the appropriateness of adaptations to a certain level. Ontologies have been used in the literature as development and run-time artifacts; however, a unified approach which combines both considerations with an ontology-driven perspective is missing (cf. Section 1.3). Regarding the end-user experience, it is important that a system supports end-user awareness (end-users should be aware of the context of execution), software intelligibility (the behaviors of the software should be understandable by the end-users), self-expressivity (software
should be able to explain the reasoning behind adaptations), user control (users should be able to interfere with the adaptations by means of mediation, recommendations etc.). Herein, high level abstractions are of use for the end-user considerations; an ontology maintains context information explicitly (almost ready to be shared with end-users), is able to explain reasoning logic through revealing the reasoning chain, and acts as a communication medium between the end-user and the application. In the literature, the use of ontologies for the end-user considerations remains unexplored (cf. Section 1.3).

In our first review article, which takes place in Section 2.1, we provide: (1) an extensive meta-level literature review surveying theoretical grounding and main notions of Pervasive Computing, (2) the current perspective in Pervasive Computing, (3) a survey and an elaborate discussion on context, context-aware systems, context modeling and representation, context abstraction, and context reasoning, (4) links between Pervasive Computing, Software Engineering, and KR, with respect to modeling and development of context-aware systems, (5) a forward looking conceptual perspective, stressing on the end-user involvement and the use of high level abstractions, along main research challenges, (6) and a perspective that situates the Web and the semantic web technologies within the broader pervasive computing vision. This review article points out the use of high level abstractions for automated development, run-time adaptations, and for addressing end-user considerations at individual application level. It allows us to grab the broader picture and to synthesize a long-running research perspective. The grounding conceptual study aims at exploring pervasive computing vision along its links with adaptive systems due to strong links between these two research domains.

In a second review, which takes place in Section 2.2, we focus on the end-user considerations and elaborate on main concepts such as intelligibility, end-user situation awareness, user control etc. We provide a discussion on the development of adaptive and pervasive software systems and applications with respect to literature. With the guidance of our first review, we explore how high level semantics, from KR and Logic perspective, can address the development and end-user considerations. We work towards synthesizing a unified approach for merging ontology-driven and model-driven development approaches. We review existing practical work and indentify major challenges required to realize a merged approach. Our practical research line, particularly regarding end-user considerations, is built on our findings and the perspective derived as a result of our reviews.

1.4.2 Web-based Personal Environments

Regarding the end-user environment, it is important that distributed applications, particularly applications with GUIs (i.e., tools) rather than the services in our context, can interoperate and can be aggregated by the end-users to form their own digital environments. This is because composition of services yields to new applications, the composition is mainly task oriented, and entities that are part of the composition are mostly unknown / unimportant to the end-users. However, the aggregation of tools yields to personal (digital) environments, where each entity functions independently, entities that are part of aggregation are known/important to the end-users, and aggregation is mainly experience oriented.
We define a personal environment as an individual's space of applications, data sources etc. in which she is engaged on a regular basis for personal and/or professional purposes. In this respect, UI mashups play a scaffolding role to enable the creation of personal environments and to support cognitive processes, like fostering reflection, awareness, control and command of the environment. Being completed with orchestration, they are intended not only to enhance but also to augment the end-user experience. Nevertheless, a large body of work in the domain deals with service and data level integration. In an ontology-driven environment, different applications can easily share data and functionality at different semantic levels (e.g., only structure, class hierarchy etc.) directly through their end-user interfaces (i.e., with embedded semantic technologies – e.g., microformats, microdata, RDFa, and eRDF (cf. [81])) - see Figure 1.1. In this way, end-users can populate their own environments and generate their own experiences. Generic and domain specific vocabularies can be used for annotating end-user interfaces (including data and functionality); however, this is a tedious task. However, a grounding ontology can allow automated generation and annotation of interfaces with necessary application knowledge and semantics.

**Figure 1.1:** Abstractions as medium of adaptation and interoperability.

The very first question concerns how we represent entities in one’s personal environment. We use web widgets as a medium of virtual encapsulation for digital (e.g., applications, data) and physical entities (e.g., devices, even people). We consider widgets as building blocks of personal environments; therefore, we define resulting environments as widget-based UI mashups. A semantic approach, that enables data and functionality to be semantically annotated within the content (e.g., embedded semantics) and being complemented with service-oriented approaches (e.g., REST), can allow physical devices to serve their functionalities through the Web. This allows data and functionalities of applications to be consumed and to be driven through the end-user interfaces and allows us to exploit the Web as a ubiquitous computing framework (see Figure 1.2) in terms of Web of Data (WoD) (cf. [82]) and Web of Things (WoT) (cf. [61]).

In our first practical article, which takes place in Section 2.3, we approach the widgetization issue in a broader perspective in terms of ubiquitous web navigation. We propose an approach enabling end-users and devices to access and navigate websites along their semantic structure and domain knowledge. Precisely, the approach allows specifying and extracting semantic data embedded in (X)HTML
documents with RDFa. Existing work misses the end-user consumption of the semantic data and is highly data-centric. The proposed approach merges document-oriented and data-oriented considerations. In our context, the approach enables us to automatically widgetize existing applications to realize effortless construction of personal environments. For this purpose, we employ embedded semantics. We propose methods and techniques to regenerate application interfaces through the extracted application knowledge (i.e., data, functionality) for end-user consumption. We provide mechanisms for specifying, extracting, and presenting annotated content. We develop a set of heuristics for fine tuning the specification and presentation of semantically annotated content for the end-user consumption, and propose methods for utilizing domain semantics for enhancing the end-user navigation experience.

![Figure 1.2: The Web as a pervasive computing framework.](image)

In our second practical article, which takes place in Section 2.4, we address the construction of personal environments. We look for means for widget interoperability. We propose an open platform and a reference architecture based on W3C widgets to enable rapid development and prototyping. Standard platform services and components such as event delivery, preference management etc. are described. We describe an interoperability framework for widget-based UI mashups. The framework is based on a standardized communication channel; messaging format for event and delivery and communication; functional widget interfaces for application interoperability; and semantic annotations of content, widget interfaces, and events, for enhanced semantic data interoperability and data mobility. We review and evaluate various approaches to enable the interplay between widgets, at UI level, which we call widget orchestration. Different orchestration approaches can be realized on top of the aforementioned interoperability framework and platform. However, loose coupling is an important criterion, since our goal is not to come up with a proprietary platform or a specific mashup application. We consider the following criteria: physical coupling, communication style, type system, interaction pattern, control of process logic, service discovery and binding, and platform dependencies, as described in [83]. Widget orchestration can happen manually (i.e., user-driven) or automatically (e.g., design-driven, distributed, system-driven etc.). We consider it important to first empower end-users with facilities to orchestrate widgets effortlessly on their own; therefore, we build a facility on top of our interoperability framework for end-user data mobility. It allows end-users to communicate data from one widget to another. An automated approach, if realized in a demand-driven...
manner, can enhance the end-user experience. Among several automated approaches (e.g., pre-designed, distributed etc.), we propose an algorithmic solution for system-driven orchestration. In proposed approach, a widget platform mines behavioral patterns (i.e., the topology of events and the routing criteria) by mining user logs through the adoption of workflow mining and decision mining techniques. The platform automates the interplay between widgets with respect to extracted patterns. Pattern mining approach greatly benefits from ontological enrichments of the event signatures while learning the routing criteria. We propose generic extensions to W3C’s widget specifications with respect to the proposed interoperability framework particularly in terms of communication infrastructure and access to platform services.

1.4.3 Putting Pieces Together

The overall research framework is driven and connected at conceptual and technical levels. Regarding the conceptual level, the main idea is to leave more room for end-user involvement while supporting them with relevant contextual and causal information and means for design and control. End-user considerations are inherently related to design and development, since the current development approaches are not able to produce scalable user-centric software. The articles presented in Section 2.1 and Section 2.2 explore the end-user aspects and establish their links with design and development at conceptual level. The latter further mainly elaborates on the end-user involvement and awareness and ontology-driven abstract development at individual application level (i.e., user vs. application). The articles presented in Section 2.3 and Section 2.4 complement the work addressed conceptually at individual application level. They follow the approach and methodology distilled through our conceptual study and build on each other to tackle with the end-user involvement at collective level (user vs. a set of applications).

At the technical level, the main approach (see Figure 1.3) presented in this thesis is based on employing high level abstractions, particularly ontologies, for design and development of user-centric adaptive and (personal) pervasive environments. The end-users are expected to have a degree of control in adaptation logic and in the design of environment. In articles presented in Section 2.1 and Section 2.2, it is proposed to acquire and model domain and application knowledge through an ontology. The resulting ontology can be used as an executable context ontology for run-time adaptation. The same ontology can be used as a development-time artifact to automatically generate the application. The context ontology is proposed to be used for end-user situation awareness, intelligibility, and user control in order to address end-user considerations. The article presented in Section 2.3 is based on the fact that application interfaces can be automatically generated from an ontology by following ontology-driven development approach. Hence, the user interfaces of the applications can be semantically annotated with respect to the grounding ontology. The approach proposes widgetization of resulting ontology-driven applications (or the ones annotated later) through harvesting annotated semantic data from their interfaces. In the article presented in Section 2.4, the resulting widgets are proposed to be used as building blocks of widget-based personal and pervasive environments. The proposed approaches for widget interoperability, end-user data mobility, and pattern mining
greatly benefit from the semantic enhancement of widget functional interfaces, events, and widget content.

![Figure 1.3](image)

**Figure 1.3**: Overall research framework: (1) abstract development, (2) widgetization/ubiquitous access, (3) personal environments, (4) end-user involvement.

### 1.5 Contributions

The review articles (presented in Section 2.1 and 2.2) define our research objectives and construct our approach. The challenge is to enable end-users to steer their own experiences when interacting with individual applications providing adaptive services and with a set of applications operating in concert. The proposed approach exploits the potential of ontologies to meet development and end-user concerns both at individual application level and at collective level. In the practical body of our research, we focus on the collective level and propose a solution strategy for the widgetization and construction of widget-based personal and pervasive environments.

#### 1.5.1 Pervasive Computing Revisited

Our literature review (see article presented in Section 2.1) [84] reveal that pervasive and adaptive systems and applications are hard to develop and manage. Existing applications mostly remain small scale. It is also seen that one of the most prominent approaches to alleviate this problem is to provide more advanced approaches, based on high level abstractions and semantics, for design and development. Advanced development approaches can be of use for sustainable and efficient development. However, even then, it is not possible to cover all possible eventualities. Moreover, automated adaptations are not always desired by the end-users (e.g., to decide on
correctness of the context and actions or to design own experiences [44, 85]). Therefore end-user involvement to deal with ambiguity of context, and to enable end-users to design and control their own environments and applications (hence experiences) are of crucial importance. The end-user interaction, hence involvement, is considered at individual application level and collective level. The former refers to end-user’s ability to interfere with the adaptation logic of software, while the latter refers to end-user’s ability to freely aggregate and manage her portfolio of digital sources. Our analysis of the literature leads us to conclude that ontologies have potential to address development and end-user considerations. This is because of the followings. (1) Ontologies can be used for automated development by following a MDD approach. (2) In contrast to static adaptations, hard-coded into the applications, ontologies can be used for dynamic adaptations due to their reasoning power. (3) Ontologies can act as a bidirectional communication medium between end-users and applications. Therefore, contextual information and reasoning logic can be communicated to the end-users, and end-user can communicate their demands. (5) Ontologies can act as a communication medium between applications; therefore, ontologies can facilitate distributed applications to be able to interoperate. (6) Interfaces of applications which are semantically annotated, possibly automatically through an ontology-driven development approach, can be re-formed/adapted for ubiquitous access. This is particularly of use for encapsulating traditional applications in form of widgets to support construction of personal and pervasive environments. In short, an approach based on high level abstractions and semantics have potential to deal with increasing development complexity and management, to act as an intermediary communication medium between computers and users, and to allow end-users to populate and manage their personal environments on their own.

1.5.2 End-user Awareness and Control

We first address development and end-user considerations of pervasive computing systems at individual application level (see article presented in Section 2.2 [86]).

Regarding the end-user aspects of Pervasive Computing, there has been a long standing debate on whether machines will be able to achieve human level of intelligence or not. Although there exist very optimistic (cf. [87]) and very pessimistic (cf. [88]) stand points, we believe that, regardless of whether a human level intelligence is possible or not, information and reasoning are keys to move the machine intelligence to a higher level. Reasoning allows better use of information at hand through mining implicit information by making use of the domain knowledge and semantics. Tribus and Fitts [89] points out that indeed there are not right decisions, there are only decisions consistent with the information at hand. Although reasoning enables machines to make better use of information and ensure the consistency of logical inferences as well as information itself, humans are still better at recognizing context and reasoning. However, it is not possible for humans to aggregate and process every piece of information relevant to execution context [90]. This motivates the need for a perspective where human intelligence and machines complement each other. Aggregated contextual information has to be processed and abstracted before being communicated to the end-users due to limited cognitive and processing capacities of the human actors.
Considering human and machine interaction, we identified several notable interlinked requirements for successful implementation of pervasive and adaptive environments from an end-user perspective which are:

- **User engagement** – ability of a system to attract and hold attention of the users (cf. [55]),
- **User trust** – ability of a system to satisfy intentions of users and achieve their objectives (cf. [56]),
- **User acceptance** – user’s intention to use a system and to follow its decisions and recommendations with willingness and commitment (cf. [11]).

We identified the following crucial to ensure user engagement, trust and acceptance:

- **Software intelligibility** to make reasoning behind adaptations clear to the end-users (cf. [53]),
- **Software self-expressiveness** to enable software to communicate its reasoning and relevant context to the end-users (e.g., [91]),
- **End-user situation awareness** to make users aware of the relevant context of the active execution setting (cf. [57]),
- **End-user involvement** to enable users to interfere with the adaptation logic and eventual behaviors (cf. [11]) (software self-expressiveness and end-user situation awareness are prerequisite).

From KR point of view, ontologies are explicit and formalized conceptualizations. For this reason, contextual information captured in an ontology can be communicated to end-users in order to realize user situation awareness. From logic perspective, ontologies can reveal the reasoning chain of adaptation rules (cf. [54]) in order to ensure software-self expressiveness. End-users can use ontologies represented in human consumable form to transmit their needs and demands.

### 1.5.3 The Two-use of Ontologies

Regarding the abstract development (see article presented in Section 2.2), in the current literature, there are several studies either employing ontologies, particularly OWL, as a modeling formalism in MDD (cf. [47]), or employing MDD modeling instruments, particularly UML, as a representation formalism for developing ontologies (e.g., [45]). However, UML does not offer any automatic inference, and there is no notion of logic and formally defined semantics. The use of ontologies in MDD, without aiming at employing ontological reasoning power of ontologies, will only introduce higher complexity. Therefore, a uniform approach is required. In [46], the author points out the potential of ontologies as a run-time and development-time artifact without a complete methodology and elaborate discussion of the approach. In [47], the authors address the use of ontologies, from a temporal perspective, in twofold: ontology driven information systems and ontology driven development of information systems. The authors further review the related work but they consider each fold in isolation from each other. A uniform approach using ontologies through the full software life cycle is not truly realized yet. Regarding the representation formalism, for such a uniform approach, we prefer to use OWL due to its support for integration with the Web. However, OWL-based approaches lack: (1) adequate visualization support, and (2) ability to model dynamic behavior of a system. This
Attracted attention of researchers to investigate means to combine power of ontologies with UML’s ability to specify dynamic behaviors (e.g., [21]). Nevertheless, UML’s lack of a formal ground disables possible use of advanced analysis and simulation methods on behavioral properties of a model. One response to this problem is Petri nets, particularly Colored Petri nets [92]. There already exist efforts toward intertwining capabilities of Petri nets with ontologies and MDD (e.g., [93, 94]). We identified followings as the most prominent properties expected from an abstract model: extensible visual constructs, reasoning and semantic validation, and behavioral analysis and validation. Combination of UML’s user-friendly standardized graphical representation constructs, expressiveness of ontologies, and Petri net’s ability to model, simulate and execute behavioral models is fruitful.

We raise three possible methodologies for such a combination. The first methodology is using each modeling paradigm for a specific purpose, that is, UML for visual design, OWL for semantic validation and reasoning, and Petri nets for behavioral analysis. This approach requires mapping and transformation from models which is a complex process. Secondly, initial models will not be expressive enough since the expressivity of UML is limited. The second approach is based on using OWL KR and its logic layer and rebuilding the meta-models of Petri nets and UML on top of OWL KR and on its logic layer. Domain specific visual constructs can be employed along with subject specific interpreters (e.g., for Petri net models).

However, the first and second approaches do not truly address a possible merger of MDD and ontologies. The third approach employs a natural authoring mechanism; the development starts with identification of relevant concepts, properties etc. as they are without the notion of a software in mind; this leads to an ontology. Then, specific models related to software can be derived from the ontology. Such an approach allows iterating from natural representations to software specific representations of the subject domain. Ontologies are broader than models; ontologies are always backward looking (describe what already exist) while models are mainly forward looking (reality is constructed from it, e.g., software) [95]. We propose an abstract methodology which injects ontologies into the main steps of MDD methodology.

We review related practical work regarding a possible implementation of a merged approach in terms of logic and rule layers, and mappings and transformations from ontologies to software artifacts. The logic and rule layer of the semantic web is still in progress. OWL DL has some particular shortcomings, since the utility of ontologies, in general, is limited by the reasoning and inference capabilities integrated with the form of representation. Integration of logic programming into OWL is known to be required to increase expressiveness of OWL-based ontologies, in terms of higher relational expressivity, higher arity relationships, closed world assumption, non-monotonic reasoning, integrity constraints, exceptions etc. (cf. [51]). Regarding transformation of ontologies to software artifacts, existing work mainly addresses mappings and transformations from RDF or OWL to object oriented program code (e.g., Java) or to relational databases (e.g., SQL) (e.g., [49, 50]). One notable challenge is decidability, because it is not possible to implement every software construct in an ontology due to decidability considerations; another challenge is completeness since not every OWL construct can be mapped into a software artifact. However, indeed it is not required to implement every software construct in an ontology and not every OWL construct is required to be mapped to a software...
construct. This is because some constructs are only required for reasoning purposes and some are only required for the software artifact. Identification of such constructs is important and not addressed extensively yet.

1.5.4 Ubiquitous Web Navigation

We investigate how the Semantic Web can enhance web navigation and accessibility by following a hybrid approach of document-oriented and data-oriented considerations (see article presented in Section 2.3 [96]). Precisely, we propose a methodology for specifying, extracting, and presenting semantic data embedded in (X)HTML documents with RDFa in order to enable and improve ubiquitous web navigation and accessibility for the end-users. In our context, embedded data does not only contain data type property annotations, but also object properties for interlinking, and embedded domain knowledge for enhanced content navigation through ontology reasoning. We provide a prototype implementation, called Semantic Web Component (SWC) and evaluate our methodology along a concrete scenario for mobile devices and with respect to precision, performance, network traffic, and usability. Evaluation results suggest that our approach decreases network traffic as well as the amount of information presented to a user without requiring significantly more processing time, and that it allows creating a satisfactory navigation experience. This work is particularly important for the widgetization of existing applications since it enables us re-forming a less complex version of the applications from their interfaces.

1.5.4.1 Approach and Methodology

The overall approach requires that, at the server-side, requests and the responses between the client and the server are observed. When a client initiates a semantic navigation request for a page of a website, semantically annotated information (i.e., embedded data) is filtered out instead of returning all the (X)HTML content. The extracted information is presented as an (X)HTML document (i.e., reduced content). Users navigate through the semantic information available in a website by following data links and relevant HTML links. The advantage of the current document-oriented web navigation is that each page contains conceptually related information. This enables the user to have an overview of the content, thereby increasing content and context awareness and control. However, the problem is that in each page the user is confronted with ample amounts of information. A purely a data-oriented approach has the advantage of enabling the user to iteratively filter the content in order to accesses the information of interest. However, applying a purely data-oriented approach to web navigation is problematic since: (1) in data-oriented approaches the navigation is highly sequential, consequently, long data chains, constructed through RDF links, can easily cause users to lose provenance and get lost, (2) embedded data available in different pages of a website does not necessarily need to be related or linked. In this context, purely data-oriented approaches are more suitable to expert users for specific purposes, like ontology navigation. We follow a hybrid approach merging document-oriented and data-oriented considerations. The hybrid approach gathers the benefits of both approaches: (1) by following HTML links a user can switch focus from one information cluster/sub-graph (i.e., webpage) to another at once, hence navigation
experience is not highly sequential, while content and context awareness and control are maintained, and (2) by following data links within a webpage, the users can access information of interest through iteratively filtering the content rather than being confronted with abundant information. A server-sided mechanism is preferred in order to isolate end-user devices from computational load of the extraction; however, the approach is not based on a central service, but rather on modules and filters for application servers.

**Figure 1.4**: An example site consisting of three annotated HTML pages.

We first compare different embedded semantics technologies, namely RDFa, microdata, microformats, eRDF (cf. [81]), for suitability to our approach in terms of independence and extensibility, DRY (Don’t repeat yourself), locality, and self-containment as stated in [81]. The comparison suggests that RDFa and microdata addresses the aforementioned requirements where their famous counter microformats lacks independence and extensibility as well as implicit knowledge representation and data interlinking. Extraction mechanism might be also of crucial for low-power client devices; in our case a distributed server-side extraction mechanism is promising rather than a purely client side mechanism (e.g., [97]) which uses the client’s resources.

We describe our methodology in three phases, namely: (1) document preparation, (2) extraction, reasoning and presentation, and (3) architecture. Regarding the document preparation, we provide a three level specification for in-content annotation in order to enable the exploitation of embedded semantic data for human consumption. Figure 1.4 provides an example site with three pages along ontological classes and properties embedded in each page while Figure 1.5 shows HTML content of an example annotated HTML page.

First level is metadata level and specifies how embedded data instances should be enriched with human understandable textual and visual elements along naming conventions. Typical data browsers usually present data by using original type,
property, and relationship identifiers as it appears in the corresponding vocabulary or ontology; such an approach is not appropriate for naive end-user consumption. The second level is domain knowledge level which specifies how additional domain knowledge with higher level semantics can enhance the navigation. We particularly address how class classifications can be used to make navigation experience less linear together with object relationships.

Figure 1.5: A fragment of an annotated HTML page.

The third level is navigation level, it specifies how navigation hierarchy can be fine tuned and tailored for the end-users at the document preparation phase. In a navigation experience generated by the proposed approach, data hierarchy is constructed through object relations and class-subclass relationships while document hierarchy is constructed through HTML links. Users are already familiar with a document-oriented navigation experience; however, in a data-oriented approach users might encounter unfamiliar situations to which only experienced users can give meaning. At his level, we specify such cases which can be addressed at document preparation phase. Regarding the extraction, reasoning and presentation phase, extraction and reasoning processes are conducted at the server side while the presentation related process can either take place at the client (through JavaScript calls to the server) or at the server side (through HTML links). We specify how annotated document should be extracted, accessed and presented. We specify a set of heuristics in order to tailor the navigation experience to the end-users at the document
processing stage. In typical semantic data browsing, navigational chains are usually longer; however, for end-user navigation, such chains can be shortened. This becomes necessary in order to prevent end-user confusion.

We finally propose an architecture based on application server modules to enable web servers directly serve semantic data with respect to our approach. The architecture consists of three modules, namely Mod semantic, Mod GRDDL, and Mod SWC. The first module is responsible for extracting contextual information from the request header. It detects the device type or extracts an explicit semantic navigation request from the request header encoded with a specified parameter. The second module is responsible for extracting embedded semantic data from the (X)HTML. It stores the data, once extracted, to the session store temporarily, in RDF form, during the client’s session life time for performance considerations. If inference over extracted data is demanded, it applies ontological reasoning and stores the inferred data-set separately. The third module is responsible for preparing and maintaining the state of the presentation. It detects the state of navigation (i.e., the active navigation level) and extracts the requested navigation level.

1.5.4.2 Evaluation

We evaluate our approach at four levels. First, we provide a prototype, named SWC, to prove its feasibility. An example is shown at Figure 1.6 for the products page.

![Figure 1.6: An example demonstrating access to an annotated HTML page through SWC.](image)

Secondly, we measure the performance of our prototype; this is particularly important since extraction and reasoning are time expensive processes. We measure the performance our approach with and without reasoning. Results suggest that the proposed approach is feasible from the performance point of view. Thirdly, we measure the network efficiency of our approach by two metrics, namely precision and number of requests. Precision is defined as the fraction of the size of retrieved data that are relevant to the user's information need (i.e., target instance), and number of requests refers to the total number of HTTP calls required to access the target instance. The results suggest that the proposed approach is better in terms precision, yet it requires higher number of requests. However, the increase in amount of network calls seems admissible since the amount of information downloaded in each request is
considerably small. Finally, we conduct a usability study to test (1) whether our semantic approach can create a satisfactory navigation experience comparable/superior to the normal navigation, (2) to find directions for more heuristics, and (3) to detect any major usability problems. We set up a think-aloud test scenario and gave a set of tasks to the test users. Our usability study shows that the proposed approach can generate a navigation experience, comparable to the normal navigation experience, and does not inherit any major usability problems. We derive a set of directions for additional heuristics. We also derive metrics named observed precision and efficiency to enable content organizers to measure effectiveness of their content organization. Observed precision accounts the unexpected navigational levels that a user follows during a targeted navigation. Efficiency is the ratio of expected precision to observed precision.

At the moment our approach misses annotation of interactional elements (e.g., HTML) forms; however this is resolved in the following study (see Section 1.5.5) for which we provided a mechanism to annotate HTML forms for the end-user data mobility facility.

### 1.5.5 Widget-based Personal Environments

Mashups have been studied extensively in the literature; nevertheless, the large body of work in this area focuses on service/data level integration and leaves UI level integration, hence UI mashups, almost unexplored. The latter generates digital environments in which participating sources exist as individual entities; member applications and data sources share the same graphical space particularly in the form of widgets. However, the true integration can only be realized through enabling widgets to be responsive to the events happening in each other. We call such an integration widget orchestration and the resulting application mashup by orchestration. We aim to explore and address challenges regarding the realization of widget-based UI mashups and UI level integration, prominently in terms of widget orchestration, and to assess their suitability for building web-based personal environments (see article presented in Section 2.4 [98]). We provide a holistic view on mashups and a theoretical grounding for widget-based personal environments. We identify the following challenges: widget interoperability, end-user data mobility as a basis for manual widget orchestration, user behavior mining - for extracting behavioral patterns - as a basis for automated widget orchestration, and infrastructure. We introduce functional widget interfaces for application interoperability, exploit semantic web technologies for data interoperability, and realize end-user data mobility on top of this interoperability framework. We employ semantically enhanced workflow/process mining techniques, along with Petri nets as a formal ground, for user behavior mining. We outline a reference platform and architecture, compliant with our strategies, and extend W3C widget specification respectively - prominently with a communication channel - to foster standardization. We evaluate our solution approaches regarding interoperability and infrastructure through a qualitative comparison with respect to existing literature, and we provide a computational evaluation of our behavior mining approach. We have implemented a prototype for a widget-based personal learning environment for foreign language learning to demonstrate the feasibility of our solution strategies. The prototype is also used as a
basis for the end-user assessment of widget-based personal environments and widget orchestration. Evaluation results suggest that our interoperability framework, platform, and architecture have certain advantages over the existing approaches and proposed behavior mining techniques are adequate for the extraction of behavioral patterns. User assessments show that widget-based UI mashups with orchestration (i.e., mashups by orchestration) are promising for the creation of personal environments as well as for an enhanced user experience.

1.5.5.1 Approach and Methodology

In our approach, each widget notifies the platform, through a communication channel, whenever a user action occurs, including data exchanges. The platform stores events into the event log and monitors the log for a certain time to extract behavioral patterns. A behavioral pattern is a partial workflow with a flow structure and routing criteria. We first introduce functional interfaces (FWI), which allow widgets to disclose their functionalities, so that the platform can automatically execute the extracted patterns. FWI addresses the application interoperability challenge. In other words, the platform simply tries to re-generate corresponding events in the associated widgets when a particular pattern is detected. Therefore, each function of FWI corresponds to a user action within a widget that generates an event when triggered.

We annotate event signatures, functional interfaces, and widget content including interactional elements (e.g., forms) with the domain knowledge and semantics by using RDFa to address the data interoperability challenge. This also enables us to exploit reasoning power of the ontologies for data matching and mining behavioral patterns. We build an end-user data mobility facility on top of this interoperability framework. We provide a plugin which visually marks annotated content pieces, i.e., data pieces and HTML forms, and associate them with specific events to enable end-users to copy data from one widget to another with simple clicks. We specify a technique that allows us to match form elements with the user-selected data chunk through transforming the HTML forms into a SPARQL query and executing it over the end-user selected data piece. We propose a reference platform and architecture for widget-based personal environments. The platform consists of a run-time system and backend system. The backend system resides at the server side and is responsible for the persistence and decision-making; each consists of set of components. We detail standardized components of the platform; those are mainly a communication channel based on HTML 5, a messaging format and its specification (for event delivery, access to platform services, and orchestration control commands), and necessary extensions to W3C’s widget specification. We provide platform services (e.g., preference, data access etc.) to the widgets through communication channel, which is also used for event delivery and can be used for inter-widget communication for other orchestration approaches. The platform and the extensions, which we propose for W3C’s widget specification, are generic enough to accommodate other orchestration approaches. Finally, we address the behavior mining challenge.

We build our system-driven orchestration on two possible scenarios. The first one is that two or more widgets can consume the same input data, suggesting that these widgets can run in parallel. The second one is that one or more widgets can consume the output of another widget, suggesting that the consuming widgets are sequential to the source widget and parallel to each other. We mine such patterns from the user log.
The output-input match is special, since the log regarding this scenario is generated through the end-user data mobility facility. We investigate workflow mining methods and techniques for mining behavioral patterns and their topology (cf. [99]) and decision mining methods and techniques for mining the routing criteria (cf. [100]) when there exist alternative paths in a pattern. We employ Colored Petri nets (cf. [92]) to represent, share and validate mined behavioral patterns. We compare our problem with traditional workflow mining and identify the differences in order to develop appropriate methods and techniques. Prominently, in workflow mining, there exists a complete workflow while in our approach we only have fragments. We limit our patterns to OR, XOR and AND with one source action and two target actions having either OR, AND, or XOR relationships. We limit number of follower widget actions to two, since excessive number of automated actions might cause increased cognitive load for the end-users. We use a variation of frequency analysis used in well-known α-algorithm [101] to detect most frequent two follower widget actions for each action. We specify how the log file can be processed since the generated log file is different than the ones generated for a workflow. We employ multi-label classification to detect decision criteria by employing a variation of problem transformation approach based on label combination [102]. We transform generated decision tree into set of rules and commit them to the widget platform for automation. We provide a facility to move interplaying widgets closer, since literature shows that this has a positive effect on the end-users [103].

1.5.5.2 Evaluation

We first evaluate our approach by providing a prototype, for language learning, to prove the feasibility of approach. Secondly, we compare different orchestration approaches (user-driven, design-driven, distributed, system-driven, hybrid) with respect to a set of criteria, namely demand driven, open, loosely coupled, clustered, simple (orchestration), effortless (orchestration), sound (orchestration), and autonomous (orchestration). We define each criterion explicitly. The results favor for a system-driven approach and hybrid approach where there is a tradeoff between these two approaches in terms of soundness and simplicity of the orchestration. Thirdly, we provide an analysis of our pattern and decision mining approach with respect to certain criteria, namely label cardinality and label density (cf. [104]). The results show that our decision mining approach is promising and is not supposed to suffer from low data density. We conduct user experiments to test usability of our system-driven approach and prototype along performance of our mining approach. Our first prototype is a widget-based personal learning environment (WIPLE). We design a set of tasks that require users to use the environment and widgets to comprehend set of words in a foreign language in three different sessions. We acquire training and test data with the first two sessions respectively. We measure the performance of our approach in terms of Hamming Loss, precision, recall and accuracy. The results reveal that our mining approach is promising. In the final session, we conduct a think-aloud session and ask end-users to evaluate the mashup idea, our prototype, data mobility facility, orchestration, and widget relocation. The results are positive for each aspect, and show directions to improve our prototype. We have several observations. One of the prominent ones is that, even if there is a semantic match between two widgets, the users might not opt for an interplay due to
various reasons (dislikes, the complexity of offered content etc.) which supports our claim that a semantic match does not necessarily mean a sound orchestration.

1.6 Structure of the Text

The remainder of this thesis is organized into the following chapters:

Chapter 2: Selection of Published Articles

This chapter consists of four publications. The links and content of each publication are already presented in Section 1.4.3 and Section 1.5.


The first article, presented in Section 2.1, is a review article and addresses Pervasive Computing, its links with Adaptive Systems, main concepts and notions, development and management issues, and main open challenges from a skeptical perspective. The review mainly suggests a unified use of ontologies for development-time and run-time and the need for end-user involvement. It draws a perspective where digital and physical entities (with digital presences) construct a unified ecosystem situated on the human layer of the earth and the digital space. The Web is considered to be main communication, application, and information space. The aggregation and cooperation of distributed entities for the creation of adaptive and (personal) pervasive environments is considered to be a crucial direction.

The second article, presented in Section 2.2, is a review article. With the light shed by the first review, it reviews the end-user considerations of adaptive and pervasive
systems and identifies necessary concepts, such as intelligibility, end-user involvement, end-user situation awareness etc., for successful realization of such systems. It elaborates on the main challenges regarding the development of user-centric adaptive and pervasive systems and explores how approaches based on high level semantics, particularly ontologies, can address these challenges. It proposes several approaches, at a conceptual level, that can make use ontologies both at development and run-time while addressing end-user considerations. It surveys related practical work, from KR and Logic perspective, which can be integrated to realize proposed approaches.

The third study, presented in Section 2.3, represents the first part of our practical contribution. In this study, with respect to the approach based on high level abstractions, presented in Section 2.1 and Section 2.2, we aim at harvesting semantically annotated content from the interfaces of the web applications and to regenerate simpler versions of the applications, particularly for ubiquitous web navigation. We review the existing embedded semantics technologies with respect to proposed approach and specify the annotation, extraction, and presentation mechanisms. We also propose a set of heuristics and methods, exploiting domain semantics, for fine tuning and facilitating the end-user consumption. We implement a prototype, named SWC, to prove the feasibility of our approach based on a mobile scenario. We test the usability of our approach with an end-user study. We also test the computational feasibility of our approach in terms of extraction and reasoning performance. The results suggest that proposed approach is feasible and of use for the ubiquitous web navigation. The proposed approach addresses the widgetization challenge.

The fourth study, presented in Section 2.4, is the final part of our practical contribution. In this study, we approach the end-user consideration in terms of personal and pervasive environments. We analyze the characteristics of personal environments and how widget-based user interface mashups can address the creation of web-based personal environments. We consider widgets as an encapsulation medium for traditional applications and physical appliances. We aim at enabling end-users to aggregate distributed entities and orchestrate them to generate their own experiences for their own needs. We provide an interoperability framework, an open platform, a reference architecture, a facility for user-driven orchestration, an approach for system-driven automated orchestration based on behavioral user patterns. The interoperability framework and the platform make an extensive use of ontologies and semantics to meet the goals. We provide a prototype for a widget-based personal learning environment, named WIPLE, to prove the feasibility of our approaches. We conduct end-user studies to test the usability of our platform. We test the computational feasibility and efficiency of our pattern mining approach through the data gathered during the end-user study. The results suggest that our approaches and methods are promising for the creation of personal environments with manual and automated orchestration support.

**Chapter 3: Conclusions and Future Work**

Chapter 3 summarizes the conclusions of the work presented in this thesis as well as our main contributions, and directions for future research.
Introduction
Chapter 2

Selection of Published Articles

This chapter collects the following list of internationally published articles:


4. Mashups by Orchestration and Widget-based Personal Environments: Key Challenges, Solution Strategies, and an Application.
2.1 Context and Adaptivity in Pervasive Computing Environments: Links with Software Engineering and Ontological Engineering

Authors: Ahmet Soylu, Patrick De Causmaecker, and Piet Desmet


I am the first author and only PhD student in the corresponding article. I am the main responsible for its realizations. The co-authors provided mentoring support for the development of the main ideas.

Earlier versions were published in:


Context and Adaptivity in Pervasive Computing Environments: Links with Software Engineering and Ontological Engineering

Ahmet Soylu\(^1,2\), Patrick De Causmaecker\(^1,2\), and Piet Desmet\(^1\)

\(^1\) KU Leuven, Interdisciplinary Research on Technology Education and Communication (iTec), Kortrijk, Belgium
\(^2\) KU Leuven, Combinatorial Optimization and Decision Support (CODeS), Kortrijk, Belgium

In this article we present a review of selected literature of context-aware pervasive computing while integrating theory and practice from various disciplines in order to construct a theoretical grounding and a technical follow-up path for our future research. This paper is not meant to provide an extensive review of the literature, but rather to integrate and extend fundamental and promising theoretical and technical aspects found in the literature. Our purpose is to use the constructed theory and practice in order to enable anywhere and anytime adaptive e-learning environments. We particularly elaborate on context, adaptivity, context-aware systems, ontologies and software development issues. Furthermore, we represent our viewpoint for context-aware pervasive application development particularly based on higher abstraction where ontologies and semantic web activities, also the Web itself, are of crucial.

1 Introduction

*Machines that fit the human environment instead of forcing humans to enter theirs will make using a computer as refreshing as taking a walk in the woods* [1, 2].

Computing has already dispersed from dedicated and stationary computing units into the user environment and presently we are surrounded with mobile, multimodal and multiuser computing devices. [3] notes that pervasive computing (a.k.a. ubiquitous computing, ambient intelligence) takes advantage of distributed computing and mobile computing while inheriting problems (e.g. remote access, high availability, power management, mobile information access) in these fields increasingly. Apart from these problems, since they have been studied under related domains effectively, it is reasonable to say that we already achieved a lot as a part of Weiser’s vision in the sense of hardware and network technologies by considering the advancements in the networking technologies, computing power, miniaturization, energy consumption, materials, sensors etc. [4]. However we are still far from the complete puzzle, pervasive computing is not just about developing such small computing residents for the real life, variety of applications exploiting such extended hardware infrastructure are the other side of the coin. Spreading computing all over
life imposes new challenges which were already foreseen in this vision. Anywhere and anytime computing needs to cope with computing devices which are mobile, users which are mobile and software applications which are mobile. [5] partly referred to this mobility as “constantly changing execution environment”; we rather call it “constantly changing computing setting” which refers to mobility and dynamism of both related parties. Furthermore, heterogeneity of such environments hardens the challenges of such vision since soft-ware and hardware markets have already been populated with variety of applications and tools coming from different vendors.

Does this increasing digitization of life require more attention of people? This question, which originates from mobility and dynamism, requires achievement of the following approach:

The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it [1].

In other words seamless integration of computing into people’s life is a must of the pervasive computing vision. If we don’t want people to bother about the computing devices and the applications surrounding them, while they are making use of them, we need to make computing devices and applications to bother about people. Utilizing all these physical resources and synchronizing various applications available through this extended dynamic infrastructure for the benefit of the users requires an “intelligence” behind. This implies that computing systems need to reach a level of understanding of the settings in which they are being used, and the complex relations between the various elements of these settings. This ability is called “Perception”, however this is one side of the coin. On the other hand computer systems need to be able to exploit this understanding by adapting their behaviors accordingly (i.e. that is to response properly according to perceived context), which is called “Adaptivity”. These two interrelated challenges make pervasive computing diverge from mobile computing and distributed computing since the challenges needed to cope with are not strictly bound with these fields. Besides they are rather new and their theoretical grounding is not yet sufficiently mature. Moreover autonomous applications in such environments need to operate collectively in order to achieve maximum utility (at least to ensure a conflict free execution), however the heterogeneity of the pervasive environments hinders seamless integration of different applications and devices, that is interoperability. Standard compliance is of prominent importance for such a requirement since standards help to ensure interoperability and five other important abilities: (1) re-usability, (2) manageability, (3) accessibility, (4) durability [6]. All in all, we define pervasive computing environments as follows:

...intelligent digital ecosystems which are seamlessly situated in user's physical environment. Such ecosystem is defined as a collection of seamlessly integrated, mobile/stationary and autonomous/non-autonomous devices and applications, where higher mobility and autonomy is of crucial. Intelligence for such systems is defined as capability of being able to perceive changing computing context and to response collectively in a proper manner (i.e. to adapt) for maximum user utility.

Accordingly, we consider perception, adaptivity, interoperability and standard compliance as key enablers of pervasive computing apart from other technical challenges, inherited from aforementioned fields, and social challenges (i.e. privacy, trust and security). Our main research is about enabling any-where and any-time adaptive learning environments which is highly dependent on pervasive computing
vision. Despite the fact that this paper is based on a domain specific (i.e. e-learning) perspective in order to enable pervasive e-learning, the solutions and approaches we do aim to follow and propose are rather generic. In this paper, our contribution can be grouped under two categories: (1) theoretical, (2) technical. From theoretical point of view, we do extract and extend theoretical aspects found in the existing literature, and we work toward to integrate these ideas into a common understanding for the future pervasive computing systems. Regarding technical point of view, we review the related literature with the purpose of integrating and extending existing technical work which are generic, standard-based, and compliant with the overall understanding which we synthesized. Readers should bear in mind that our purpose is not to give an exhaustive review of the literature, but merely to provide a selective and integrative review of comparatively important and promising approaches found in the literature. Our selection criteria is particularly based on following parameters: (1) standard-compliance (although limitations of the available standards might hinder our efforts, since available standards are based on the characteristics of traditional computing, keeping available standards in the core of the development and research and to extend them when required is our guiding mantra), (2) generalness, (3) applicability, (4) simplicity, (5) ease of development, (6) extensibility, and (7) scalability.

The remainder of this paper is structured as follows: in Section 2, we introduce methodology and domain specific motivation of our research. We elaborate on the notion of context and its relation with adaptivity in Section 3, we further refer to characteristics and categorization of context in respective subsections. In Section 4, categorization of context-aware systems is briefly referred while context management is elaborated in Section 5. We further investigate some key problems and basic solution approaches in Section 6. In Section 7, we introduce our viewpoint for context-aware application development based on model driven and ontology driven approaches by referring related literature, we also emphasize use of World Wide Web as an information source for pervasive environments. Finally we conclude this paper in Section 8.

2 Motivation and Methodology

Challenges which are based on natural characteristics of pervasive computing systems (i.e. mobility, dynamism and heterogeneity) can be evaluated from a more domain specific perspective, that is, e-learning in our case. E-learning refers learning which uses variety of technologies such as internet, television etc. in a manner pointed out by [7]:

...e-enhancements of models of learning. That is to say that; using technology to achieve better learning outcomes, or a more effective assessment of these outcomes, or a more cost-efficient way of bringing learning environment to the learners [7].

E-learning evolved a lot by the emergence of computers and later internet, and continues its raise with the advancements in network and mobile services and software market which offers variety of advanced learning environments, tools and adaptive technologies. Apart from technological advancements, e-learning also faced with some important pedagogical movements particularly learner centric and self
directed approaches which are based on constructivist learning theories. These approaches consider learners as active participants of the learning instead of passive consumers and change the role of teachers as facilitators who assist learners to clarify their goals and enable them to be capable of planning, executing and evaluating their learning progress and outcomes collaboratively, without taking a particular position in the discussions, rather than being pure source of information [8, 9]. [10] notes that providing active, stimulating, authentic learning experiences that support learner collaboration, construction and reflection is major challenge for success of e-learning. Such approaches triggered the creation of learner-centric, social and collaborative learning environments. Today embedding social networking and collaboration into learning progress is considered as driving force for learner’s motivation and activity [11]. Moreover social software (e.g. blogs, wikis etc.) gained an important place for e-learning thus the mine of data, World Wide Web, because of Web 2.0’s great collaborative potential, Wisdom of Crowds, and simple find-remix and share rule. As a consequence, e-learning market has already been over populated with such tools and platforms to support different types of learning communities with learning management, content management and communication tools [9]. Learners are not bound to neither individual learning environments, as closed box of pure information, nor to classical in-class learning environments anymore. Instead by the guidance of the constructivist theories they are facing with variety of tools including their particular learning environments which enables them to collaborate, to reach endless amount of information of the Web, and to remix-share it, thus also to create social networks. Depending on the case, these tools are being used individually by learners, or by means of mash-ups, or as heterogeneous systems which involve several tools and might be centered around a particular learning system [12]. Furthermore, with the emergence of pervasive computing vision learners and the learning process also goes for time, place and device independence, that is, learn anywhere-anytime. Pervasive learning goes hand-in-hand with the idea of “always on” education and extends concepts of collaborative learning, cooperative learning, constructivism, information rich learning environments, self-organized learning, adaptive learning, multimodal learning, and a myriad of other learning theories [13]. Growing tool and device landscape and the pervasive computing vision, forces e-learning domain to adjust itself within this new landscape appropriately. Therefore there is also a line of research towards pervasive learning (a.k.a. ubiquitous learning) where a pervasive e-learning environment might be defined as a setting in which students can become totally immersed in the learning process [14]. Pervasive computing takes part in an experience of immersion as a mediator between the learner’s mental (e.g. needs, preferences, prior knowledge), physical (e.g. objects, other learners close by) and virtual (e.g. content accessible with mobile devices, artifacts) contexts [15]. We work towards enabling different applications in such learning environments to be seamlessly integrated (i.e. to be interoperable) and to be aware of the setting which they are used and to collectively adapt their behaviors according to the available context information. Enabling computing settings where capabilities, requirement and characteristic of entities are known to each other decouples these entities, that is, independence which is required for mediation process, which is adaptation. Hence, we particularly list following basic interrelated requirements for such pervasive learning environments: (1) device independence: applications and data should be
always accessible without any device dependence, (2) application independence: data should be always accessible without any application dependence, (3) adaptivity and adaptability: learning environment and elements of this environment should dynamically adapt according to context of learner(s) and users should be able to configure such environments such as composing/decomposing data and applications, (4) collective operation: applications in such environments must be able to collectively operate for the benefit of users in a seamless manner. Adaptivity is long studied both in adaptive web systems and adaptive e-learning systems [16], and in such systems adaptivity is generally considered as an aspect between user and application based on user profiles and models. However, although we do follow a user-centric approach, other requirements (1,2 and 4) make it necessary to broaden the adaptivity from learners to the whole environment in which user is engaged in order to be able to mediate between different independent entities of such settings. Although we do not claim to propose solutions for all the challenges of pervasive computing or pervasive learning, the approaches which we propose are common enough to be employed within generic pervasive environments. That is only possible by first providing a generic understanding (i.e. theory). Briefly our research question can be formulated as follows:

**How to enable adaptivity (in broader sense) in Pervasive Learning Environments through applying available context information?**

Accordingly, our main approach is to integrate and extend available technical and theoretic approaches in pervasive computing, context-aware computing and adaptive systems literature into e-learning. In this stage we mainly focus on constructing a theory which represents overall framework of our understanding and to which our future practice should comply with. The theory that we focus on is broad while the practice is limited in the scope (i.e. e-learning) which is based on constructed theory. Therefore many of the challenges introduced in this paper are either in our long term agenda or merely mentioned for the attention of other researchers. Challenges specific to our main research is subject to another publication. The overall approach is depicted in Figure 1, the lower domains are much more generic and theory intensive in order to constitute overall frame of our research. The upper domains are more specific and dependent on lower domains, innovative aspects of the research increases towards specific domains while integrative aspects are higher in more generic domains. In this stage of our work, we mostly focus on the theoretical and technical
aspects of context-aware pervasive computing and adaptivity (in a generic sense) in such environments, that is, first two levels of our research pie.

Adaptivity (in a more specific sense) and e-learning is subject to another in depth research where specifics of the domain and existing work (i.e. e-learning, adaptive e-learning) need to be elaborated based on the theory and practice introduced in this paper.

3 Context and Adaptivity

The notion of context is of crucial for pervasive computing systems, it is a central notion for context-aware pervasive computing environments as we already mentioned in Section 1. Indeed, according to the view represented previously, pervasiveness, context-awareness and adaptivity are bound to each other, that is, one implies the other one. The notion of context has, over time, been extensively discussed in the literature [17, 18, 19, 20, 21, 22]. [23] reviews related work and after briefly criticizing the concept, author gives the well known definition of context:

`Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between the user and application, including the user and applications themselves [23].`

Previous definitions of context in the literature usually refer to context as location, identity of users, and nearby people. Intuitively it is reasonable to accept location, identity, activity and time [18, 23] as important elements of context. However these elements are not sufficiently broad to cover the notion. The definition given in [23] is more generic and open-ended and covers context as a whole. The reason why it is not possible to give a more specific definition is the openness of the notion of context; a particular knowledge is considered to be context information in one setting while it is not part of context in another setting. [24] points out that it is not always possible to enumerate a priori a limited set of context that matches the real world context and [25] also refers to the same issue by pointing out that it is not possible to enumerate all important aspects of a situation. Therefore, by following the definition of [23], we are lead to conclude that defining the scope for context should leave an important role for context-aware application development rather than providing an exhaustive definition of context.

Since it is not possible to predefine all the dimensions of context, then how can we decide whether a piece of information can be counted as context or not? [26] remarks that context of use will have a substantial impact on the appropriate behavior of applications, without being a primary input source. Well, then imagine an automatic door which uses a sensor to detect presence of a person in front for switching between its states (i.e. close or open). Location of the user is indeed primary input for this particular system, and obviously the application for this system is primarily designed to sense the situation (i.e. presence of a person) and to act accordingly. Several example applications can be listed where context information is used to adapt application behavior without being primary input of the application in contrast to the previous example. Then, should we also consider primary inputs of applications as
context information? Or should we only consider context as the information which is not primary input of the application but which characterize the situation? Here is another example: consider a word predictor application for speaking-impaired people [27]. This application can use previous user inputs to predict the word which the user presently tries to type. Here primary input of a previous context turns out to be another context dimension. Indeed every application, whether we consider it context aware or not, is designed for a specific and restricted context of use. Therefore these applications provide a particular set of behaviors for a fixed context of use. Hence, we are lead to conclude that context awareness is ultimately related with adaptivity. It is based on exploiting recruited context information and to adapt its behavior accordingly. In order to consider a piece of information to be context, it has to be ensured that this piece of information enables the corresponding application to modify its behaviors with respect to this piece of information and its relation with other context dimensions. [28] states:

[...] something is context because of the way it is used in interpretation, not due to its inherent properties. The voltage on the power lines is a context if there is some action by the user and/or computer whose interpretation is dependent on it, but otherwise is just part of the environment [28].

The mostly used context dimension is location, however it is a known fact that context is not limited to the location and physical objects in the environment. Consider the field of the Adaptive Web [16]. Much work has been done to define user models (e.g. user knowledge, goals etc.) and user profiles (e.g. user interests etc.) in order to enable web applications to act accordingly. Such adaptivity includes adaptive presentation, adaptive information filtering etc.. User profiles and user models are also type of context which are abstracted to a higher level mainly from logs of applications with which users interact. This implies that context information does not necessarily require to be gathered by sensors. Finger prints of users (i.e. application logs, web logs etc.) collected by applications can also be exploited to reach high level context information. Adaptive web applications belong to the field of Adaptive Systems and indeed they eventually are useful in the field of context-aware and pervasive computing systems. This relation implies that most of the practice and knowledge constructed in this field may be applied to the context-aware systems. Eventually it is the application which needs to adapt. Applications primarily need to adapt to the user, however the environment that the user lives in and the devices that are in contact with the user in turn influence the user. Applications adapt to the user as wells as to the environment, the devices and the complex relationships among each other. This is the result of mobility and dynamism of aforementioned peers. In an arbitrary setting where each device has its own characteristics and resources like screen size, CPU power, memory size, available input and output devices etc., applications need to adapt according to the context of the devices (i.e. resource awareness) to better serve the users [29, 30]. Adaptivity should not be understood as a one-to-one relation between user and application, in a pervasive computing setting, rather it should be considered as a relation between application and other elements of such settings (e.g. devices, physical environment, users etc.). Pervasive computing considered to be the third wave in the computing where first wave is main frame computing – one computer for many users-, second wave is personal computing – one computer per user -, and third wave is the one where many computers available for
one user. Indeed the later wave (i.e. pervasive computing) should be considered more broadly, that is, many computers for many users. That makes computing much more sophisticated from application development point of view, since applications are not only required to accommodate needs of only one user but of many users, that is, to adapt masses.

As a conclusion, context is an open concept since it is not limited with one’s imagination. Any system that exploits available context information needs to define the scope for the context. Adaptivity is the primary relation between computing and context, and to count any information as context we need such a relation. Any system can focus on any context category (in particular to the user(s)). However, we need to be aware of that the application needs to adapt to various context dimensions although it also has its own context dimensions.

3.1 Characteristics of the Context

In the previous section instead of focusing on the definition of context, we rather tried to comment on context from different perspectives to give a deeper insight into the notion. We now investigate some specific characteristics.

First of all, context is “dynamic” [24, 31, 32, 33]. Although some context dimensions are static like the name of a user, most of the context dimensions are highly dynamic like the location of a user. Furthermore, some context dimensions change more frequently than others. One dimension may change its state every second while another dimension only changes its state every year, this also implies that context is temporal [33, 34]. What is more important to see is the evolving nature of context, i.e. it is “dynamically constructed” [32]. Consider user knowledge: it evolves dynamically over time, i.e. user adds new knowledge pieces to his knowledge or some knowledge is forgotten. These changes in state do not require destruction of previous states, but the states evolve. Therefore [32] suggests not to support a particular context but to support the evolution of context:

[…] not to use of predefined context within ubiquitous computing system, but rather how can ubiquitous computing support the process by which context is continually manifest, defined, negotiated, and shared [32].

It is intuitively evident that several context dimensions are somehow interrelated [32, 33], that is, context is “relational”. For instance, there are different kinds of relations between people in your home and in your job. Your being at home or in office is normally related with present time. Perception is not just about realizing concepts but also about understanding relations between these concepts which are necessary to interpret situations and behaviors. Relationships between context dimensions thus hold an important place for both context representation and interpretation.

[35] points out that computational systems are good at gathering and aggregating data and humans are good at recognizing contexts and determining what is appropriate. The computer system level of understanding and recognition is limited, hence computer systems are far from recognizing situations properly. Besides, it is a known fact that even human beings sometimes are unable to understand/evaluate the exact situation. That is what we call misunderstanding. Hence even for a given well
modeled closed domain (i.e. a closed set of real world data), a computer system might lack proper perception. This is related to imperfection of context information, that is, context is “imperfect”: ambiguity, irrelevance, impreciseness and incompleteness of context dimensions [33, 34, 36]. Consider the context information acquired via sensors. It is a known fact that sensors do not provide hundred percent of accuracy. Besides, multiple sensors might provide different readings for the same context value. How can one really judge a student’s knowledge based on his answers to a multiple choice exam? Can one logically decide that it is night by simply considering the light level?

3.2 Categorization of the Context

It is possible to categorize context in various ways by considering different characteristics of the context. These categorizations are useful both for application development and for understanding of the context. [33] notes that classifying context is useful for managing quality of context, for instance dynamic context elements are prone to noise. Moreover such classifications are also useful for context modeling, in early conceptual phases and later, and they are required to define some specifics of adaptivity and context management (e.g. abstraction).

Acquired raw context information usually requires a certain level of abstraction which will be discussed briefly in Section 5. However, for a short insight, consider the example of location: a sensor might sense location as coordinates whereas the application might require this information in a more abstract way like the name of the city. Therefore location information based on coordinates requires to be abstracted in order to be comfortable with the application. Hence it is possible to categorize context from the application point of view [37] into (1) low level context information (a.k.a. implementation context) and (2) high level context information (a.k.a. application context). Low level context information is usually sensed by sensors or collected by means of application logs. [33] considers low level context information as environmental atomic facts. High level context information is derived from low level context information. However these are implicit means of collecting low level context information. It is also possible to gather context information explicitly, e.g. asking the user to provide context information directly. [33] suggests that the ideal case is placing fewer demands on user attention (i.e. less direct user interaction).

Context can also be categorized from the collection point of view [38] which is indeed related with the above categorization: (1) direct (sensed or defined), (2) indirect (by means of inferring from direct context). Direct context refers to the collection of context information without realizing any extra processing of the gathered information. If the information is gathered implicitly by means of sensors, it is called “sensed context”. If the information is gathered explicitly, it is called “defined context”. We already mentioned that sensors are not the only means of collecting context information, application logging is just another way to do so. Therefore we propose to further categorize sensed context as “sensor based” and “application based”. Direct context refers to low level context and indirect context refers to high level context information according to the previous categorization.
Context information can be categorized from a temporal point of view into two categories: (1) static context, and (2) dynamic context. Static context does not change by time like gender or name of a person. Dynamic context keeps changing in different frequencies depending on the context dimension like your location or age. This implies that for a dynamic context dimension, various values might be available. Hence, management of temporal character of context information is of crucial either in the sense of historical context or in the sense of validity of contextual information available.

Apart from categorizing context based on characteristics of the context, [5] categorizes context based on grouping similar context dimensions into: (1) computing context, (2) user context, and (3) physical context. Later [39] extends this categorization with (4) time context. [40] provides a similar context categorization; (1) physical context, (2) social context, and (3) internal context. [41] provides another categorization which includes (1) infrastructure context, (2) system context, (3) domain context, and (4) physical context. These categorizations are usually at higher granularity, hence they do not reveal enough information about themselves, and this might limit their usefulness for development of context-aware systems. Moreover some of them are more application oriented, hence the categorizations are not well balanced. We propose eight categories for context aware settings where we want to achieve an optimal granularity and want to represent main actors (i.e. entities) of a typical pervasive computing setting in a more real-world oriented manner. This categorization provides a clear layering for context-aware system development and may serve as an initial step toward a generic conceptualization. We argue for a layered categorization of context without considering any taxonomical relation: (1) user context (internal, external), (2) device context (hard, soft), (3) application context, (4) information context, (5) environmental context (physical, digital – e.g. network –), (6) time context, (7) historical context, (8) relational context. It is important to know what application is in use in which device, in which environmental setting, and at what time by which user etc.. Therefore context varies as a product of dimensions under disclosure of these context categories. User context splits into “external user context” and “internal user context”. External user context is easier to sense (e.g. name, gender, height, and weight etc.) while internal user context is harder to sense [40] (e.g. user feelings – hate, love etc. –). Internal context might be derived by interpreting diverse low level context information such as blood pressure, hormone levels etc. Considering the device context, we distinguish “hard device context” and “soft device context” where hard device context refers to the physical properties of the device (e.g. CPU, memory etc.) and soft device context refers to the available software components in the device etc.. Application context refers to capabilities and requirements of an application, e.g. target platform, memory requirements etc.. Concerning environmental context, we distinguish “physical environment context” and “digital environment context”. Physical environment context covers the real world entities and their characteristics such as nearby objects and their identities while digital context refers to the digital entities such as network capabilities. Information resides in digital space together with applications, context adaptive access of information is of crucial part of computing, particularly for the web environment. Hence information context refers to properties of meaningful information pieces available in different formats (e.g. text, image etc.), it is surprising
to see that information has not been considered as an independent entity either in available context categorizations or various context models in the literature (to the best of authors knowledge). Time usually refers to time of situation, time zone, part of the day etc.. Historical context refers to situations that occurred before based on the temporal characteristic of context. Relational context refers to relationships between the different context dimensions, that is, it aggregates and represents different types of relations between the elements of a particular context-aware setting. Although, relations have been used in context conceptualization, they have not be considered as a context entity explicitly, we advocate that it is worth to consider relations as an contextual information since they also characterize the situation of an entity. Historical context elements and relationships among context elements (this is relational context) are important for interpreting the situations. We previously mentioned that proposed categorization may serve as a generic conceptualization (i.e. upper ontology) for our future context model. In Figure 2 and Figure 3 a rough conceptualization is depicted with some possible immediate sub-entities.

![Diagram](image)

**Figure 2:** First part of the proposed upper context conceptualization, external state and internal state concepts has been shown as a part of user concept where user concept is part of environment.

[42] notes that generic uniform context models are more useful. Although there are already some proposals for a generic context models in the literature (see Section 5), our rough proposal provides clear advancements similar to previously mentioned context categorizations such as optimal granularity and balanced representation of actors. Secondly information has been shown as an independent entity, and as a main actor of context which has been omitted in previous conceptualizations and categorizations, importance of such approach is detailed in Section 7. This initial conceptualization only defines the borders of our understanding of context, a more elaborate formalized conceptualization (i.e. ontology) is to be developed where previous context models and standardized vocabularies are to be re-used.

We can further group aforementioned context categories into technical and non-technical context for the sake of separation of concerns. Non-technical context
includes context categories which are not related with the technical aspects such as internal user context, while the technical context involves context categories related with the technical aspects of context such as device context, and digital environment context. Although there is no straightforward way to distribute previous context categories into technical and non-technical context folds, non-technical context categories are mainly domain specific and require to be identified by domain experts, e.g. for pervasive learning environments, an expert is required to identify context categories or individual context dimensions related with learning aspects.

**Figure 3:** Second part of proposed upper context conceptualization, environment concept is composed of digital environment and physical environment concepts.

## 4 Context-aware Systems

In previous sections, context in pervasive computing has been reviewed. In what follows we elaborate on the definition of context-aware computing as it has been discussed in e.g. [17, 22, 23, 40, 43]. Earlier definitions usually involve a loose enumeration of context dimensions (e.g. location, nearby people etc.), and the later ones often concatenate on the relation between computing, context and user. It is clear that context needs to be employed to better serve the users, such point is already commonly noted in definitions like:

* [...] context aware computing] aims to enable device to provide better service for people through applying available context information [40].*

Above a generic definition of context aware computing is given, which emphasizes the relation between user, context and computing, but how do we apply available context information? Although various categorizations for context-aware systems are already given [5, 23, 43], we prefer to re-interpret these categorizations based on adaptive systems, particularly according to adaptive web systems. This is because we defined adaptivity as a key factor of intelligence and as a key relation between context and computing for context-aware computing systems. Therefore by referring to [5, 23,
and the field of adaptive web [16] for categorization of context-aware computing applications, we propose below categorization: (1) context based filtering and recommendation of information and services: examples might include finding the nearest printer, accessing the history of a nearby object etc., (2) context based presentation and access of information and services: e.g. selecting voice when screen displays are not available (multimodal information presentation and user interfaces), dynamic user interfaces etc., (3) context based information and service searching: e.g. location aware query rewriting for a search for available restaurants (query rewriting is a technique used in adaptive web systems for information filtering by rewriting a user query according to the user profiles) etc., (4) context adaptive navigation and task sequencing: adaptive navigation is a technique employed in adaptive web systems. We can extend this idea in pervasive computing since a user’s interaction might consist of several related sub-tasks in relation with his goals and might lead to context aware task sequencing, (5) context based service and application modification/configuration: this need mainly arises from different devices available in the environment, e.g. disabling particular features depending on the capabilities of target device, (6) context based actions: [44] proposes three levels of context dependent automatic actions: manual, semi-automatic, and automatic. [45] notes that fully automatic actions based on context are rarely useful, and incorrect actions can be frustrating, (7) context based resource allocation: this might include allocating physical recourses (e.g. memory, even non-hardware physical resources) for the use of other entities in the setting (e.g. applications, users etc.).

It is worth to note that, adaptive behaviors of context-aware systems are not necessarily need to depend on the current context, rather such systems should also be able to adapt proactively by making use of current context or historical context to predict future context of the setting. An example is given in [46] where a user walks through the building and submits a printing request, the selected printer should not depend on the user’s current location but rather to his final destination. According to presented categorizations and elaborations, we extend previous definition of context-aware systems as follows:

**Context aware computing aims to enable better service delivery through proactively adapting use, access, structure and behavior of information, services, applications and physical resources with respect to available context information.**

Above categorization also stresses the applicability of several techniques and methods in the field of the adaptive web as we already mentioned previously. Other interesting examples might be applications of collaborative filtering, mass adaptivity, case based adaptivity etc. in context aware systems. Collaborative filtering is the process of filtering or evaluating items using the opinions of other people [16]. Since pervasive computing systems are able to interact with different people in different context settings, they can use captured information for collaborative filtering, case based recommendation, and these systems can employ adaptivity for masses which are sharing common characteristics (e.g. understandings, behaviors etc.) in common pervasive computing settings. [47] is an example which provides recommendations by comparing users with other users in pervasive computing systems. Furthermore, a case based reasoning example is provided by [48], proposed methodology is to abstract raw context to user situation, to generate current user’s case, and to provide
adaptive behaviors by semantically comparing user’s current case with other previously stored cases and corresponding behaviors of the system.

As a final remark, pervasive computing environments do not necessarily fully automate their behaviors where such behaviors can be in varying granularity as shown in Figure 4 [49].

![Space representation of context based Adaptation Customization](image)

**Figure 4:** Space representation of context based Adaptation-Customization [49].

Such environments should also allow users to customize structure and behaviors of their environment (i.e. user control). Pervasive systems might facilitate such customization by means of adaptive guidance where environment does not automatically act or force user to one action but rather provides users with the required contextual information and recommendations. That is, adaptive behaviors do not necessarily need to result in “must”s or “have-to”s but in many cases also in “should”s and “might”s to give users a degree of control with the possible directions and their reasoning behind. In other words, in the scale of dynamic and static system adaptation, enabling users to control the environment does not imply that contextual information is useless for such a case. Rather, system can extend the limits of contextual information perceivable by the user’s physical capabilities by serving contextual information gathered by sensors to the users rather than automatically adapting itself. An up-to-date and specific example is a famous social networking website, Facebook. This web application provides users with the contextual information of their network (by means of notifications) like who watches, reads what or who becomes friend with whom. In this way users can identify people with similar likes and arrange their own environment accordingly. Such case is also of use in the domain of e-learning, a system can provide users with the contextual information of the environment and other learners like who read what, who knows what, who takes the same courses or who works on the same problem, so learners can find appropriate mentors or construct a learning path for themselves. Such approach might be called as “environment awareness” for users which is counterpart of context-awareness for machines.
5 Context Management

We identify following groups of components for context management infrastructure by adopting [50] and [51] as shown in Figure 5 which are required for realization of context-aware adaptation: (1) context modeling and representation, (2) context capturing (sensing), (3) context abstraction and reasoning, and (4) context dissemination (access and querying).

![Figure 5: Components of context management infrastructure: context modeling, context capturing, context reasoning, and context dissemination.](image)

Context capturing is handled by applications and physical sensors. [52] classifies sensors into following categories: (1) physical sensors which are hardware sensors available through physical environment to deliver physical measurements, (2) virtual sensors which are based on information and logs captured by the user applications and, (3) logical sensors which are based on reasoning various contextual information to produce higher level context information. Context dissemination is strictly related with the architecture of the context-aware application. Context information might be stored in central context brokers/blackboards, e.g. [42, 53, 54], or every application might hold its own contextual information, that is, context information might be distributed, e.g. [55, 56, 57]. Furthermore a hybrid approach might be possible where common contextual information is centralized and every application holds its specific contextual information. In all cases it is reasonable to call own-managed context information as “local context”, context information managed by other entities as “remote context”, and context information managed by central brokers as “central context” by extending the understanding presented in [58]. The most commonly used methods for context dissemination are push and pull mechanism [54, 59, 60, 61, 62]. In push mechanism applications register themselves to remote context entities or the central context brokers in order to be updated whenever a context of interest changes or is added. In pull mechanism, applications actively pool the remote and central context entities to check availability of context of interest, this might be possible by submitting synchronous or asynchronous query requests to the remote or central context entities. Within the same application, similar mechanisms can be employed, either by registered context listeners, e.g. [63], which triggers actions or asserts new contextual information, or by an ad-hoc manner where application itself checks the state of particular context information according to active execution stage.
Although distributed context management is researched by several users in the literature, e.g. [55, 56, 57], complexity and low efficiency of such approaches for the real-time systems do not seem to be promising yet. Resource-limited devices can hold their own contextual information, however even for limited amount contextual information reasoning can be time consuming for such devices (see Section 6) or even not possible according to available resources. Considering e-learning, e-learning environments are complex, and variety of contextual information might be of use, hence presently we would prefer to use context broker architecture where reasoning, privacy and security, dissemination of contextual information are handled by such central architectures. A promising example is given in [53]. Such approach is of great use for the real-time, reasoning intensive applications. Scalability issues might arise in such architecture, for such a case using several powerful context-brokers can be of immediate solution.

In the following sub sections we do elaborate on the context modeling and representation, and context abstraction and reasoning respectively.

5.1 Context Modeling and Representation

Applications become perceptive when they maintain the model of its occupants and activities and user is only willing to accept an intelligent environment offerings services implicitly if he understands and foresees its decisions [64]. Furthermore, [65] notes that it is hard to re-use and change context information embedded into functional modules. Today’s traditional intelligent computing is based on either ad-hoc AI techniques (e.g. data-mining, machine learning etc.), or based on hard-coded enumeration of possible contexts of use. However pervasive computing opens up infinite context space where it becomes hard to manage bindings between infinite context and behavior spaces (i.e. adaptive behaviors). Hence in order to enable computers to decide on (i.e. reason) adaptive actions (i.e. automatic, semi-automatic, manual) through automated reasoning and/or mediation processes - which requires to construct a bridge between humans and computers by enabling them to share a common world model - computing systems need to maintain a formal model of the settings in which they are being used and the complex relations between the various elements of these settings.

Several machine learning techniques (e.g. Bayesian networks, fuzzy logic etc. [50, 66]), statistical methods [67], and ontologies as an AI paradigm can be used to model contextual information. [36] analyses several approaches in the literature according to data scheme used and concludes that ontologies are promising for context modeling. They represent explicit, formal (i.e. machine understandable) and shared conceptualization of real world aspects [68]. [69] refers to several reasons in order to use ontologies for context modeling: (1) knowledge sharing, (2) logic inference, (3) knowledge re-use. Considering context representation based on ontologies, [70] lists the following requirements for context representation: (1) structured, (2) interchangeable, (3) composable / decomposable, (4) uniform, (5) extensible, (6) standardized. There are several techniques to represent ontologies. We adopt categorization provided in [71] into: (1) AI based, (2) software engineering (e.g. UML), e.g. [33], (3) database engineering (e.g. ER, EER), and (4) application
oriented techniques (e.g. key-value pairs), e.g. [31]. Software engineering techniques and database engineering techniques are limited in expressivity, i.e. they are not capable of expressing heavyweight ontologies (i.e. ontologies which model a domain with more constraints and expressiveness) but rather capable of modeling lightweight ontologies (i.e. ontologies which model a domain in a less expressive way and with less constraints). Indeed software engineering and database engineering are highly related with abstracting and modeling real world phenomena and logics into computer applications for a restricted context of use. This restriction causes software engineering and database engineering techniques to fall short when modeling generic context information. However it is not surprising to see that several software methodologies are well suited for ontology development (e.g. ontology re-engineering and software re-engineering [71]). AI based techniques are capable of representing high level ontologies, techniques based on frames and first order logic are mainly used. OWL (Web Ontology Language) [72], which provides a syntax and knowledge representation ontology, appeared as a prominent ontology formalization (i.e. representation) language with the advent of the semantic web. OWL is capable of representing main components of an ontology like classes (i.e. concepts), relations, instances and attributes. Since OWL is among the AI based techniques, it is suitable for high level ontologies. It can express complex relations between concepts, it is capable of acquiring dynamic information. Furthermore strong reasoning techniques and tools based on OWL provide a mean to deal with ambiguity in context. Hence it is reasonable to state that it is capable of capturing characteristics of context and criteria listed by [70]. There are already various works in the literature which employs ontologies, examples include [40, 43, 44], in order to maintain a context model and to apply reasoning over this model.

There are various tools and standards in the domain which supports ontology development and use based on OWL. We refer to prominent ones in what follows. Protégé provides a graphical interface to develop OWL based ontologies, JENA provides a semantic web framework where different ontology querying languages such as SPARQL and RDQL, and reasoning support are available. Semantic web rule languages such as RuleML and SWRL are already available and supported by various tools, which are used to describe logic rules.

5.2 Context Abstraction and Reasoning

We previously mentioned that context information is categorized as low level (i.e. implementation level) and high level (i.e. application level) context information. Low level context information is usually sensed by sensors or might be acquired from application logs. Afterwards it requires to be abstracted to the high level context information. According to [37] this happens in three ways: (1) one-to-one: one low level context value matches one high level context dimension, (2) context fusion: several low level context values match one high level context dimension, (3) context fission: one low level context value matches several high level context dimensions. Accordingly, we prefer to define context abstraction as process which asserts new contextual information by processing available context information.
We refer to [55] for an analytical understanding of context abstraction. We incorporate low level context – high level context mapping approaches given in [37] and [55] (see Figure 6). [55] defines “application space” (i.e. or in broader sense: context space, C) as the universe of discourse in terms of available contextual information for an application and defines subspaces which reflect the real life situations within application spaces, which are, “situation spaces” (S). Authors further define “context state” as collection of context attributes’ (i.e. dimensions) values at time t. Each context dimension have a “value space” (Vn) where value spaces represent range of values that a particular context dimension might have (e.g. 01 to 100 for age of a user). These value spaces might have discreet number of qualitative or quantitative elements or might represent a continuous range (discretization required). According to [55], some context dimensions might have greater importance than other context dimensions for a specific situation, therefore a weight is needed to be defined for each context dimension in each particular situation. Furthermore, authors note that for a particular situation, every context dimension can only match to some accepted values in its value space, and each accepted value in this set might have a different level of importance for this particular situation. Therefore every accepted value for a particular context dimension in a particular situation should have a different weight assigned (e.g. number of people in a room: 40 people should add greater contribution than 10 people would add for the situation of having a party, for example, where number of people in a room might vary from 10 to 50 for the situation of having a party). Moreover, some situations in situation space consist of combination of other situations (i.e. sub-situations). In order to have a consistent terminology we advocate the following understanding by re-interpreting [55]. Context information which maps to an adaptive behavior is a situation where a situation might be abstracted from low and high level context information and from other situations. A single atomic context dimension is low level context information where high level context is abstracted from low level context information and from other high level context information. High level context information does not map to any adaptive behavior but to the situations. Adaptive behavior represents both actions (manual, automatic etc.) and the change in application’s normal flow and structure (e.g. adaptive presentation, recommendation etc.) based on the context. Accordingly we prefer to define context reasoning as a function which maps situations to adaptive behaviors. It is worth to note that abstraction can also be considered to be a reasoning process; however, for the sake of simplicity and consistency, we prefer such distinction.

According to Figure 6, situation $S_1$ is abstracted by one to one match of context dimension $c_1$, and $S_2$ is abstracted by fusion of $c_2$ and $c_3$. $c_2$ affects several situations (i.e. $S_2$ and $S_3$), that is, fission. Furthermore, some situations in situation space consist of combination of other situations (i.e. sub-situations). For instance $S_2$, $S_4$ and $S_5$ are sub-situations of $S_5$, since context dimensions of sub-situations are totally covered by $S_5$. However this also requires that situation $S_5$ and its sub-situations need to have same accepted values of their context dimensions. Weighting approach allows us to calculate confidence values for inferred situations. That means provides a way to deal with ambiguity of context information.

Projecting low level context information to the high level context information and mapping situation space to behavior space, that is building the relation between the
context and adaptive behavior, are not usually straightforward. It is hard to handle these mappings for systems having huge application and behavior spaces. The difficulty also arises from the main characteristics of context as discussed before: context is a dynamic construct, it is relational and imperfect. Context abstraction and reasoning based on ontologies is mainly handled by rule sets (i.e. pre-defined or user defined [48]) and ontological reasoning, i.e. subsumption and realization [71, 73]. A typical system usually includes a knowledge base and a context reasoner. Knowledge base stores terminological knowledge (in a T-box) e.g. concepts, properties etc., and assertional knowledge (in an A-box), e.g. individuals. Subsumption determines subconcept-supercconcept relationships of concepts occurring in a T-box where realization computes which a given individual necessarily belongs to [73]. Reasoner holds context transformation rules in order to abstract low level context information, and context-behavior binding rules which binds context dimension(s) into a particular application behavior [37, 73].

![Context abstraction diagram](image)

**Figure 6:** Context abstraction based on one-to-one, fusion, and fission approach. V sets represents value spaces for context dimensions, C represents context space while S represents situation space.

Considering adaptive behavior; rules which maps application space to behavior space (i.e. to the automatic behaviors) are usually pre-defined or user defined, as previously mentioned, that is called first order adaptation [74]. On the other hand if such rules are learnt by the system (i.e. through machine learning techniques), that is called as second order adaptation [74].

### 6 Key Problems and Basic Approaches

In this section, we will briefly refer to some key problems and basic solution approaches. Scope of this section is mainly limited with ontology based approaches, hence problems and solutions also are.
6.1 Dealing with Imperfectness

Imperfection of context has been studied by several researchers in the literature. Since basing automatic actions on imperfect context information is problematic, researchers usually refer to user involvement to decide on correctness of the context or actions (i.e. mediation), to detect inconsistencies or evaluate correctness of the context based on artificial intelligence techniques, e.g. [27, 35, 44, 45, 75, 76].

Considering other approaches in the literature, it is quite common to employ a metadata approach [77] to annotate acquired and derived context information with quality parameters. RDF reification is a common way of annotating ontologies based on OWL and RDF with quality parameters. However [78] notes that such approaches are not expressive enough to capture rich types of context information and to support reasoning. [79] list several metadata parameters for the quality of context: (1) precision, (2) confidence, (3) trust level, (4) certainty, (5) granularity and (6) Uptodateness, while [38] uses following similar quality measurements: (1) accuracy, (2) resolution, (3) certainty and (4) freshness. Uptodateness or freshness usually associated with an aging function based on a life cycle management approach where this aging parameter also affects the value of other quality elements like decreasing confidence or accuracy level depending on the freshness of the context information [57, 80, 81]. [78] notes that related approaches in the literature for reasoning about uncertainty with various metadata terms such as confidence and accuracy are not expressive enough to capture rich types of context information and support reasoning mechanism. Therefore authors decide to go for an integrated solution by combining Bayesian networks and ontologies, since ontologies are good at representing structural contextual information where Bayesian networks are good at representing probabilistic contextual information. Such approach combines probabilistic models for uncertainty and ontologies for knowledge reuse and sharing. Authors achieved their aim by adding new language elements to OWL and by creating a mapping through OWL model to Bayesian model.

Approach presented in [55], which has been introduced in Section 5, can be considered amongst more generic solutions. The introduced weighting approach allows us to calculate confidence values for inferred situations since every context dimension and every acceptable value for a contextual dimension is associated with a weight value. That means it provides a way to deal with ambiguity of context information. Authors also enable agents to merge or partition different perspectives of context which are managed by different agents in order to provide increasing level of accuracy. Another approach represented in [73] introduces means to handle irrelevant context dimensions where OWL ontologies are used as a representation formalism. It uses a context filter where authors define situation-action mapping as a policy. The more a policy is used, the more important it is. Authors use a weight recorder to record usage of policies where they eliminate irrelevant contextual information according to usage records of policies.

Since it is not possible to clean all the ambiguity of the context information based on artificial intelligence techniques, metadata approaches or many others, it is reasonable to use artificial intelligence techniques and others to some extent, and to employ a user mediation mechanism [27] for crucial situations, examples include [27, 75]. [76] emphasizes user involvement because user knows more, without user
involvement system cannot evolve and system can lead wrong operations. The matter is enabling right level of balance between automatic actions and user mediation which should of course optimized based on priorities and importance of the situations.

6.2 Reasoning Performance and Manageability

Ontologies might grow up into huge knowledge bases which is problematic along with the heavy reasoning load for resource constrained devices in a pervasive environment [82, 83]. Through experiments [69] concludes that reasoning is time expensive, but still good for non-real-time applications. Authors identify three main performance factors: CPU speed, complexity of logic rules and size of context information.

In [69], authors suggest separating context use and reasoning where reasoning is done by resource rich devices and complexity of rule set need to be controlled. In a knowledge base there is a T-box which holds general concepts, their properties etc., and an A-box which holds individual specific information (i.e. instances). T-box is usually static and classification and loading is time consuming in T-box [84, 85], hence it is usually loaded and classified offline [85]. There are numerous attempts in the literature to cope with this challenge like partial ontology fetching and evaluation, ontology encoding, synchronization and replication of ontologies etc. [30, 83, 86].

The most basic approach is to create plug-in (i.e. modular) ontologies, it is also beneficial from management point of view. [87] notes that modularity is the key requirement for large ontologies in order to achieve re-use, manageability and evolution. Usually there is an upper ontology (generic ontology) and a domain ontology (lower ontology, or plug-in ontology) [53, 62, 69, 88, 89], this approach enables corresponding domain ontologies to be plugged into a generic ontology based on the application domain. We further advocate that a domain ontology alone also might include considerable amount of irrelevant contextual information hence it needs to be further partitioned, it is reasonable to call these sub-partitions as task ontologies where the root element of such ontologies are called as active or master context element. An example might be as follows, a smart home domain ontology can be partitioned as bed-room ontology, kitchen ontology etc., where master context elements are “being in the bed-room”, and “being in the kitchen” respectively. It is reasonable to say that identifying such active context spaces might be used to control size of T-box and logic reasoning. A possible approach might be using basic data-mining techniques over the condition set of the inference rules in order to partition context space. A similar approach is presented in [90], since only one context is active at any point in time the number of rules that have to be evaluated are limited. [61] remarks that A-box increase causes exponential increase in reasoning process time, hence only related items need to be collected at the time of reasoning [84]. [61] notes that subscribe (PUSH) method allows us to know what we need in A-box beforehand for Pre-selection. Approach presented in previous subsection which focuses on eliminating irrelevant context information based on policy recorders [73] also enhances reasoning process according to the view presented in this section.

Performance of reasoning engines is also of crucial. [91] lists two types of inference engines which are database (DB) based and main memory (MEM) based. In
main memory systems reasoning is done when the query is requested. They are more efficient but they lack scalability because of memory needs. DBMS based engines are slower but good choice when large and complicated knowledge is required, and they are scalable. [91] evaluates performance of following inference engines Minevra (DB), Hawk (DB), Pellet (MEM), Jena (MEM) based on a set of criteria, e.g. load time, query response time, query soundness and completeness etc.. Experiments lead authors to conclude that all the mentioned inference engines are far from being commercialized although Jena presents a better performance overall. Although research on enhancing performance of reasoners is challenging, only encountered example is [85] which employs prime numbers to encode concepts in an ontology for enhancing ontological reasoning (e.g. subsumption).

We refer to scalability and manageability issues briefly, prominently based on a database approach. [92] notes that database style management is much more scalable then ontologies however it is not standardized. Reasoning engines usually hold individuals and concepts in a specific format (e.g. RDF triples) which is usually not subject to be accessed directly by other users or applications, even this is possible, it is hard to manage. However database style of management allows other users and applications to access and manipulate data in a easy way (e.g. various views, query engines etc.). Contextual information is not only required for reasoning purposes, applications and users might also need to manipulate such information. For example, imagine set of questions (i.e. items) and answers which are given by students to these questions. They are stored in a DB, item difficulties can be considered as contextual information. In order to abstract item difficulties from set of answers given, a computational process is required which is difficult to apply through ontological representation of the data. [93] uses a hybrid approach based on using knowledge bases and databases however authors limit use of databases for static contextual information. We advocate that scalability and manageability of databases and reasoning support of knowledge bases need to be employed together. Therefore we propose following rough model which is inspired from SQI [94] which might be of use. Overall approach is depicted in Figure 7. According to proposed model, contextual information should be kept in databases, and only required contextual information need to be loaded to knowledge base for reasoning purposes. A query interface enables various applications and agents to submit queries in different query formats (e.g. SQL, SPARQL, RDQL etc.) which is subject to arrangement between the application and the query interface. Query is mapped to local query language of the database or knowledge base and query results are returned back in a common format (RDF, XML etc.) which is also subject to arrangement. Application also can send a command to load related contextual information from database to knowledge base. In order to enable such approach, a wrapper needs to maintain a mapping between knowledge base and database. Automation of such mapping is possible, we refer readers to Section 7 for details of such automation and mapping.
Figure 7: Merging relational databases and knowledge bases in order to enable scalable and efficient context reasoning and context management, only relevant contextual information is loaded for reasoning.

7 Towards a Generic Approach

[95] reports that context-aware applications are not yet come into market because of high development overheads, social barriers such as privacy and security, and an imperfect understanding of truly compelling uses of context-awareness. Furthermore several researchers remarks that context-aware computing lacks of appropriate infrastructure and middleware support, e.g. [38, 96, 97]. Hence, several research initiatives focused on developing such frameworks or middleware infrastructures based on various available software architectures, methods, techniques etc., e.g. [53, 54, 57, 65, 98, 99].

Almost none of these developments or frameworks has been really considered as a killer application, they are usually based on context models and encapsulated common functionalities in one way or another way, e.g. agent based [98, 99], service oriented [57, 65], central brokers [53, 54] etc. However approaches presented in the current literature of pervasive computing did not really manage to go beyond the borders of traditional computing and software engineering, although use of context models, particularly based on ontologies, can be considered as an important movement. Available studies employ ontologies for modeling and reasoning over context information, however we advocate that use of ontologies should be employed in every phase of software development, that is, both for separating reasoning logic and designing and specifying software artifacts. In other words we consider shift
towards approaches based on higher abstraction as a key challenge in order to cope with increasing complexity of pervasive computing environments. Secondly, available approaches greatly undermine the place of World Wide Web (WWW) for tomorrow’s pervasive computing environments. WWW is the biggest available digital layer of today and it is reasonable to claim that it will continue to be so tomorrow. Therefore, it is a fact that utilizing such a huge information source for pervasive computing environments is of great challenge. Fortunately, semantic web approaches which are already being used for context modeling and reasoning will be of great help. In Section 7.1 and Section 7.2, proposed approaches are elaborated respectively.

All in all, researchers should re-construct and adjust software engineering approaches and use of WWW for pervasive computing environments which is tomorrow’s computing indeed. In the following subsections we introduce our approach for these two mentioned challenges. Bear in mind that although the approaches we are going to mention are not purely novel since they are studied in their corresponding domains, our contribution is mainly based on synthesis and integration of such promising approaches based on a pervasive computing perspective.

7.1 Application Development

Apart from introducing new challenges pervasive computing also greatly catalyzes the problems inherent to the software development. Such challenges can be considered from development-time and run-time point of view. It is a known fact that maintaining knowledge of the application is essential since software development is subject to various changes. [100] refers to four fundamental forms of change: personnel (e.g. programmers, designers etc.), development platforms, deployment platforms, and requirements. Hence, in traditional application development, practically, that expert knowledge is lost; more accurately, that knowledge is embedded in code ready for architectural archaeology by someone who probably wouldn’t have done in that way [101]! Therefore, a properly managed application knowledge ensures sustainability of the application by absorbing such changes. On the other side, pervasive computing requires computing entities in such environments to be aware of each other’s characteristics and functionalities and to be able to communicate and share information in order to ensure collectivity. That is, assumptions done at the development-time should be minimal and applications should be able to adjust themselves according to the various run-time settings which might differ from each other in the sense of underlying technology, capabilities, requirements etc.. Approach presented in [102] reflects our understanding for context-aware pervasive application development (e.g. application context, hard device context, soft device context etc.). [102] simply considers devices as portals, applications as tasks, physical surroundings as computing environments. Based on this vision, authors divide the application life-cycle into three parts: design-time, load time and run-time. Authors define criteria and models for each part. Considering design time, it is suggested that applications and application front-ends should not be written with a specific device in mind. Besides applications should not have assumptions about available services, therefore abstract user interfaces and abstract
services need to be described. The structure of the program needs to be described in terms of tasks and sub-tasks instead of simply decomposing user interaction. Considering load time it is suggested that applications must be defined in terms of requirements and the devices must be described in terms of capabilities. Considering run-time, it is noted that it must monitor the resources, adapt applications to those resources and respond to changes. Such approach is based on higher abstractions of entities, including applications themselves. Indeed, that is how programming evolved from machine code assemblers to data structures, to object oriented languages and to the compilers in order to cope with increasing complexity. High-level languages replaced assembly language, libraries and frameworks are replacing isolated code segments in reuse, and design patterns are replacing project-specific code [103]. The next cycle of the abstraction, compelled by pervasive computing era, needs to reduce semantic gap between problem domain and representation domain based on higher abstractions of the business logic, application itself, and the reasoning logic based on contextual information. Conceptualizing the problem domain which is based on encapsulated abstract representations of entities, their capabilities, requirements, available functionalities and the complex relations between these entities and their characteristics will greatly reduce semantic gap with the representation domain and will isolate developers from the low level technical aspects of development. Ontologies might be considered as solution for such a higher level of complexity. [104] notes that it will be important to integrate ontologies with the software generation and management, perhaps using ontologies to semi-automatically generate interfaces. We further advocate using a top level of abstraction to automatically derive required software artifacts ranging from application code to specification, that is, letting programmers specify what programs should do rather how it should do it [99]. Moreover, ontologies can be used to automatically verify applications [105] before creating the code by means of using ontological reasoning process.

Early examples of context-aware pervasive applications are rather ad hoc, and are not based on a high level context models, hence reasoning support is not available, limited or hard coded. Examples include [31, 106, 107], although these systems did progress in various aspects of pervasive computing they are weak in supporting knowledge sharing and reasoning because lack of a common ontology [53]. Pervasive computing vision has opened up infinite context space which is required to be bind over infinite behavior space. Hence, it is hard to manage model of a setting and increasing number of rules in an ad hoc way, besides it is hard to share or reuse constructed knowledge which is directly hard coded into the application. Accordingly, latter applications are based on high level context models, particularly based on ontologies represented by OWL, RDF, UML etc., examples include [33, 62, 69, 70, 99, 108]. However existing work in the literature is mainly based on using traditional software engineering and computing paradigms in one or another way (i.e. various software architectures, encapsulating context management functionalities in various ways etc.), but far away from being revolutionary, and the novelty of contribution is almost limited with separating reasoning logic from application code. However, according to our perspective, ontologies need to be employed in every part of context-aware pervasive application development in order to enable higher abstraction. [109] notes that ontologies can be of use for (1) communication between computer-computer, between human-human, between human and computer; (2) computational
inference; (3) knowledge re-use and organization. Communication between computer and computer addresses the interoperability problems, where human-human communication addresses the terminological ambiguity between developers and leads to a consistent framework for unification [110, 111]. One of the benefits of using ontologies is that they aid interaction between users and the environment since they concisely describe the properties of the environment and the various concepts used in the environment [104]. Particularly, enabling higher level of user-computer communication is of help for user mediation which is only possible when both entities share the same conceptual understanding of the setting. Furthermore, [112] points out that ontologies can be used for software engineering either at run-time or at development-time. Having a knowledge base which is external to application for reasoning purposes is example of use of ontologies at run-time (i.e. computational inference). Considering development-time, a system can be specified and designed by the use of ontologies in a computing independent way, then designed ontology can be used to automatically generate application code, code skeletons (i.e. skeletal code) and other software artifacts such as database schema, UML diagrams etc.. Moreover, constructed knowledge is preserved and is ready to be re-used or to be shared. Development-time use of ontologies is highly undermined by previous approaches for context-aware pervasive computing. Indeed a typical context ontology, by the nature of context, involves considerable amount of application knowledge. Therefore constructed knowledge should be used for automated code generation rather then re-modeling, re-defining and manually generating application.

Hence, we refer to related literature briefly for ontology driven development which can be employed for context-aware pervasive application development. [113] proposes a development method called “ontology oriented programming”, where the problem domain is expressed in the form of an ontology and such ontology is used to generate object oriented application code. This is programming paradigm is of a higher abstraction level than object-oriented programming (concepts versus objects), but which finally, through the indicated compiler, makes it possible to generate object-oriented code [113]. Although [114] sufficiently addresses the related literature for ontology-driven development, particularly at development-time, Model Driven Development (MDD) [100, 101, 105] approach which is based on the same idea of automatically generating application code from models seems to be more mature. This is because of the experience, tools and standards available for this approach are more standardized and advanced. Prominently, Model Driven Architecture (MDA) [115], which is initiated by OMG consortium, holds an important place for MDD. MDA initiative offers a conceptual framework for defining a set of standards in support of MDD [104]. MDA software development life cycle includes a five step process [103]: (1) capture requirements in a Computing Independent Model (CIM), (2) create a Platform Independent Model (PIM), (3) transform the PIM to one or more Platform Specific Models (PSM) by adding platform specific rules and code that the transformation did not provide, (4) transform PSM to code, (5) deploy the system in a specific environment. UML standard which uses UML meta-model [101] is in the core of MDA for modeling. We previously mentioned that UML is considered to be a software engineering paradigm which can be used for representing ontologies, however it is only limited to represent lightweight ontologies. Therefore, UML approach in MDA might not be a proper choice to model
both reasoning logic, application logic and contextual entities. Hence use of OWL and OWL knowledge representation ontology (i.e. ontology to represent an ontology, corresponds to UML meta-model) instead of UML and UML meta-model might satisfy our purposes. People use UML or object oriented languages because they are more close to development layer and might facilitate it, so OWL should also come closer to development layer [115]. This is possible by developing easy-to use visual development environments and tools. Accordingly, MDA process can be adopted to such ontology based approach as shown in Figure 8 [117].

In this approach, first a domain ontology need to be created, probably by applying one of the techniques [71, 118] for ontology development (omitted in the figure for the sake of brevity). Later, part of such ontology need to be employed for reasoning purposes, this is because not every element of this ontology need to be part of reasoning logic but rather part of application logic. Therefore, a Platform Independent Application Model (PIAM) is to be subtracted from the Domain Ontology. Platform Specific Application Models (PSAM) - e.g. JAVA, .NET etc. - and Artifact Dependent Models (ADM) need to be derived from PIAM. Finally, platform specific code, and various software artifacts need to be created by using PSAM(s) and ADM(s) respectively. Furthermore it might be required to fine-tune the code itself or to complete skeletal application code. Inserting handwritten code in MDA is especially important in MDA, because the process is both model-driven and iterative. That means that MDA tools are continually generating code [103]. Use of ontologies as a top level abstraction and to map it to different purpose-specific representations is supposed to enable rapid, sustainable application development which is quite suitable to the nature of application development for context-aware pervasive settings. An interesting example is presented in [78] where authors derive a Bayesian model from an ontology, that is, to merge Bayesian models and ontologies, for better reasoning in

![Figure 8](image-url)

Figure 8: An integrated abstract software development approach based on Model Driven and Ontology Driven Development where models are used both for automatic software artifact generation (i.e. development-time) and for creating external reasoning logic (i.e. run-time) [117].
the sense of context quality. Such example clarifies what we do really refer by saying “purpose specific representation”, it does not necessarily need to be application code, or database schema, higher expressivity of ontologies enables them to be mapped less expressive representations. Complexity of pervasive spaces requires availability of different viewpoints of a model (e.g. a UML diagram might be more proper for documentation purposes, since it provides higher visual expressivity).

In this stage, it is required to have a brief look at available work in the literature which focus on ontology based automatic software artifact generation, particularly application code and database schema. [119] shows how to convert RDF schema and RuleML sources into Java classes, and [120] presents how to create a set of Java interfaces and classes from OWL ontology such that an instance of Java class represents an instance of a single class of the ontology with the most of its properties, class relationships and restriction-definitions maintained [120]. Similarly in [121], the authors show how an OWL/RDF knowledge base can be integrated with conventional domain-centric data models (Enterprise Java Beans) and object-relational mapping tools (e.g. Hibernate). Considering relational databases, [122] presents a mapping technique from ontologies to relational databases in order to facilitate rapid operations (e.g. search, retrieval etc.) and to utilize benefits of relational management systems (e.g. transaction management, security etc.)

[97] points out that middleware must enable programmers to develop applications dynamically without having to interact with the physical world of sensors, actuators, and devices. In other words, we need a middleware that can decouple programming and application development from physical space construction and integration. We further advocate that ontologies also have potential to enable users to program their own environment by means of synchronizing (e.g. sequencing, conditioning etc.) accessible services given by various entities (e.g. Outlook, TV, refrigerator etc.) of the environment. This is more than just being in the loop by means of meditation. We refer to such approach as “environment programming” which is only possible by enabling users and computers to share same conceptual understanding and enabling different entities to be plugged in the environment in a plug and play manner in order to advertise their available services.

7.2 The Semantic Web

Pervasive computing enlarged traditional computing setting into the human layer of the earth. World Wide Web is the biggest digital information layer, hence it is unavoidable to stretch and integrate such an information layer over this new enlarged computing setting [123]. Two different approaches can be listed to merge the Web and pervasive computing environments, the first one is from application point of view where researchers use the Web as an application and communication space, that is, mapping is from real world to the Web as presented in [124] where real life objects have web presences. Second approach is from information point of view [123] where mapping is from the Web to real world which is our focus in this paper. Various researches in the literature use the Internet as an information source, and for many others use of Internet might greatly advance their work (e.g. schedule information for
smart spaces etc.), examples include [61, 99, 125]. Particularly, challenge can be identified as follows [123]:

...to enable variety of devices in pervasive computing environments to be able to extract and use valuable web resources by semantically structuring commonly published information chunks (e.g. events, user profiles etc.) and annotating various web documents with contextual information (e.g. size, format, requirements etc.) in order to enable adaptive retrieval and presentation of such documents.

Semantic web standards (e.g. OWL) have been used for varying challenges of pervasive computing systems such as context modeling as previously mentioned. However apart from its constructive existence in pervasive computing environments, semantic web activities, particularly embedded semantics [126], also have a crucial role for enabling pervasive computing environments to exploit web information. Different devices in pervasive environments connected to each other in various means such as wired and wireless networks, infrared connections, Bluetooth etc. Apart from these local ties, pervasive computing environments are also mostly connected with World Wide Web environment as shown in Figure 10 (a). Web environment is a huge information source, hence enabling pervasive computing environments to exploit valuable information in the web environment without imposing any extra burden is a challenging task which is of prominent importance.

Semantic web components such as XML, RDF, OWL etc. allows machine understandability of information however in explicit means. They are useful for system level aspects such as context modeling, or service messaging etc. and they aim at machine readability. However, RDFa [126], eRDF [128] and Microformats [129], embedded semantics, enables implicit annotation of information in webpages by using class attributes of (X)HTML elements which both provides human and machine readability of information. Accordingly, four layers of abstraction for information (including contextual information) can be identified (see Figure 9) which is adopted from three layer of abstraction proposed by [34]: (1) storage layer, (2) exchange layer, (3) conceptual layer, and (4) representation layer. Representation layer, particularly embedded semantics, constitutes the missing link in current approaches.

**Figure 9:** Four layers of abstraction for contextual information and information itself: Storage Layer, Exchange Layer, Conceptual Layer, and Representation Layer.
With respect to the partial context model depicted in Figure 10 (b), such approach particularly focuses on the information which is part of the digital environment together with the applications. This approach is also in line with the idea we mentioned in Section 3; context for information (i.e. information as an independent entity). Annotating web documents by contextual information or structuring commonly published information chunks by using proper and standard meta-data elements and vocabularies will enable context aware applications to filter, search and recommend and present these information pieces to users depending on the match between user context and information characteristics (i.e. context). An example is learning objects [12]; each might have different competence levels, media format etc. that might interest devices with different capabilities or users with different competence levels.

Structuring meaningful information chunks residing in the web environment and annotating various documents with their respective contextual information enables us to identify and retrieve information of interest easily by leaving out unnecessary content. Since information is decoupled from presentation, such approach also enables us to present retrieved information by making use of abstract user interfaces and multimodality.

Figure 10: (a) Top level view of pervasive computing environments and World Wide Web, (b) partial view of our generic conceptualization.

8 Conclusions

Available research on context-aware pervasive computing lacks of a general understanding and a concrete methodology although required knowledge and vision are already distributed over respective literature. Particularly software engineering requires to be revised in order to cope with complexity which is introduced by pervasive computing. Proper understanding of context and its relation with adaptivity is of crucial in order to construct a new understanding for context-aware software development for pervasive computing environments. Role of the user in such environments need to be clearly understood in order to decide on right level of user control and automatic system behavior as well as for involving users for the mediation purposes. Moreover, pervasive computing expands the physical infrastructure of the digital environment which requires WWW to be properly coupled with such physical infrastructure as an ultimate information source.
Accordingly, we first introduced our general understanding and methodology both in generic and domain-specific sense (i.e., e-learning). Through this paper, we spend an effort towards integrating, and extending available theory and basic practice into a common understanding and into a conceptual framework which will lead us during both our long term and short term research. We elaborated on context and adaptivity by creating links with the application development issues. We referred a combined use of ontology-driven and model-driven approaches both at run-time and development-time where current practice is only limited with run-time use of ontologies. Such combined approach which is based on increasing level of abstraction might greatly facilitate rapid and sustainable application development. We further pointed out the semantic web approaches and web itself within the perspective of context-aware pervasive computing and indentified comparatively important challenges.

Our future work will be more focused on specifics of anywhere and anytime adaptive e-learning environments in compliance with general understanding and framework introduced in this paper. Consecutive complementary studies are expected to complete theoretical and technical framework constructed in this paper, that is, first two levels of our research pie depicted in Figure 1. Further work will be based on specifics of adaptivity and context for e-learning domain which is indeed uLEarning (i.e., ubiquitous learning) for our case. Our basic steps, more practically, will involve development of a generic and domain specific ontology for e-learning, and setting up required infrastructures and software components in order to make use of ontological reasoning and web information in such pervasive learning environments.

Acknowledgments. This paper is based on research funded by the Industrial Research Fund (IOF) and conducted within the IOF Knowledge platform “Harnessing collective intelligence in order to make e-learning environments adaptive” (IOF KP/07/006).

References


2.2 Formal Modelling, Knowledge Representation and Reasoning for Design and Development of User-centric Pervasive Software: a Meta-review

Authors: Ahmet Soylu, Patrick De Causmaecker, Davy Preuveneers, Yolande Berbers, and Piet Desmet

Published in: International Journal of Metadata, Ontologies and Semantics, volume 6, issue 2, pages 96-125, 2011.

I am the first author and only PhD student in the corresponding article. I am the main responsible for its realizations. The co-authors provided mentoring support for the development of the main ideas.

An earlier version was published in:

Increasing demand for large scale and highly complex systems and applications, particularly with the emergence of Pervasive Computing and the impact of Adaptive Systems, introduces significant challenges for software development, as well as for user-machine interaction. Therefore, a perspective shift on software development and user-machine interaction is required. An amalgamation of Model Driven Development and ontologies has been envisaged as a promising direction in recent literature. In this paper, we investigate this merged approach and conclude that a merger of both approaches, from formal modeling and Knowledge Representation perspective, on the one hand enables use of ontologies at run-time together with rules, prominently in terms of run-time reasoning, dynamic adaptations, software intelligibility, self-expressiveness, user involvement, and user situation awareness; and on the other hand at development-time, prominently in terms of automated and incremental code generation, requirement adaptability, preservation of application knowledge, and validation and verification of structural and behavioral properties of the software. The core contribution of this paper lies in providing an elaborate and exploratory discussion of the problem and solution spaces along with a multidisciplinary meta-review and identification of complementary efforts in literature required to realize a merged approach.

1 Introduction

Although the emergence of Pervasive Computing goes back to the early 1990s [1], we are still far away from completing the puzzle. It is reasonable to say that with the proliferation of hardware technologies, we have witnessed various advancements in networking technologies, computing power, miniaturization, energy consumption, materials, sensors, etc. (see [2-3]). However, Pervasive Computing [1,4] is not just about developing small computing residents for real life; a variety of applications exploiting the hardware infrastructure is the other side of the coin. Pervasive Computing (i.e., ubiquitous computing) aims at the creation of ‘intelligent’ digital ecosystems which seamlessly situate (i.e., are immersed) into the user’s physical
environment. Software ‘intelligence’, in such systems, is tightly coupled with the notion of adaptivity, that is ability of a system or application to dynamically customize itself to the computing setting and respond to changes in properties of the entities (e.g., device screen size, user competence level etc.), available in the setting and relevant to the computing process, by re-arranging its execution flow, interface, etc., accordingly. An immediate requirement for such ‘intelligence’ is context-awareness [5-9]. Context-awareness is defined as the capability of being able to perceive (through physical, virtual, etc., sensors [10]) the dynamic computing context and to respond collectively, proactively [11], properly, and seamlessly to better serve users without creating any distraction at the user’s side [12]. In this respect, context is any information (e.g., device screen size, etc.) characterizing the situation (e.g., characteristics, requirements, capabilities, etc.) of any entity (e.g., applications, users, devices, etc.) [6, 8].

The emergence of complex systems, particularly with the rise of the Pervasive Computing era [1] and the impact of Adaptive Systems [13], introduces significant challenges for software development, as well as for user-machine interaction. On the one hand, traditional software is designed for a specific and restricted context of use [12] following a one-size-fits-all approach. On the other hand, today’s ‘intelligent’ software systems and applications try to address the individual differences of the users (i.e., personalization), or in a broader sense, customization to the context of the computing setting (i.e., adaptation). However, employed adaptation mechanisms are based on the enumeration of possible contexts of use through hard-coded and predefined mappings between context and behavior spaces (i.e., a set of available context elements and a set of possible adaptive behaviors respectively) [12]. Such mappings are built on strong logical assumptions, which are predefined and usually not explicitly available (i.e., embedded in application code) and do not take the semantic relationships between different elements of the application domain into consideration.

First of all, from a development point of view, adaptive and pervasive computing enlarges the context and behavior spaces of software substantially and, consequently, hardens the management of hard-coded mappings between the context and behavior spaces and implicit reasoning logic, as well as validation and verification of structural and behavioral properties of software. In turn, it hinders the consistency and sustainability of the development and the management process and the reliability of the software respectively. Furthermore, since the contextual information is not always explicit in pervasive and adaptive software systems and applications, it is required to exploit the semantics of domain to infer first-order relevant information at run-time. Such systems and applications are also subject to rapidly changing requirements demanding frequent structural changes which cannot be handled through dynamic adaptations, but rather with re-design and development (i.e., static adaptation or requirement adaptability). Accordingly, dynamic adaptation mechanisms, which are able to consume available contextual information and domain semantics, and cost-effective and rapid design and development methodologies, which can absorb the development-time adaptation overheads, are required. For the latter, it is crucial to enable incremental development of the software by re-using existing application knowledge without the need for redesign or re-engineering. For the former, it also
becomes necessary to validate and verify properties of the software (at least partially) at design time, and the available contextual information at run-time.

Secondly, from the end-user point of view, it has already become apparent that absolute machine control, i.e., fully-automated adaptations without explicit user intervention for the sake of a seamless and unobtrusive user experience, as manifested by the pervasive and adaptive systems vision, cannot be fully realized as of yet, due to the ever-growing context and behavior spaces and the imperfectness of contextual information. Furthermore, absolute machine control is even not desirable in many cases. In this regard, user involvement at run-time emerges as an important paradigm, as a way to enable user control and to prevent undesired automatic actions taken by the machines. Any approach putting the user in the loop by means of user control requires software to clearly communicate its relevant internal logic with users and to support users with appropriate mechanisms to incorporate their feedback/input to change/adjust system behavior. User involvement necessitates software to ensure intelligibility [14] of its behavior/decisions which is possible through user situation awareness [15], which is realized through communicating acquired contextual information relevant to the state of execution with the end-users possibly at a user-selected degree of detail and abstraction, and with ability to explain the reasoning logic behind (i.e., self-expressiveness through causal explanations) the automated actions taken or the recommendations given (guidance, advisement etc.). User situation awareness and intelligibility is also important for establishing user engagement, trust, and acceptance.

The following experience report puts the aforementioned considerations into a concrete form. A developer team reports its experience on the development and maintenance of a university’s automation system. The system is developed and maintained by the assistants (graduate students) of the computer science department. Each developer is attached to the team during her period of study (2 or 4 years), hence, members of the development team change regularly. The system handles academic and administrative facilities such as grade management, course management, registrations, scheduling, reporting, etc., as well as e-learning facilities such as offering educational content and tools. The system exhibits ‘intelligent’ behavior by automatically enforcing university regulations on the administrative and academic operations realized by the students, and on the academic and administrative staff. Therefore, the business logic of the system is driven and constrained through numerous rules based on the university’s regulations, which vary according to terms, faculties, departments, students (e.g., first year, second year etc.), courses (e.g., elective, mandatory etc.), etc. (e.g., if a student has three failing mandatory courses and has an average below a specific grade he/she has to retake the failed courses at the first term in which these courses are available, except final year students). The rule set is subject to frequent changes due to periodic end-of-term (i.e., academic term) revisions. The rules are distributed to different parts of the system and realized through programming language under use (i.e., not directly in the form of if ... then rules but through combination of ‘if clauses’, ‘for loops’, SQL queries, etc.). The system is always under active development to address changing needs of different academic and administrative departments. Different programming languages and frameworks are used by the developers as appropriate for easiness or appropriateness of the technology with respect to the task at hand. The university later decides on:
supporting access to and use of the system through mobile devices, etc.,
• enhancing the e-learning part of the system by offering adaptive learning materials tailored to the characteristics of the learners (e.g., knowledge, skills etc.), and other context entities such as device (e.g., mobile, desktop, etc.), location (e.g., environmental conditions) and time (e.g., available time of the student).

The system has to consider various contextual information while operating, and this is handled through a dense rule set distributed in the code, resulting in a high complexity. The following significant difficulties are witnessed:

a) **Management:** Since the rules and the relevant contextual information are embedded into the system and distributed, it becomes difficult to report which rules are in force and to add new rules by detecting the relevant context information. Semantics of the domain cannot be exploited, hence leading to a higher number of rules (e.g., a rule which applies to all users has to be enumerated for every user type such as student, instructor, etc.).

b) **Consistency:** It is difficult to validate and verify that the system behaves as expected, and it is almost impossible to check that the rules are not conflicting. Comprehensive and long testing periods are required.

c) **Design and Development:** The behavior and overall structure of the system is almost undocumented. Since the system is already at a massive size, a considerable investment is required for the documentation. Due to the regular change of developers, system knowledge is not fully known by any developer. The result is low productivity because of unavoidable repeats of re-engineering processes for every new functionality and revisions for each developer. After a certain period, due to discontinuation of support for the main implementation technology used, it is required to migrate to a new platform. The migration process necessitates a complete re-engineering and re-coding of the system. Since such a process requires a considerable investment, it is decided to freeze the system as it is and to consider developing a new system.

d) **Use:** although the system offers intelligent facilities which shorten the formal procedures drastically and are mostly not existent in similar systems, the users are quite negative about the system, as can be observed through student forums and complaints delivered by the staff. This is because of:

a. Erroneous rules due to misinterpretation of the developers (application knowledge of a developer depends on an error-prone re-engineering process requiring a considerable code review, and there is a lack of a common terminology between technical and administrative staff).

b. **Inconsistency of the rules:** Since there is no reference on which contextual information to use within the rule bodies (i.e., conditions), similar rules are implemented in different parts of the system with different logics, and behave differently, resulting in low reliability, trust and user acceptance.

c. **Inconsistency of context information:** Such inconsistencies usually originate from user mistakes, system errors, etc. and either lead to incorrect processes or termination of the user sessions. With the
emergence of adaptive and context dependent enhancements of the system, the amount of contextual information grows significantly, which results in considerable increase in such inconsistencies and system crashes. The administrative and academic staff complaints that the system should let them know existing inconsistencies, since erroneous processes result in severe data losses and errors, and students mostly complain about the number of system crashes as well as incorrect context-related adaptations. User-involvement, to deal with inconsistent or missing data, requires a systematic approach and appears to be almost impossible with the system at hand.

d. Since the system behavior depends on composition of different application and contextual information, in many cases it is not clear for users (even for the administrative and academic staff who indeed manifest the rules) why the system takes a particular decision or behaves in a particular way. Enabling the system to explain the logic behind each behavior/decision and to communicate relevant contextual information requires a considerable manual effort. These inconsistencies and problems can only be detected after various complaints are received from the users, particularly from students.

The aforementioned discussion along with the experience report leads us to conclude that a perspective shift in the current adaptive and pervasive systems vision, and novel approaches for the software design and development are required. Considering design and development, an incremental design approach needs explicit preservation of application knowledge at the highest possible level of abstraction, structural and behavioral (in terms of models), and use of semi/fully automatic mechanisms for transformation from application knowledge to application code. This allows the realization of static adaptations which cannot be handled through dynamic adaptations. To enable run-time reasoning/inference, as well as validation and verification of the structural and behavioral properties of the software and the consistency of the contextual information (with respect to structural model of the software), it is required to maintain application knowledge formally with its relevant semantics. Application logic and adaptations then can be defined on top of such formalized models explicitly, which leads to increased manageability. Considering the users, a formal and abstract model of the application forms a solid substance acting as an unambiguous communication medium and language between the software and its users. The software can express its internal logic and communicate contextual information relevant to its adaptive behavior through elements of the model.

Two important paradigms, namely, MDD and ontologies target the discussed main challenges. Although each paradigm addresses different purposes, they do share a common origin, i.e., abstraction. Hence, an amalgamation of MDD and ontologies has been envisaged, argued for a limited problem space, and presumed to be promising in [16-20]. On the one hand, in the software engineering domain, automated development of complex software products from higher abstract models has gained great momentum [21], and considerable expertise along with a mature tool portfolio has been constructed, particularly with the emergence of MDD. On the other hand, ontologies as KR and logic paradigm have been utilized as run-time and development-time software artifacts due to their higher level of expressivity, formal
semantics, and reasoning and inference capabilities. In this paper, we investigate how such a merged approach can alleviate the aforementioned problems, and conclude that merging both approaches has the potential to provide a rapid sustainable development process for manageable, consistent, and reliable ‘intelligent’ software systems and applications. The resulting approach employs ontologies:

- at run-time together with rules, for the purpose of run-time reasoning, dynamic adaptation, software intelligibility [14], self-expressiveness, user involvement, and user situation awareness,
- and at development-time, for the purpose of automatic code generation, requirement adaptability (i.e., static adaptation), application knowledge preservation, and validation and verification of structural as well as behavioral properties of the software.

Considering the run-time aspects, ontologies are of use as external knowledge bases, over which a reasoning component can reflect over the available contextual information. The use of ontologies enables the separation of application logic from code, thereby facilitating the management of reasoning logic and bindings between broad spaces of context and behavior. Furthermore, the use of ontologies provides a unified framework of understanding between computers-computers, users-computers and users-users, and makes reasoning logic of the software explicit. This, in turn, facilitates self-expressiveness, intelligibility, user involvement (e.g., user control, user mediation, adaptive guidance/advisement, and feedback), and user situation awareness. Considering the development point of view, an ontological approach, following the MDD path, can be used to automate application development and requirement adaptation. These are important for rapid and sustainable development of long-lived ‘intelligent’ systems and applications. The core contribution of this paper lies in providing an elaborate and exploratory discussion of the problem and solution spaces along with a multidisciplinary meta-review (i.e., conceptual sketch of the problem and solution spaces) and identification of available efforts in the literature that can be combined to realize the aforementioned merged approach.

The rest of the paper is structured as follows. In Section 2, we present our motivation in four respective subsections; we first elaborate on adaptive and pervasive computing and the notion of context, secondly discuss the affects of new computing trends on software development, thirdly comment on the computer way of exhibiting ‘intelligence’ with respect to human intelligence, and finally emphasize the necessity of having humans in the loop for future adaptive and pervasive computing environments. In Section 3, we present a theoretical background on MDD and ontologies, and discuss a possible merger of approaches with respect to existing literature. In Section 4, we refer to the related work that can be combined to realize the presented approach. In Section 5, we provide a discussion of the literature while we conclude the paper in Section 6.
2 Pervasive and Adaptive Systems

In this section, the driving motivation behind the overall approach will be discussed with respect to a broad multidisciplinary literature. It will be constructed over four important pillars questioning:

- the shift in the current computing paradigm and its impact on software engineering (see Section 2.1),
- design and development issues for software systems and applications following the new computing paradigm (see Section 2.2),
- ‘intelligence’ for software systems and applications with respect to human intelligence (see Section 2.3),
- user aspects of user-machine interaction (see Section 2.4).

We will particularly elaborate on the soaring challenges of software development, the level of ‘intelligence’, hence the adaptivity, that the current software systems and applications can exhibit, and the need for user involvement. We will discuss what formal modeling, KR and reasoning can offer to counter the elaborated issues while giving glimpses of an approach merging KR and software modeling instruments and fundamentals.

Application Life Cycle: Design Time

Stakeholder: Software developer

Figure 1: Application life cycle, design time, with respect to a possible merger of ontology and model driven considerations.

Figures 1 and 2 provide a generic overview of the subject approach towards intelligent adaptive and pervasive applications. It covers both the design and development phase of the application, as well as runtime aspects of how applications are enabled to exhibit smart behavior. The model based design, see Figure 1, is built upon software and knowledge engineering best practices. The former results in models that capture the structure and behavior of the application; the latter focuses on
models that formalize the application semantics and the operational context. A model-driven development approach is envisioned to blend software models like UML diagrams and MDD design artifacts, with knowledge models like ontologies and rules to generate context-aware adaptive applications. This model-based design process allows software developers to embed a formalized representation of the characteristics and behavior of the application as an explicit model into the software. These explicit models not only support the adaptation logic at runtime, but also allow the software developer to formally validate and verify key properties of the application.

**Application Life Cycle: Run-time**

Stakeholder: End-user

Pervasive Application \(\rightarrow\) Intelligent Behavior \(\rightarrow\) Intelligible and Self-expressive

- End-user involvement
- Situation awareness
- Adaptation

- Adaptation traceability
- Formal reasoning

**Figure 2:** Application life cycle, run-time, with respect to a possible merger of ontology and model driven considerations.

At run-time, see Figure 2, the end-user becomes the main stakeholder of the application, using it in a particular context. Based on the circumstances at hand, the application will anticipate the user’s intentions and adapt itself to fit the current situation. However, the intelligence that the software developer has put into the application may not match the end-user’s expectations. Therefore, it not only suffices that the application exhibit intelligent behavior through anticipation and adaptation, the end-user should also understand the application’s capabilities and its adaptation behavior. By tracing the adaptation steps back towards (i.e., self-expressivity through causal explanations) the conditions and the decisions that triggered the adaptation and with forward/backward reasoning, the application can offer insights to the end-user on why the (unexpected) adaptation occurred. The ability to explain its own behavior together with user situation awareness (i.e., user awareness on the state of execution and relevant contextual information that led to the current state) will be instrumental to implement support for the application’s intelligibility (i.e., the reason behind the behavior of the software is clearly understandable by the end-user). Intelligibility is the basis for end-user involvement, which aims at explicit user intervention for adjusting or designing adaptive behavior of the system.
2.1 Context and Adaptivity

“Programs must be written for people to read, and only incidentally for machines to execute.”

Abelson and Sussman

Development and management of software is not limited to the design and administration of small scale software systems and applications anymore [22-24]. An increasing demand for large and complex software systems that are able to adapt to dynamically changing computing settings has appeared [25] with:

- the rise of mobile devices and computing and, later, the emergence of Pervasive Computing [1-2, 4, 26],
- the increasing need for adaptive, particularly user-adaptive (i.e., personalized) software systems [13, 27-28].

The dynamic and heterogeneous nature of pervasive computing settings [29] requires software systems and applications to be adaptive to the varying characteristics, requirements and capabilities of changing computing settings and the entities and resources available through these settings. Prominently, they must provide a user-tailored computing experience by considering different characteristics and needs of the users. For instance, in the e-learning domain, it is already demonstrated that personalized computer-based instruction is superior to the traditional approaches [30]. In other words, the pervasive computing vision manifests an unobtrusive, anytime and anywhere [2] user experience which requires expansion of the personalization era to the context-awareness era. In this regard, the computing paradigm has changed from user-computer perspective to context-computing setting perspective. The context [6, 8] simply represents formalized snap shots of the computing setting with all its members (i.e., entities and resources), involving the user as a core entity.

The traditional computing process is often perceived as an execution of a program to achieve the user’s task; it stops whenever the task is fulfilled [23]. In contrast, the new perspective (see Figure 3) considers computing as a continuous process of recognizing user’s goals, needs, and activities and mapping them adaptively onto the population of available resources that responds to the current context [31] (i.e., context-awareness [5, 7, 9, 12, 26, 32-33]).

One needs to construct a clear understanding of ‘context’ and ‘application’ with respect to the pervasive computing vision before addressing the following main challenges:

- how to manage ‘intelligent’ behavior (i.e., adaptation processes),
- how to design and develop such pervasive applications.

Context is a broad concept encompassing an infinite amount of elements and therefore, the description of the notion is quite open. This leaves an important role to the determination of the scope of the context with respect to the subject application [12, 34]. Contextual information is mainly collected through physical sensors acquiring real world data, and through virtual sensors acquiring transactional data through the application logs [3, 10]. This type of contextual information (i.e., acquired through sensors) is called low-level context [35] and each represents an atomic fact called context dimension (e.g., humidity, etc.). It is usually required to infer new
knowledge from low-level context information, often through mapping available context dimension(s), probably each having a different weight [36], to particular composite context(s). This mapping might be one-to-one (i.e., one context dimension map to one context), fusion (i.e., several context dimensions map to one context) and fission (i.e., one context dimension maps to several contexts) [35]. The resulting contextual information is called high-level context.

![Diagram](https://via.placeholder.com/150)

**Figure 3:** The new computing perspective: context and computing setting.

Contextual information is often imperfect [7, 12, 14, 32, 37] because of incompleteness, irrelevance, ambiguity and impreciseness of sensory (i.e., virtual or physical) information, hence various techniques are required to be employed to avoid unwanted actions. We refer interested readers to [7, 9, 12, 33, 35-37] for further analytical and conceptual information on contextual inference and reasoning. The final phase is usually the definition of adaptive behaviors and their mappings to the identified contexts. Context-behavior mapping follows a similar inference procedure, i.e., mapping a set of related context elements – probably each having a different weight – to particular behavior(s). One can further abstract such sets of context elements in terms of ‘situations’ [14, 36] possibly with a similar weighting approach (e.g., situation: someone is cooking, context dimensions: light is on, heater is on, someone is in kitchen, etc.).

### 2.2 Design and Development

Regarding the development of pervasive and adaptive systems and applications, the perspective presented in [38] is notable. The authors consider devices as portals, applications as tasks, and physical surroundings as computing environments. The application life-cycle is divided into three parts: design time, load time, and run-time. Considering design time, it is suggested that an application should not be written with a specific device in mind, and it should not have assumptions about the available services. The structure of the program needs to be described in terms of tasks and subtasks, instead of decomposing user interaction. Considering load time, it is suggested that applications must be defined in terms of requirements and devices must be described in terms of capabilities. Considering run-time, it is noted that it must monitor the resources, adapt applications to those resources and respond to changes. The proposed vision fosters use of declarative methods for software development in which the focus is on what software should do rather than how it
should do it, and necessitates a development process based on high level abstractions not depending on any particular context. The software is expected to continually mediate (i.e., dynamically adapt) between changing characteristics of the computing setting and itself (i.e., context) to achieve its goals.

Context and behavior spaces can be encoded in the application itself and dynamic adaptations can be handled through hard-coded behavior-context mappings within the application through the programming language under use. However, such an approach is apparently insufficient and inflexible for the development and management of adaptive and pervasive systems and applications, since it is not possible to enumerate all possible context dimensions and behaviors, as well as mappings between ever-growing context and behavior spaces. Accordingly, it is necessary to maintain an extensible and formalized conceptualization of the possible computing setting, separate from the hard-coded application, on which dynamic adaptation rules can be created and executed anytime without touching the application code. This is important in the sense that an abstract approach, by following the end-user development paradigm [39], might also enable end-users to program and control their own environments in the future (i.e., through adjusting the application behavior or introducing new context-behavior mappings). This is what we call environment programming [17, 40-41] (or user-driven design of pervasive environments), and user control respectively. Although such an approach diverges from the main pervasive computing vision by employing user involvement, it allows different requirements to be addressed at run-time (i.e., adaptability: customization with explicit user input) without being identified at design time. In [42], it is criticized that it is impossible to create adaptation rules for all possible situations and eventualities.

Revisiting the development process issue, applications require redesign and re-configuration with respect to the various changes (e.g., functional, non-functional requirements, deployment platforms, etc.) [25] that cannot be handled simply through run-time adaptations. The need for such changes is expected to be higher for adaptive and pervasive systems and applications, since they are expected to address a variety of different contexts of use. Successful development approaches should allow incremental development of the software without requiring re-engineering and redesign from scratch, and ensure consistency of the software through its evolution. In such complex software systems and applications, it is not possible to identify a complete set of requirements beforehand, and it is hard to maintain application knowledge that is required to ensure a sustainable and rapid development cycle. Application knowledge is expected to be larger, which cannot be easily acquired through reverse engineering or assumed to be known by developers who might also change during the software life-cycle. Furthermore, on the one hand, validation and verification of such complex systems after or within the development process is quite costly, and on the other hand, validation and verification of the software at the design stage is a complex human-centric process. Apparently, explicit preservation application knowledge, in an abstract and formalized form [38] closer to the human-level language, is important to avoid repetitive coding of the application by automatically transforming application knowledge to software artifacts, including the application itself, to have a formal and unambiguous snapshot of the application knowledge at every stage of the development cycle, and to apply formal software
validation and verification processes (i.e., structural and behavioral) at early design stages.

2.3 Human vs. Machine Intelligence

“All programmers are playwrights and all computers are lousy actors.” Anonymous

The long-lasting debate on whether it is possible or not to built machines that can achieve human levels of intelligence [43-48] is still not over. Although humankind has considerably benefitted from Artificial Intelligence (AI), it has not even approached the higher level of human cognitive abilities and thought processes in computers, and it is not likely to do so in the foreseeable future [48] (opposed to the more optimistic views [46]). There are years of research ahead of us, most probably only with limited achievement in terms of real intelligence [47]. However, it is important to see that intelligence is a scale; it is not only 0 or 1. It might be very difficult to define exactly what the human level of intelligence is, in terms of quantitative measures, so it is more appropriate to talk about the qualitative degrees of intelligence [47], based on some concrete elements. Although it is important to figure out the necessary and sufficient elements of intelligence, it is more important – within the perspective of today’s software engineering – to ensure whatever (i.e., at whatever level and with whatever properties) we have as ‘intelligence’ to be:

- manageable,
- reliable and rational.

The latter should be appreciated by almost everybody, if not all, who has already experienced a dummy online customer or registration service, as mentioned in [48].

Considering the computer way of ‘intelligence’, it is possible to donate computational devices with a variety of sensors, and it is also evident that in the sense of processing power and computational resources (e.g., memory, CPU), computers are far beyond the abilities of humans. If so, why is today’s computing still far behind the human level of intelligence? In [49], it is said that:

“Indeed there are no right decisions; there are only decisions which are consistent with the given information and whose basis can be rationally related to the given information.”

This fundamental principle is the key to reaching an answer. It is true that the amount of information basis to decisions is crucial, however what is equally, if not more, important is the ability to infer implicit information hidden within the information at hand and to arrive at rational decisions through a reasoning process. In short, the answer lies behind the term ‘reasoning’, that is the capability to reason, converse and make rational decisions in an environment of imprecision, uncertainty, incompleteness of information and partiality of truth and possibility [48]. Computational systems are good at gathering and aggregating data and humans are good at recognizing contexts and determining (e.g., to reason) on what is appropriate [50].

Today, ‘intelligent’ systems and applications are ‘intelligent’ only to the extent of completeness of the real-world contexts modeled by the developers. Their
'intelligence' is strictly based on strong logical assumptions or computational and algorithmic procedures prepared by the developers [51]. Hence, we prefer to call such 'intelligence' machine encoded human intelligence/simulated intelligence (i.e., weak AI or pseudo-intelligence [52-53], and we prefer AI alone to refer to strong AI or human level of AI [46, 48]). This is because it is only a limited reflection of the human intelligence, which consists of a limited conceptual model and limited reasoning logic of the developers (i.e., mental models) on a specific problem. In other words, it is not intelligence itself through imitating functional aspects (i.e., how such mental models and rules are created), but rather an output/artifact of human intelligence. Technically, ‘intelligence’ in many current pervasive and adaptive systems and applications is usually predefined and implicit (refer to a survey on intelligent smart environments [3]). In these systems, reasoning logic mostly consists of hard-coded logical bindings spread into the different parts of the software code, and there might be several inconsistent versions of the same binding available in different components (due to different developers or forgotten knowledge of the software). Implementations are mainly small scale, or not reliable [14], which is primarily due to difficulty of managing ‘intelligence’ and growth of such systems and applications. Manageability problems mainly originate from implicit and hard-coded software knowledge and reasoning logic which can easily grow into heavy masses. It also becomes harder to check consistency of the software knowledge, adaptation logic, as well as its behavioral properties, thereby leading to reduced reliability and rationality.

Since predefined and implicit logic is not sufficient (either in terms of if-then rules or machine learning techniques, etc.), a step toward human-level AI, regardless of whether it is possible or not, requires reasoning about context [46] to exploit semantics of the domain. Manageability problems can be addressed through accommodation of reasoning components where the formal models and reasoning logic built upon are easy to manage and external to the application. Through employing formal models and reasoning processes, it becomes possible to extract first-order relevant information, which is implicitly available in information at hand, at run-time in a standardized manner, without requiring hard-coded logical bindings encoded in the software (see Figure 4).

Figure 4 compares two approaches, with and without exploitation of domain semantics, for an example course management application. The upper part of the figure reflects partial conceptual formalization of the application knowledge. The scenario assumes two types of system users, namely ‘Student’ and ‘Instructor’ which are subclasses of ‘Person’ type. The ‘involvedIn’ relationship is defined with domain ‘Person’ and range ‘Course’. ‘takes’ and ‘gives’ are sub-properties of the ‘involvedIn’ property with domain ‘Student’ and range ‘Course’ and domain ‘Instructor’ and range ‘Course’ respectively. A ‘Course’ has subclasses ‘Lecture’ and ‘Lab’ (i.e., laboratory session). Each ‘Lab’ is attached to a ‘Lecture’ which is realized through the ‘attachedTo’ property (which is symmetric, i.e., if (x, y) holds then (y, x) also holds). In an implementation without such formalized conceptualization, the semantics of the domain are implicit and only known by the developers. If a person is involved in a course then he also has to be involved in its attached courses. Since it is not possible to exploit domain semantics, this rule has to be implemented as shown at the (B) part of Figure 4, which enumerates the rule for each subclass of relevant classes.
Normally, such hard-coded rules are implemented through the programming language with combinations of ‘if clauses’, ‘for loops’, SQL queries, etc., and therefore, the rule is supposed to be lengthier than the one shown in part (B). In contrary, the same rule can be implemented more efficiently and explicitly by exploiting domain semantics, as shown in part (A) of Figure 4. Part (A) assumes that the application knowledge is explicitly available as well.

Figure 4: Complex systems are harder to design without the ability of exploiting the semantics of domain; (A) with semantics, developers construct generic rules through exploiting the semantics of the application domain (e.g., subclass, sub-property etc.), (B) without semantics, developers have to enumerate every possible concept while constructing the logic rules.

Although the approach is of use for manageability and for application development, imperfectness of the contextual information [3, 14, 37] decreases the level of reliability and rationality of the reasoning. The impact of this imperfectness might be severe, depending on the situation. Reasoning based on formalized conceptualizations can be used to some extent for consistency checking and verification and validation of structural and behavioral properties of the software. Furthermore, various AI techniques can be applied to alleviate imperfectness [54-59]; however, such techniques do not provide 100% success. Approaches based on human intervention, which will be further discussed in Section 2.4, seem to be required where fully automated mechanisms are not enough. In this respect, explicit software knowledge and reasoning logic construct a basis for user involvement. Ethical, social,
and legal aspects of human–machine relation are already subject to in-depth discussions [60-62].

2.4 Human and Machine Interaction

“Computers are incredibly fast, accurate, and stupid. Human beings are incredibly slow, inaccurate, and brilliant. Together they are powerful beyond imagination.”

Albert Einstein

Ubiquitous computing environments aim at immersing into the daily lives of humans with the promise of an enhanced and unobtrusive user experience through ‘intelligently’ satisfying the needs of the human beings. However, even human beings are not better at anticipating the real needs of others, even in relatively simple situations [53]. In this regard, successful ubiquitous computing systems need to satisfy several requirements. We identified the following among the most important ones within the theme of this paper:

- **User engagement** [63]: It refers to the ability of a system to attract and hold the attention of the users [64]. In [65], the authors remark that successful technologies are not just usable; they engage the users, given the increased emphasis on user experience [66].
- **User trust**: Trust is an important factor affecting user performance, which is defined as the ability of users of a system to satisfy their intentions and achieve their objectives efficiently and reliably [53]. The absence of trust introduces inefficiency, demanding added vigilance, encouraging protective and unproductive actions, and complicating interaction [53].
- **User acceptance** [15]: We understand user acceptance as the user’s intention to use a system and to follow its decisions or recommendations with willingness and contentment.

It is reasonable to say that, human intelligence is still the dominant intelligence. Therefore, to establish user engagement, trust, and acceptance, humans should be a part of the loop while using such systems [15, 53, 65, 67]. User involvement can be considered both as a:

- Development-time issue [68] referring to users being part of the development cycle by providing relevant feedback,
- Run-time issue referring to users’ ability to intervene in the application’s behavior at run-time, probably based on the appropriate feedback and guidance given by the application.

We place our focus on the latter within the context of this paper. The following interrelated elements are among the important constructs of the aforementioned requirements within the frame of user involvement (i.e., integration):

- perceived user control,
- user mediation,
- adaptive advisement/guidance and feedback.

In [15], the author points out that perceived control is the conviction that one can determine the sequence and consequences of a specific event or experience [69].
Control over a system might be totally held by the system or the user. Alternatively, it can be shared [70] while the final decision is still taken either by the system or the user. In the latter case, informative input is provided by the second party through user mediation [71-74] (i.e., user feedback to machine) or adaptive user guidance/advisement [75-76] (i.e., machine feedback to user). Considering adaptive advisement, we argue that adaptive application behaviors do not necessarily need to result in ‘musts’ or ‘have-tos’, but can also result in ‘shoulds’ and ‘mights’, leaving some control to the user while providing possible directions and the reasoning behind those directions (i.e., intelligibility through self-expressiveness and user situation awareness). The system can extend the limits of contextual information perceivable by the user’s sensory capabilities by serving contextual information gathered to the user, rather than automatically adapting itself, where incorrect actions might be frustrating [77]. Considering user meditation, as previously mentioned, adaptive behaviors are realized by means of predefined behaviors mapped to possible contexts of the setting and use. However, imperfectness of context information decreases the reliability of adaptive behaviors. Hence, according to the severity of the results, systems should be able to mediate with the user, to decide on the accuracy of the contextual information or the appropriateness of the possible adaptive behaviors, while the ideal case is placing fewer demands on the user’s attention [78-79]. In any scenario, it is required that the system is transparent, i.e., intelligible, to the user by giving appropriate feedback providing the underlying logic of decisions given and the awareness of the current or related context.

In [15], the author points out that pilots in cockpits most frequently ask questions like “What is it doing?”, “Why is it doing that?”, “What will it do next?” and “How did it ever get into that mode?” [80]. Furthermore, people usually resist the introduction of automation; for instance, there are strong debates between airlines and pilots in terms of the degree of automation in cockpits [15]. It is further argued in [15] that the reason for these questions and resistance is lack of situation awareness [81]. These incidents confirm the basic requirements mentioned; as previously noted, ‘intelligent’ computational systems only exhibit a limited representation of human intelligence, which requires involvement of the human user. User control over adaptation is preferred because the user can maintain the system’s interaction state within a region of user expectation [82], while delegating too much control over machines causes lack of situation awareness [15]. However, such systems should also be able to deliver the reasons for their decisions and the relevant context to the user clearly. Then users will tend to accept and use the reasoning of such systems [14, 83-84]. In this context, the assistance systems have an important place, in terms of enhancing user perception, interpretation of data, feedback, and motivation [85-86]. The solution lies is providing the right level of balance between automatic system decisions and user involvement. Apparently, such optimization should be based on the priorities and significance of situations to provide a better user experience.

In traditional software and in most of the current pervasive and adaptive systems and applications, communication of the adaptation logic and the information basis to adaptation (i.e., context) to the end-users is not truly addressed. Indeed, the way these systems are developed, as mentioned in previous sections, hardens the traceability of the decision logic in a consistent manner. Therefore, attempts towards user situation awareness, self-expressiveness, intelligibility, and hence, user involvement remain ad-
hoc and small scale. A formal representation of software knowledge, context, and adaptation logic provides a common substance of communication between machines and human users. Logic reasoning mechanisms employed on top of formal models enable traceability of the decisions arrived at, leading to intelligible software systems and applications (see Figure 5). This common language further allows users to deliver their feedback. Since it is possible to check the consistency of the contextual information and the logical assertions with respect to the available contextual information and software knowledge, the behavior of the software remains consistent at every stage.

Figure 5: A logic based approach leads to intelligible software systems and applications by enabling traceability of the reasoning logic behind the intelligent behaviour (i.e., self-expressiveness) and communication of relevant contextual information (user situation awareness).

Figure 5 is based on the example application given in Figure 4. It assumes existence of two rules. The first rule is (R1) already explained in Figure 4. The second rule (R2) ensures that if a student attempts to take a course (e.g., C2), and if she is already registered to a course (C1) which is same as course C2 (defined with the ‘sameas’ property), then that student should not be allowed to take course C2. In an example use case, a student takes a ‘Physics1’ lecture, and then attempts to take a ‘PhysicsLab2’ lab class. The application does not allow it and explains the reasoning. The application informs the student that she already took ‘PhysicsLab1’ lab and it is same as the ‘PhysicsLab2’ course. The student wonders why she is registered to the ‘PhysicsLab1’ course and the application informs her that she took the ‘Physics1’ lecture and it is attached to ‘PhysicsLab1’ lab. In that way, the application gradually
provides causal explanations for its reasoning logic through iterating over the inference chain, as shown in the lower part of Figure 5.

3 Formal Modeling, KR and Reasoning

MDD and ontologies are complementary in terms of their main uses, that are, automated code generation and reasoning respectively. They overlap in terms of abstraction which leads to the approaches surveyed in Section 3.3.

3.1 MDD

MDD aims at automatically generating application codes and code skeletons from higher order abstract models, thereby reducing the semantic gap between the problem domain and the solution domain. A model is defined as an abstraction representing some view of reality, by necessarily omitting details, intended for some definite purpose [87-88]. The shift towards higher abstractions has a long history; high-level languages replaced assembly language, data structures, libraries and frameworks are replacing isolated code segments in reuse, and design patterns are replacing project-specific codes [89-90]. It eventually approaches human language through the use of representation formalisms with a higher degree of abstraction [91] – any program code is simply another, albeit low level, abstraction [89] – thereby enabling programming less bound to underlying low-level implementation technology [92-93]. A basic development process in MDD starts with the identification of target platforms. Afterwards, it is important to select appropriate meta-models [94], which provide basic primitives (i.e., constructs) for developing models belonging to a specific subject area (i.e., any realization of source meta-models or target meta-models), and an appropriate language for formalization. The next step involves the definition of mapping techniques and required model annotations, defining the projection from source meta-models and meta-models of the target platforms. Mapping techniques can be executed over the models manually or automatically with tool support. This process is necessarily iterative, and human intervention in terms of code completion might be required (e.g., when the skeletal code is generated). Model Driven Architecture (MDA) [93], initiated by the Object Management Group (OMG), holds a prominent place for MDD. The MDA initiative offers a conceptual framework and a set of standards in support of MDD [95]. Prominently, UML [96] as a modeling formalism is in the core of MDA. MDA utilizes a meta-meta-model which allows construction of different meta-models belonging to different subject areas. The MDA process consists of five main stages [89-90, 97]:

- creation of a Computation Independent Model (CIM) which gathers the requirements of the system or the application,
- development of a Platform Independent Model (PIM) which describes the system design through defining its functionality without any dependency on a specific platform or technology,
- conversion of PIM into one or several Platform Specific Models (PSM) through application of a set of transformation rules,
- automatic generation of code form PSM(s) with another set of platform-specific transformation rules,
- deployment of the application or system onto a specific platform.

The design begins with a high level model and iteratively transforms the model to more concrete models through introduction of more platform specific information at every stage [98].

The benefits of MDD can be discussed under two main and interrelated categories: abstraction and automation. We first consider abstraction [92]. A model has multiple views, some of which are revealed [91]. Irrelevant details can be hidden based on a specific view of the model (i.e., separation of concerns), which in turn enables different experts to work on the system from different points of view. This is particularly important for enabling development of complex systems [93, 100]. One of the goals of MDD is to enable sensitivity to inevitable changes that affect a software system [101-102]. In [101], four fundamental forms of change are identified for software: personnel (i.e., developer), requirements, development platforms, deployment platforms. In [99], the authors point out that, practically, expert knowledge is lost since the knowledge is embedded in code ready for architectural archaeology by someone who probably would not have done it that way. An abstract model of the software ensures that the application knowledge is preserved and reduces the amount of effort to understand it (i.e., increased understandability [92]). This in turn ensures the sustainability and the longevity of the software. Furthermore, it allows quick implementation of business level updates, thus providing a potential for improved maintainability [102]. When considering automation, portability [97] is another concern for MDD, which is handled through creation of a new PSM from PIM, and regenerating the code and deploying it into a new platform without substantial code reviews [90]. This provides a faster and cheaper migration [89]. In [102], it is pointed out that code generation results in high reusability and increased productivity, since repetitive coding for the application is not required [89]. MDD increases the quality of the software. Firstly, errors are reduced by using automated tools for transforming models to application code [93]. Besides, it is possible to verify consistency of the models through formalized models [92]. Secondly, it becomes easier to automatically apply mature software blueprints and design patterns. Finally, the documentation process is well-supported with lesser, if not none, manual effort. Produced documentations are based on a formal model of the application, thereby preventing misinterpretations and ambiguity [104]. Although the initial cost of the investment is higher in the earlier stages, compared to the traditional software development process, in the long term, abstraction and automation increase cost effectiveness because of the reduced maintenance and development costs.

3.2 Ontologies and Rules

Gruber and Borst [105] define ontology as a formal and explicit specification of a shared conceptualization where a conceptualization refers to an abstract model of a phenomenon in the world by identifying the relevant concepts of that phenomenon.
Formal refers to the fact that the ontology is formulated in an artificial machine readable language which is based on some logical system like First Order Logic (FOL) [106]. An ontology refers to an engineering artifact, constituted by a specific vocabulary and set of assumptions (based on FOL) regarding the intended meaning of the vocabulary’s words [107]. Ontologies can be classified with respect to their level of expressivity into lightweight and heavyweight ontologies. A lightweight ontology includes concepts, concept taxonomies, properties and relationships between the concepts [108-109], and in the simplest case, an ontology describes a hierarchy of concepts in subsumption relationships [107]. A typical heavyweight ontology requires suitable axioms in order to express more sophisticated relationships between concepts and constrain their intended interpretation [107], and is usually composed of concepts (i.e., classes), attributes (i.e., properties), relations (i.e., slots, roles, properties), constraints, axioms (i.e., logical expressions – rules – that are always true), and functions. Different KR formalisms can be used to model ontologies which can be categorized as follows [12, 108]:

- AI based,
- software engineering (e.g., UML),
- database engineering (e.g., ER, EER),
- application-oriented techniques (e.g., key-value pairs).

Software engineering and database engineering techniques fall short for developing heavyweight ontologies. Although the expressivity of application-specific approaches differs, the main drawback is their ad-hoc nature. AI based techniques are well-suited for the development of heavyweight ontologies, since ontologies built using AI techniques constrain the possible interpretation of terms more than other approaches [108]. A KR ontology (i.e., similar to a meta-model) provides representation primitives (e.g., concepts, relations, etc.), and is built on top of a particular KR formalism to enable development of ontologies. Ontologies based on Description Logic (DL) are usually divided into two parts: TBox and ABox [110]. TBox contains terminological knowledge such as definitions of concepts, roles, relations, etc., while ABox contains the definitions of the instances (i.e., individuals). ABox and TBox together represent a knowledge base. Prominent utilities of ontologies can be summarized as follows: reduced ambiguity, knowledge share, interoperability, re-usability, knowledge acquisition, communication between human-human and human-machine, inference and reasoning and natural authoring [111-115].

In the context of this paper, inference and reasoning support holds a crucial place. It is important to decide on what is required in terms of reasoning and expressiveness before selecting the representation formalism for developing an ontology. This is because, every formalism has different level of expressiveness and reasoning support [108], and there is a trade-off between expressiveness and reasoning power [116]. In this respect, a combination of rules and ontologies is important, since rules are used for constraint checking, logical inference and reasoning, etc. Two different combinations are possible:

- to build rules on top of ontologies (i.e., rules use the vocabulary specified in the ontology),
- to build ontologies on top of rules (i.e., ontological definitions are supplemented by rules) [117].
This mainly originates from the difference in fundamental characteristics of rules and ontologies. Ontologies under the KR paradigm focuses on content (i.e., knowledge) while rules under the logic programming paradigm focus on form to arrive at logical conclusions. Two prominent examples are OWL and F-Logic respectively. OWL is a member of the semantic web family, and it is based on DL [118]. DL languages belong to a family of KR formalisms based on FOL [119]. The reasoning tasks supported in DL are subsumption and realization. Subsumption determines sub-concept/super-concept relationships of concepts occurring in a TBox, where realization computes whether a given individual necessarily belongs to a particular concept [120]. F-logic [121] is a language layered on top of logic programming and extends classical predicate calculus with the concept of objects, classes and types which are adapted from object-oriented programming [122-123]. Although F-logic is mainly used as a language in 'intelligent' information systems, it is being widely used as a rule-based ontology representation language [122-124]. There is already a line of research to improve expressive and reasoning power of OWL with rules to fill the gap with F-logic. The main drawbacks and the related work are to be presented in Section 4. Finally, we would like to mention different rule types. In [125], rules are categorized into three types:

- deduction rules,
- normative rules,
- reactive rules.

Deduction rules are used to derive new knowledge from existing knowledge (important for context reasoning) while normative rules constrain the data or logic of the application to ensure the consistency and integrity of the data and the application. Finally reactive rules (production rules and Event-Condition-Action rules) [126] describe the reactive behavior through automatic execution of specific actions based on occurrence of events of interest (important for the dynamic nature of pervasive environments). Depending on the inference engine, rules can be executed through forward reasoning (i.e., data driven) or backward reasoning (i.e., goal driven). Forward reasoning starts with the initial sets of facts and continuously derives new facts through available rules. This is crucial for adaptive and pervasive systems having a rapidly changing context space. Backward reasoning moves from the conclusion (i.e., goal, hypothesis) and tries to find data validating the hypothesis. It is important to apply an appropriate reasoning mechanism, for instance, using forward reasoning, where backward reasoning is sufficient, but will be more costly. We refer interested readers to [125] for more information. Rules, as a logic paradigm, are quite important, as they provide the capability of explaining why a particular decision is reached [115, 127-128]. This becomes possible by tracing back the inference chain of the executed rules and revealing the conditions and any intermediate data inferred during the reasoning process.

### 3.3 A Merged Approach

Ontologies can be used for diverse engineering purposes such as for formalizing engineering activities [129] and artifacts. Our focus is on software as an engineering artifact. There has been a considerable debate on the formal/informal form of software
specifications [130] gathering structural knowledge (about the components which comprise the design object and their relations), behavioral knowledge (about the behavior of the design object), teleological knowledge (about the purpose and the way the design object is intended to be used), and functional knowledge (about the behaviors and goals of the artifact itself) [131].

Ontologies are particularly of use for complex software while MDD is an appropriate approach for developing large scale systems and applications. Since Pervasive Computing and Adaptive Systems (highly complex and large scale) are an integral part of tomorrow’s computing, amalgamation of ontologies and MDD is of crucial importance. Such a marriage seems to be possible, since both approaches employ a similar paradigm, that is, abstraction. Therefore, it is not surprising to see that, in current literature there are several studies either employing ontologies – particularly OWL and OWL KR – as a modeling formalism in MDD [16, 104, 111] or employing MDA modeling instruments - particularly UML, the UML meta-model and OCL – as a representation formalism to develop ontologies [78, 108, 132-135]. However, such approaches do not exploit the full benefits of the abstraction. On the one hand, using UML, the UML meta-model and the Object Constraint Language [136] (OCL – used to increase expressivity of UML through allowing constraints to be defined) for ontology development is not preferable, since they do not offer automatic inference, and there is no notion of logic and formally defined semantics [19, 127, 137-138]. Available AI-based KR formalisms or logic programming languages (i.e., F-logic, OWL, etc.) are preferable due to their links between DL and dynamic logic [139]. In [140], the authors remark that ontology languages support building axioms, inference rules and theorems, which form a logic theory, but UML does not provide such support. On the other hand, without aiming at employing reasoning support of ontologies in terms of consistency checking, validation, and prominently for run-time reasoning and dynamic adaptations, use of more expressive ontology formalisms in MDD will only introduce higher complexity where the limited expressivity and tool support based on UML should already be sufficient.

In [16], by giving valuable insights, the author points out potential benefits of using domain models not only for code generation but also as executable software artifacts at run-time, without providing an elaborated discussion of these benefits and a possible methodology. The World Wide Web Consortium (W3C) and OMG, the main organizations behind the semantic web and MDA respectively, are already aware of the significance of using knowledge and tools available in each field. Several initiatives have already been started in this direction, for instance, Ontology Definition Meta-model (ODM) of OMG (see http://www.omg.org/ontology/) for developing OWL ontologies through UML, and Ontology Driven Architecture (ODA) of W3C (see http://www.w3.org/2001/sw/BestPractices/SE/ODA/) for outlining potential benefits of the semantic web for system and software engineering. Note that previously mentioned approaches using UML and OCL for ontology development should be considered apart from ODM, since UML meta-models used by these approaches do not produce OWL ontologies, while ODM allows development of expressive OWL and RDF ontologies through incorporating and visualizing OWL and RDF KR constructs. In [107], the author considers the use of ontologies in information systems in twofold, from a temporal perspective, that is, ontologies for
information systems (i.e., design time), referred as “ontology driven development of information systems” and ontologies in information systems (i.e., run-time), referred as “ontology driven information systems”. In [111], the authors elaborate on the use of ontologies in Software engineering from various points, by covering the temporal perspective suggested by [107] as a core. The authors point out that, for development-time, a set of reusable ontologies can possibly be organized in libraries of domain or task ontologies, and the semantic content of the ontologies can be converted into a system component, reducing the cost of analysis and assuring the ontological system’s correctness; on the other hand, for run-time, ontology can be considered an additional component (generally, local to the system) which cooperates at run-time to achieve the system’s goals and functionality. The authors provide a review of the related work for both cases, however, use of ontologies at run-time and at development-time is considered in isolation from each other, and the reviewed work follows the same line. This is mainly due to the characteristics of R&D projects, i.e., with software engineering goals and not knowledge engineering goals or vice versa. In [16-18, 20], the authors favor a merged approach, which we are interested in, by employing ontologies at run-time and development-time, that is:

- ontologies as a KR formalism for deriving models of systems and applications to automate the development of the complex software,
- ontologies as a logic-based formalism for run-time reasoning, inference and dynamic adaptation.

Furthermore, with the availability of expressive rule languages employed on top of ontologies, the bigger part of the application logic will be represented in formal declarative models [16]. However, realization of such an approach is not trivial, and this is mainly because of the fundamental differences available between logic and KR paradigms, e.g., expressivity vs. decidability, which will be further elaborated in the following sections along with the practical aspects of the approach. Although, in some recent projects, links between MDD and ontologies are highlighted and said to be exploited, e.g., in the MUSIC project [141], ontologies primarily have been used as a conceptual basis for MDD rather than as a direct input for automated development processes. Therefore, design and development of full applications by employing ontologies throughout the whole software development cycle is not realized.

One of the main concerns raised regarding the use of ontology as a central substance for MDD is that while UML provides means to specify dynamic behavior of the system, current OWL-based approaches do not [19, 137-138, 142]. The ability to model dynamic behavior of a system is crucial for the automated development of pervasive and adaptive systems and applications, since behavioral models (including constraints imposed) are central to the adaptation process. UML’s ability to specify dynamic behavior leads researchers to investigate means of combining power of the ontologies and UML.

In [19], a approach called TwoUse, which integrates UML and OWL to exploit the strengths of both paradigms, is introduced. Integration is mainly from OWL to UML through increasing expressiveness of OCL by using SPARQL-like expressions using ontology reasoning, so that UML/OCL developers do not have to enumerate actions and constraints class by class. That is, classification of complex classes remains in OWL (intertwined with OCL) and the specification of the execution logic remains in UML. TwoUse integrates, by composition, the OWL2 meta-model to describe classes
in a higher semantic expressiveness and Class::Kernel of the UML2 meta-model to describe behavioral and structural features of classes. TwoUse employs profiled class diagrams for designing combined models. The aim is to transform models, conforming to TwoUse meta-model, into application code as well as to OWL ontologies.

In [138], the authors present how processes modeled with SPEM (the Software & Systems Process Engineering meta-model of OMG [143] based MOF and UML) can be translated into an ontology to exploit the reasoning power of ontologies. The authors use Semantic Web Rule Language (SWRL, extends OWL with logic based rules, i.e., logic layer) to check project constraints and to assert new facts from existing data. The translations are not used to substitute the original SPEM models, but are used to complement the original models with reasoning support. In the context of SPEM, the process does not refer to dynamic behavior of a system, but to a set of activities, methods and practices which people use to develop and maintain software and associated products. Nevertheless, the work still remains relevant in the sense that it demonstrates an example of transformation from models involving dynamic behavior to ontologies.

Although the combination of UML and ontologies unites the power of formal semantics (i.e., validation of structure and semantics, and reasoning) and the ability to model dynamic behavior of a system, the literature points out that UML’s lack of a formal ground disables possible use of advanced analysis and simulation methods on behavioral properties of the model. One response to this consideration is the integration of Petri nets [144], particularly high level Colored Petri Nets (CPN) [145], to the MDD process [137, 146]. Petri nets are a graphical and mathematical modeling tool providing the ability to model, simulate and execute behavioral models; they are a sound mathematical model allowing analysis (e.g., performance), validation, and verification of behavioral aspects of systems (e.g., liveness, reachability, boundless, etc.) at design time. This is quite appealing for complex pervasive and adaptive systems; for instance, the validation of the liveness property guarantees a deadlock-free system behavior. Due to the hierarchical structuring mechanism of CPN, it becomes easier to design complex systems in terms of modules and sub-modules. Efforts towards intertwining capabilities of Petri nets with ontologies and MDD have already emerged.

In [146], the authors describe a Petri net ontology (corresponds to a Petri net meta-model), based on OWL and RDF, to be able to share Petri nets through the semantic web. The proposed ontology allows semantic descriptions of Petri net concepts and relationships (place, transition, arc, etc.) and allows semantic validation of Petri net models against the Petri net meta-model (i.e., ontology). The authors first review existing Petri net presentation formalisms (e.g., PNML – Petri net Mark-up Language), which are mainly tool-specific, to extract required concepts, attributes, etc. The authors opt to use UML for the initial development of the Petri net ontology (i.e., through a UML profile for ontology development) and further refine their ontology through the Protege ontology editor after importing ontology through Protege’s UML backend. The authors manually reconstruct OCL constraints into corresponding Protege PAL (Protege Axiom Language) constraints. Regarding visual development of Petri net models, a tool named P3 is used. P3 has the ability to export Petri net models in RDF with respect to proposed the Petri net ontology.
In [137], the authors introduce a methodological framework called AMENITIES employing UML, OWL, and CPN to advance modeling and analysis of collaborative systems. The methodology is based on a collaborative process domain ontology allowing representation of processes (i.e., behavioral aspects) and relevant entities along with their relationships (i.e., structural aspects). Collaborative processes are modeled through UML-based notations and validated against this domain ontology; this has been enabled through provision of mappings from UML to OWL. The UML-based design is preferred due to UML’s human-friendly visual representation formalism covering structural and behavioral constructs. The mapping from UML to OWL results in the formalization of process models, and hence, enables ontological reasoning and validation. A mapping from ontological entities involved in description of behavioral aspects of the process model to the entities in CPN meta-model is also defined to exploit advanced behavioral analysis, validation and verification properties, and simulation power of the CPN.

According to the aforementioned approaches, the most prominent properties that are expected from a model are summarized in Table 1 along with the comparison of support given by OWL-based approaches, UML, Petri nets and their combinations. Although we are aware that these paradigms are distinct in terms of their purposes, the comparison is at abstraction level. The use of UML is primarily due to its user-friendly and standardized graphical representation constructs within the scope of the presented approaches. Although Petri nets and OWL development tools also provide visual constructs, they are quite generic while UML’s visual notation is specific and can be customized for particular domains. The literature reflects that combination of three paradigms is quite fruitful; this combination is highly important for design, development, analysis, validation, and verification of pervasive and adaptive systems.

Table 1: Comparison of modeling paradigms with respect to three prominent properties.

<table>
<thead>
<tr>
<th></th>
<th>OWL</th>
<th>UML</th>
<th>Petri Net</th>
<th>UML + OWL</th>
<th>UML + Petri</th>
<th>OWL + Petri</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extensible visual constructs</td>
<td>~</td>
<td>✓</td>
<td>~</td>
<td>✓</td>
<td>✓</td>
<td>~</td>
</tr>
<tr>
<td>Reasoning and semantic validation</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Behavioral analysis and validation</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

The main question is on a possible methodology for such a combination. Two prominent methodologies can be identified. The first one follows the common approach presented by the aforementioned works, that is, using each modeling paradigm for a specific purpose – i.e., UML for visual design (while respecting UML semantics), OWL for semantic validation and reasoning, and Petri nets for behavioral analysis and validation.
analysis, validation and verification. This approach requires defining mapping schemas, and realizing transformations of models from one to another at each step. However, on the one hand, maintaining this distributed process along with these mapping schemes is a complex process and bi-directionality of the transformations should be guaranteed. On the other hand, the initial ontology is not expressive enough, since semantic expressivity of UML is limited. In this first approach, ontologies are not meant to be a base for the original models, but are used to complement the original models with a limited reasoning support. Hence, the second approach (see Figure 6) is based on using OWL KR as an underlying representation formalism and rebuilding the meta-model of each paradigm on top of OWL KR and its logic layer.

![Diagram](image)

**Figure 6:** An integrated abstract development environment based on OWL KR and logic layer.

A logic layer is required, since it is not possible to realize every constraint within OWL KR. This is because OWL KR only provides general axioms and representation primitives. However, not every specific construct of a meta-model needs to have its semantic correspondence in an OWL ontology (and cannot due to decidability constraint) (for instance, it is possible to describe structure and state of a Petri net with OWL), but its behavioral semantics should be interpreted by subject-specific engines (i.e., separation of concerns) which support the designer with custom visual constructs matched to the subject-specific classes and properties (e.g., places and transitions for Petri nets) in the ontology. Since the underlying model and representation is ontology-based, such subject specific engines can exploit the reasoning power of ontologies, for instance, the expressiveness of CPN guards (i.e., expressions representing branching constraints in a CPN model) can be enhanced. This also applies to UML class diagrams. Not every construct can be represented with its underlying semantics, for instance identification and functional dependencies. Although similar to the CPN example, such constraints can be represented in terms of classes and properties, but their interpretation remains to be done by the relevant UML engines.
However, the first and second approaches are indeed not truly merging an ontology driven and model driven approach. They do employ expressiveness and capabilities of ontology representation languages for models. The third approach, see Figure 7, follows a natural authoring mechanism. The development starts with identification of related concepts, properties, relations, etc., of the application domain without considering the notion of software at all (e.g., a concept does not represent a software class, but a real world phenomena as it is), and only focusing on what exists. Once the target phenomena is conceptualized and formalized in terms of an ontology, specific transformations can be used to transform parts of ontology to specific models, for instance, structure of an ontology can be transformed to a UML class diagram (which is not only different in terms of visual notation, but new constructs will also appear, e.g., use of Java interfaces for multiple inheritance, based on various patterns). Such an approach allows iterating from natural representations to specific representations of the domain [98, 147]. The overall approach is based on the understanding that ontologies are boarder than models in terms of semantics and the reality they describe [148], and ontologies are always backward looking (i.e., descriptive: describe what already exists) that in a real world is described with the concepts of the ontology, while models are mainly forward looking (i.e., prescriptive: prescribe a system that does not exist, and reality is constructed from it), that is, objects of a system’s elements are instances of the model elements [88, 149]. Ontologies are primarily used to describe domains [129] while models are used to prescribe systems [98]. Although there is an ongoing discussion on distinguishing between models, meta-models and ontologies (in the literature, several authors directly investigate comparisons of ontologies and meta-models as well as comparisons of ontologies and models [88]) from a philosophical perspective, since at this stage we are more interested in practical issues we refer interested readers to [88, 98, 149-150].

![Figure 7: An objective-based merger of model-driven and ontology-driven approaches.](image)

Regarding a possible methodology towards integrating ontologies and MDD based on the third approach, the following has been proposed in [12, 17], shown in Figure 8, for the development of adaptive and pervasive computing systems and applications (an informative methodology rather than a normative one). Note that the proposed methodology is adopted from MDA [90]:

- define computing independent domain ontology (CIDO) and a Generic Ontology (GO), and convert part of the CIDO together with GO to Context Ontology (CO),
- convert part of CIDO to Platform Independent Application Model (PIAM) and define platform specific annotations and mapping techniques,
- convert PIAM to Platform Specific Application Models (PSAM) and Artefact Dependent Models (ADM, e.g., Database Schema), and define mapping techniques and annotations,
- apply conversion of PSAM(s) and ADM(s) to application codes and software artifacts.

**Figure 8:** A possible methodology merging MDD and ontologies.

In the given procedure, some parts of CIDO are converted into PIAM and some parts are converted into CO, this is because some knowledge included in CIDO might not necessarily be needed to map knowledge of the application, but rather the reasoning logic of the application which is required to be used by the application externally, and in the other way around some knowledge included in CIDO might not necessarily be needed to map contextual knowledge, but rather the internal structure of the application. At this point, it is important to clarify that the mapping process defines the conversion between the constructs (i.e., primitives) of KR ontology and meta-model of the software artifact, so that the transformation of the given ontology to the specific software artifact can be realized.

Development-time use of ontologies is highly undermined in the current literature of adaptive and pervasive computing, and ontologies are solely used for reasoning purposes [32-33, 151-154]. A recent approach [155] takes steps toward employing MDD for automated code generation, and ontologies for run-time reasoning. The proposed approach is based on a UML based meta-model (PervML) for modeling
pervasive computing systems on which automated code generation is based. The
behavioral aspects of the system are modeled through state transitions diagrams (non-
executable). An ontology for PervML is manually developed, and exchange/transition between the PervML model and ontology is realized. Although the proposed work puts an important effort through combining run-time reasoning and automated code generation, the ontology is developed based on a UML model, and hence, is limited in expressivity. The mapping between the model and ontology is a manual process which is problematic in terms of redundancy. Finally, the approach misses the possibility of analysis, validation, and verification of behavioral system properties due to insufficient formality of the modeling paradigm used.

A truly merged approach is expected to inherit major benefits of the MDD and ontologies and to clearly address run-time and development-time concerns mentioned earlier. Considering the design time, a merged approach is supposed to enable:

- consistency checking and the validation and verification of the software, which is not sufficiently addressed in MDA currently [104],
- rapid, sustainable, high quality, and cost-effective development through increased re-usability, portability, understandability, documentation power (i.e., up-to-date, unambiguous and formal), consistency, and reduced maintenance efforts and ambiguity (by providing a consistent framework for unification of concepts and terminology [113, 156]),
- a smooth requirements elicitation and modeling phase [157], and preservation of constructed knowledge which is ready to be reused or to be shared.

Considering the run-time, it is supposed to enable:

- increased manageability through explicit management of dynamic adaptations (even by end-users), and the application logic,
- increased user acceptance, trust and engagement based on improved communication between end-users and the software, since ontologies enable intelligible applications:
  a. to explain the reasoning logic of the system decisions through giving the relevant feedback and the guidance [115, 128],
  b. to enable users to mediate system decisions through the given feedback,
  c. to aid interaction between the users and the environment, since they concisely describe the properties of the environment and the various concepts used in the environment [158].
- alleviation of imperfectness through consistency checking of the context,
- interoperability at semantic and syntactic levels [111].

The combination of ontologies and MDD enables automated development of adaptive and pervasive systems at different architectural layers. At this point, it is appropriate to examine a possible architecture for such systems and applications. In [111], the authors point out that a typical ontology-driven information system consists of a knowledge base, formed by the ontology and its instances, and an inference engine attached to this knowledge base, and there are numerous types of proposals for such systems. Each of them shares a great similarity and varies according to the application domain. Therefore, a layered view of such systems seems to be more
appropriate in this context. The following layers are supposed to be included in a
typical architecture of an adaptive and pervasive software – adopted from [12, 32,
159] – at a conceptual level:

- sensing layer provides means to acquire contextual information through
  physical and virtual sensors,
- data layer provides means to store application data involving the contextual
  information (i.e., for procedural computations or scalability matters, to be
discussed in Section 4), e.g., in relational databases,
- representation and reasoning layer (declarative) accommodates acquired
  contextual information and generic and domain ontologies, infers new
  contextual information and provides reasoning facility, i.e., ontologies and
  rules,
- dissemination layer enables exchange and access of contextual information
  through push or pull [12, 160] based mechanisms,
- application layer (procedural) queries the data layer and the representation
  and reasoning layer, and manages adaptation processes, sessions, user
  interfaces, etc.

Considering target meta-models (i.e., in the solution domain), apparently, it is an
appropriate decision to use an object-oriented language, (e.g., Java) for the procedural
body of the software and relational or object relational database systems for the
storage, because of their wide acceptance, proven success and existing similarities
with OWL (i.e., better mapping and transformation results). Section 4 presents the
existing work concerning the mapping rules and procedures for transformation of
OWL ontologies to Java source codes and database schemas. Please note that there
might be (and should be) transformations to intermediate models as Figure 7 suggests;
however, the existing work elaborating on direct transformations from ontologies to
software artifacts is also an indicator of possible problems and challenges, since such
intermediate models are derived from the ontology and gradually approach the target
application.

4 Practical Grounding

F-logic [121], under the logic programming paradigm, and OWL [161], under the KR
paradigm, are the most prominent formalisms within the context of ontology
development. F-logic integrates logic and object-oriented programming in a
declarative manner [122], while OWL, from the semantic web family [162], targets
ontologies for the Web. F-logic has already been used successfully for ontology
modeling, software engineering, agent-based systems, etc. [122]. Although F-logic is
based on object-oriented primitives [123], and has strong links with logic
programming, and availability of mature development environments, notably
Ontobroker (already commercial), TIPLE, Florid, and Flora-2, which are valuable
assets for our approach (i.e., better mappings from representation formalism to object-
oriented meta-model); we prefer to focus on OWL from now on. This is because of
the availability of adequate support for integration with the Web. Web integration
[108] holds an important role in Pervasive Computing because the Web is expected
to be the main application space for pervasive software, main information and data space for storage and exchange (including contextual data), and main medium of communication between the applications and the ambient devices [34].

4.1 The Semantic Web: Logic and Rule Layers

At the OWL site, everything is not perfect, since the logic and rule layer of the semantic web is still a work in progress [117, 163-164]. In this section, existing problems and related work will be introduced in two parts:

- reasoning support for OWL and integration of rules and OWL,
- transformations of OWL ontologies to the application artifacts (mainly to object-oriented application code and database schemas).

The ontology layer is the highest layer that reaches a sufficient maturity in the famous semantic web Layer Cake illustrating semantic web architecture [117]. OWL is divided into three layers with increasing expressivity, namely, OWL Lite, OWL DL and OWL Full. OWL Lite, with a lower formal complexity, is good for work requiring a classification hierarchy and simple constraints, while OWL DL provides higher expressivity and guarantees that all conclusions are computable (i.e., computational completeness) and finish in finite time (i.e., decidability) [161]. OWL Full OWL provides maximum expressivity, but computational guarantees do not exist, since it has non-standard semantics [123]. In particular, decidability is an important criterion, since complete and undecidable algorithms can get stuck in infinite loops [124]. In this respect, OWL DL stands as an optimum choice for most of the applications of adaptive and pervasive systems. This is the case even with respect to its logic programming-based alternative, F-logic, which is not decidable [165]. However, OWL DL has some particular shortcomings, since the utility of ontologies, in general, are limited by the reasoning and inference capabilities integrated with the form of representation [32, 166]. It has been already known before that it is required to integrate logic programming with OWL, thus rules and ontologies, to overcome the limitations of the OWL [125]. This is the central task in the current research [32, 117, 167-168]. OWL DL is based on DLs, with an RDF syntax [169], which can be considered as a decidable fragment of FOL [124] (i.e., a DL knowledge can be equally translated into FOL [123]). FOL follows an OWA and employs monotonic reasoning, while logic programming follows the CWA and allows non-monotonic reasoning [32, 108, 119]. Several reasons can be listed for integrating logic programming and OWL – extended from [123]:

- **Higher relational expressivity**: The basic primitives provided by OWL DL for expressing properties are insufficient and not well-suited for representing critical aspects of many practical applications. OWL DL can only express axioms which are of a tree structure, but not arbitrary axioms [170, 171]. Therefore, it is not possible to construct composite properties by exploiting relationships between available properties such as construction of ‘uncle’ property through composition of ‘brother’ and ‘parent’ properties [119].
- **Higher arity relationships**: OWL DL supports unary and binary predicates to define concepts and properties respectively, however, higher order arities are only supported as concepts [108]. However, in practice higher arity
predicates are encountered, such as ‘connect’ property, which is ternary, stating that a road connects two cities [108, 123],

- **CWA**: CWA considers statements that are not known to be true, as false (i.e., negation as failure) while OWA, in contrast, states that statements that are not known to be true, should not be considered false. In [123], the author points out that a closed world querying can be employed on top of OWL without a need to change semantics for the applications requiring closed world querying of open world knowledge bases,

- **Non-monotonic reasoning**: OWL assumes monotonic reasoning, which means new facts can only be added but not retracted and previous information cannot be negated because of the new information acquired [108]. However, adaptive and pervasive systems require non-monotonic reasoning, since the dynamic nature of the context requires retraction and negation of the existing facts [32],

- **Integrity constraints**: Integrity constraints, which are non-monotonic tasks, cannot be realized in OWL [123, 172] due to incomplete knowledge originating from the underlying OWA,

- **Exceptions**: Exceptions are unavoidable in real life; a well-known example is that all birds fly but penguins are exceptions.

One of the first responses for the integration of rules and ontologies is SWRL [173] which combines OWL and Rule Markup Language (RuleML) [174] (i.e., an XML based mark-up language for rules [117]), however, it does not support non-monotonic reasoning and it is undecidable [119, 125]. There are several reasoners supporting SWRL such as KAON2 and Pellet (for known decidable fragments – i.e., DL safe – of SWRL [171]), RacerPro (uses a SWRL like syntax and supports closed world semantics) [175]. It is worthwhile mentioning the Jena 2 semantic web framework [176] which is used to create semantic web applications. It does not use SWRL but employs its own rule language and supports monotonic and non-monotonic rule formalisms, and backward and forward chaining. It realizes a weak negation (i.e., negation as failure) through providing an operator only checking the existence of a given statement and provides an operator to remove statements. We refer to [117, 119, 124, 163-165, 171, 175, 177-179] for the current research towards improving OWL with the expressiveness of logic programming for interested readers. Similar to the standardization of the ontology layer of the semantic web through OWL, rule layer also needs to be standardized, to enable use of ontologies and rules for innovative applications [166] and exchanging rule based knowledge [125]. W3C already initiated a working group for developing a standard exchange format, namely Rule Interchange Format (RIF) [180], for the rules. In [125], the authors remark that the development of RIF includes two phases, where the first phase includes realization of stable backbone and the second phase includes extensions with first order rules, logic programming rules, production rules, etc. We refer [125, 180-181] for further details, syntax, and semantics.
4.2 The Semantic Web: Ontologies to Software Artifacts

Existing work in the literature for transforming ontologies to software artifacts, including application code, is of great use for the proposed approach in terms of tool support. In [182], the author introduces two cross compilers, namely, OntoJava and OntoSQL to realize the automatic generation of Java- and SQL-based inference engines. The former one converts RDFS and RuleML into sets of Java classes, while the latter one maps recursive and non-recursive rules into SQL-99. In [183], the author first describes the main similarities and differences between object-oriented and ontology-oriented paradigms. Afterwards, the author introduces a compiler which produces a traditional object-oriented class library, for the .Net language family, from ontologies. In [184], the authors focus on OWL, which is a more expressive DL than RDFS, by building on the existing work of [182], and describe how to map OWL into Java interfaces and classes. The authors remark that such mapping is not expected to be complete, because of the semantic differences between DL and object-oriented systems. However, the authors aim at mapping a large part of the richer OWL semantics through minimizing the impact of such differences. The mapping involves basic classes, class axioms (e.g., equivalent class, sub class), class descriptions (e.g., union of, complement of etc.), and class relationships (including multiple inheritance) realised through Java interfaces; set and get methods for accessing values of the properties of the classes realized through Java beans; property descriptions (e.g., inverse functional, symmetric, transitive, etc.), property relationships (e.g., equivalent property, sub property, etc.), property restrictions (e.g., cardinality, etc.) realized through constraint checker classes registered as listeners on the properties; and property associations (e.g., including multiple domains and ranges) realized through Java interfaces and listeners. In [185], a tool called RDFReactor (available at http://semanticweb.org/wiki/RDFReactor), which transforms a given ontology in RDFS to Java API based on type-safe Java classes, is described. These classes act as stateless proxies on the RDF model, thereby enabling the developers to develop semantic web applications in a familiar object-oriented fashion. The code generation process provides:

- support for both RDFS and OWL,
- generation of full documentation for the API through JavaDocs,
- realization of multiple inheritance in a type-safe manner,
- realization of cardinality constraints checked at runtime.

The implementation of the RDFReactor is based on the Jena framework. An abstraction layer, based on various adaptors, for triple stores is also developed, to prevent any dependence on a particular triple store. It is remarked that, compared to the [185], the earlier work of [182] lacks some basic features, such as multiple inheritance and properties with multiple domains and in [184] the OWL type system is only supported through raised exceptions. Considering the relational databases, in [186] and [187], the authors enumerate possible means for the persistent storage of ontologies and instances such as flat files, object oriented, object-relational, and special purpose database systems tailored to the nature of the ontology formalism (i.e., triple stores). Scalability is one of the main disadvantages of the flat file approach. Relational database systems offer maturity, performance, robustness, reliability and availability which are their significant advantages over object and
Design and Development of User-centric Pervasive Software

object-relational database management systems [187]. In [186], the authors present a set of techniques to map OWL to the relational schema, thereby enabling applications to use both ontology data and structured data. In [188], the authors describe a set of algorithms, based on the work of [186], mapping OWL (OWL Lite and partially OWL DL) to relational schema, thereby enabling transformation of domain ontologies to relational database tables, attributes and relations. An algorithm for each of the following transformation tasks is provided:

- **OWL classes and subclasses**: A breadth first search is applied to transform classes into tables and to create one-to-one relationships between the classes and their subclasses.
- **Object properties to relational database**: A breadth first search is applied to transform object properties into relations by considering cardinalities (i.e., one-to-one, one-to-many etc.) and property hierarchy (i.e., sub properties).
- **Datatype properties**: Datatype properties are transformed into data columns in their respective tables, matching the domains of the properties.
- **Constraints**: A breadth first search is applied to transform constraints into meta tables specific to the type of constraint (i.e., cardinality, domain, range etc.).

In [187], the authors refer to the related literature for OWL to the relational schema mapping (e.g., [186, 188-189]) and claim that they suffer from one or more of the following problems:

- restrictions are ignored,
- not implemented,
- semi-automatic,
- structure loss is not analyzed.

The authors propose an elaborate list of mapping rules specifying the mapping from the OWL to the relational schema involving classes, class hierarchy, datatype and object type properties, and value restrictions (using SQL CHECK constraint) for datatype properties and data type conversions. The authors note that not all the constructs in an ontology map to relational schema, and their solution maps all constructs except those constructs which have no correspondence in the relational database (an exhaustive list of constructs is not given). In [190], the authors propose a hybrid approach where ontology classes and properties are mapped to a database schema and instances are stored in database tables, while more complex constructs that cannot be adequately represented by database concepts are stored in metadata tables. In [191], the authors demonstrate how a programming interface can be generated through translating an OWL/RDF knowledge base into Java beans and hibernating object-relational mappings for persistent storage of content of the classes – i.e., OWL individuals (rather than using triple-store approach which is powerful but unconventional). Authors employ Java interfaces for classes, class relationships, etc. (including multiple inheritance) by following the aforementioned work. Mappings from OWL properties to Java beans, table attributes, and relations are described in terms of literal properties (i.e., data attributes) and object properties (i.e., relations).

- Mapping of literal properties involves:
  - functional or single cardinality literal property (i.e., cardinality restriction is equal to one) representing a one-to-one relationship,
b. multiple cardinality property (i.e., cardinality restriction is more than one) representing a one-to-many relationship.

- Mappings of object properties involves:
  a. non-inverse functional object property representing one-to-one unidirectional relationship,
  b. non-inverse object property representing many-to-many unidirectional relationship,
  c. functional property inverse of a functional property representing one-to-one bidirectional relation,
  d. functional property inverse of a non-functional property representing one-to-many bidirectional relationship,
  e. object property inverse of a object property representing many-to-many bidirectional relationships.

There also exist several efforts towards the opposite direction, that is, transformations from software artifacts to ontologies (e.g., relational databases to ontologies [192]). Although this is not the focus of this paper, such efforts are helpful to reveal the semantic gap between the software-related modeling formalisms and ontologies, and for moving the existing applications to an ontology-based framework by reusing the existing application knowledge.

Although some basic tools for automating OWL transformations to application artifacts have been developed by aforementioned works, they are not sufficient. Semantic differences (e.g., DL vs. object oriented, etc.) between meta-models needs to be further investigated and should be defined exhaustively. The success of the proposed approach is mainly based on the completeness and quality (e.g., structure loss, data loss etc.) of transformations done. Completeness also needs a redefinition in this context. This is because, not all the constructs are required to be mapped, since some constructs might only be needed for reasoning purposes and should only be accessible through a knowledge base. Therefore, a complete mapping does not include all available constructs, but only the ones required for data access and preservation in the data layer, leaving the complex constructs required for reasoning and high level KR to the representation layer. Hence, the identification of necessary constructs and a complete evaluation of the mapping and transformation processes, which is beyond the scope of this paper, are required.

5 Discussion

MDD, the semantic web, and Adaptive and Pervasive Computing are still subjected to extensive discussions regarding whether or not it is possible to fully realize the given promises. We believe that they have a considerable potential to contribute to each other’s development. The strong focus of business on short term return of investment prevents long term investment of software development methods and tools [193]. In [194], the authors point out that organizations managing their processes with ontology-enabled tools and methods would benefit from a flexible infrastructure prepared for inference and partial automation of processes. In [195], the authors further claim that in the future software will not be designed without using an
ontological approach, especially when adequate tools are available. The high complexity introduced by adaptive and pervasive systems makes it necessary to use more elaborate and systematic development approaches capable of development and management of long-lived ‘intelligent’ systems. The amalgamation of ontologies and MDD becomes an important response in this respect, because of the reasons detailed and discussed through this paper from different temporal perspectives, i.e., design time and run-time, in terms of management, design and development, consistency, and use.

Although an awareness on the potential benefits of using ontologies and MDA tools together has appeared in respective communities to some extent, mainly in terms of run-time reasoning and automated application development, the use of ontologies in adaptive and pervasive computing still remains limited to reasoning purposes, by omitting the already exploitable benefits of automated software development. Even such limited use of ontologies, in terms of reasoning, is not well addressed, since existing works based on ontologies do not report any large scale deployments, and there is only a small amount of work reporting quantitative evaluation [32]. Although different generic and domain ontologies and software architectures are proposed (e.g., for smart rooms), which is only a good share of practice, since every system has its own requirements demanding redesign and development, the literature still lacks the following contributions:

- systematic approaches for design, development and evaluation of adaptive and pervasive systems and applications,
- an elaborate analysis of the rules used in these systems and applications, i.e., how complicated they are, to which extent they reflect the real life uses, etc.,
- analyses reporting to which extent these systems and applications suffer from the immaturity of the logic layer of the semantic web (as described in Section 4),
- availability of large scale deployments, and proofs reflecting that existing proposals can scale well for large deployments,
- availability of futuristic and realistic scenarios (e.g., Pervasive Computing is not limited with smart rooms and with its variations).

User aspects (e.g., involvement, acceptance, etc.) are insufficiently addressed, although even an ontological approach itself alone has important potential in that sense (e.g., intelligibility, self-expressiveness, situation awareness, user control, etc.). The literature also lacks real user tests; a limited number of studies include user involvement. However, they are mostly limited to the user mediation, although an important body of knowledge on user involvement in terms of user control, feedback and guidance is already available in related disciplines. User assessment and user involvement for such systems are important, more than ever, since such systems are expected to immerse our lives and take partial, maybe all, control of it. Hence, the need for large scale deployments with user involvement and assessment addressing user engagement, acceptance and trust is evident. A significant contribution on this matter would involve:

- an empirical study reporting the importance and effects of user involvement in adaptive and pervasive computing systems and applications,
- an answer to what extent we can exploit ontologies for user involvement (i.e., user control, self-expressiveness, situation awareness, feedback/guidance, mediation, etc.),
- investigation of appropriateness of available human-computer interaction approaches and methodologies for design and management of complex user-machine interactions in adaptive and pervasive systems.

It is clear that, the development process through abstract models is more complex (as of now), but it is unavoidable. Therefore, it is important to demonstrate the entire benefits of such an approach, to convince practitioners. In this respect, as already mentioned, an approach using automated code development and reasoning support of MDD and ontologies is a quite good incentive. However, adequate tool support for design and development is crucial. Apparently, for a typical developer it will be quite hard to understand and work with complex KR tools and constructs. Therefore, high visual support and familiar development constructs and tools are required. A few studies have already been made to use UML as a visual development instrument for OWL ontologies, such as [196-197]. However, at this point, the OMG initiative of ODM provides an important contribution by allowing visual development of OWL ontologies through UML. This is also important in terms of standardized visual representation and exchange of ontologies by considering different non-standard visual representation formalisms used in the literature. Tool support is already available for ODM, for instance, UML2OWL (see http://protegewiki.stanford.edu/index.php/OWL2UML) plug-in for Protege visualizing OWL ontologies using ODM, one ATL (a model transformation language and toolkit) implementation (see http://www.eclipse.org/m2m/atl/usecases/ODMImplementation/), etc. We refer interested readers to the ODM documentation [198] for further information and available tool support.

Considering the methodology and development, ontology formalism and languages (AI based) have the power of formal semantics and are more expressive than the ones used for model development, while modeling formalism and languages provide extensible visual support and are easier to use. Although there are some distinctions between modeling and ontologies in practice (ontologies are based on commitment, and follow an OWA, in contrast, models are mostly based on CWA, etc.), in the literature from the abstraction point of view, the increased use of OWL for different modeling subjects reflects that expressive formalisms and languages are required to be employed as a basis for the development of ontologies and models. The literature also reflects that subject specific visual support is an important criterion affecting decisions in favor of usability in functionality (expressiveness in this case) vs. usability dilemma. Therefore, the use of expressive ontology formalisms for conceptual development (for ontologies and models) with the possibly to use subject-specific interpretation engines and visual notations (i.e., profiles) and to choose between CWA or OWA and monotonic or non-monotonic reasoning is of crucial. However, apart from the challenge of arriving easier to use and expressive abstraction formalisms and languages, an approach truly merging ontologies and MDD requires first capturing the broad application domain with its semantics, without any software engineering concern, and then gradually approaching to target application (or software artifact(s)) by iterating through intermediate models with increased
concreteness and decreased semantics. The ontology derived at the first step is also of use for a formal requirement elicitation and analysis process, as well as for validation and verification and consistency of the intermediate models. The approach provides a natural authoring process, while also exploiting ontologies as run-time artifacts for the reasoning purposes.

Limited research and tool support exists for automated transformations of OWL ontologies to the application artifacts (hence to the intermediate models). Available work is not elaborate and does not provide sufficient criteria to ensure consistency and accuracy of the transformations. Elaborate studies through real use cases should be implemented to reveal what intermediate models are required and what constructs should be transformed at each stage. The existing efforts for directly transforming ontologies to the software artifacts (e.g., java code, database schema, etc.) by skipping any possible intermediate models are important, since they reflect the ultimate semantic gap between the initial artifact (e.g., ontology) and the final artifact (i.e., the software artifact). A very remarkable observation is that, in the literature, there are efforts towards mapping broad semantics of ontologies to the software constructs, hence enabling transformation of expressive semantics to the software artifacts (e.g., [182, 199-200]) to try to convert rules and axioms to the application domain rules in the form of SQL triggers, Java, etc.); however, not every ontology construct is required to be transformed, since a part of them will be only required for reasoning purposes. Furthermore, software specific constructs and constraints should not be included in the ontology, this is because, on the one hand, it breaks the natural authoring chain, and on the other hand, trying to model every construct related to the software might break the decidability of the ontology. For instance, number restrictions (e.g., cardinality) cannot be defined on non-simple roles (e.g., transitive) [201]. Therefore, software specific constraints should be left to the intermediate models (e.g., conceptual schemas) for data access, preservation, and integrity in the data layer, and complex constructs required for the reasoning and high level KR should be left to the ontologies in the representation layer [148].

## 6 Conclusion and Future Work

Clearly, Adaptive and Pervasive Computing has been changing the computing paradigm and the way people interact with technology. The software will be more complex and long-living, thereby requiring growing amounts of revisions. Systems and applications will be subject to considerably increased amount of contextual information and will be expected to provide appropriate adaptations. Computing will further enhance the quality of life, but this will not be because of more ‘intelligent’ machines. This will happen because of the fact that computing technologies will be ubiquitous and will extend our physical (e.g., remote controls), sensory (e.g., digital sensors) and mental (e.g., automated analyses, simulations) abilities. Therefore, the focus for the coming years should be on:

- smooth integration of human intelligence and machine capabilities with a clear emphasis on human aspects,
Design and Development of User-centric Pervasive Software

- cost-effective development approaches, methodologies, and tools appropriate for development-time and run-time adaptations.

In this paper, we have provided a meta-review and discussion motivating an approach merging MDD and ontologies to cope with increasing software complexity, and provided theoretical insights pleading that adaptive and pervasive computing systems and applications foster such an approach. The presented review and discussion spans related trends and paradigms in software engineering, artificial intelligence and human-computer interaction, thereby demonstrating the interdisciplinary nature of the work required. We have presented a broad literature to set the given discussion into a concrete context, to put forward available work required to realize the discussed approach, and to identify current weaknesses in the related literature. Our future work includes the application of this approach for automated development of adaptive and pervasive learning environments (APLEs [202]) to provide a personalized, any-time and any-place learning experience.

Acknowledgments. This paper is based on research funded by the Industrial Research Fund (IOF) and conducted within the IOF knowledge platform “Harnessing collective intelligence to make e-learning environments adaptive” (IOF KP/07/006). This research is also partially funded by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, and by the Research Fund KU Leuven.

References


Design and Development of User-centric Pervasive Software


122  Design and Development of User-centric Pervasive Software


2.3 Ubiquitous Web Navigation through Harvesting Embedded Semantic Data: A Mobile Scenario

Authors: Ahmet Soylu, Felix Mödritscher, and Patrick De Causmaecker


I am the first author and only PhD student in the corresponding article. I am the main responsible for its realizations. The co-authors provided mentoring support for the development of the main ideas.

An earlier version was published in:

Ubiquitous Web Navigation through Harvesting Embedded Semantic Data: A Mobile Scenario

Ahmet Soylu¹, Felix Mödritscher², and Patrick De Causmaecker¹

¹ Department of Computer Science, ITEC-IBBT, CODES, KU Leuven, Kortrijk, Belgium
² Institute for Information Systems and New Media, Vienna University of Economics and Business, Vienna, Austria

In this paper, we investigate how the Semantic Web can enhance web navigation and accessibility by following a hybrid approach of document-oriented and data-oriented considerations. Precisely, we propose a methodology for specifying, extracting, and presenting semantic data embedded in (X)HTML documents with RDFa in order to enable and improve ubiquitous web navigation and accessibility for end-users. In our context, embedded data does not only contain data type property annotations, but also object properties for interlinking, and embedded domain knowledge for enhanced content navigation through ontology reasoning. We provide a prototype implementation, called Semantic Web Component (SWC) and evaluate our methodology along a concrete scenario for mobile devices and with respect to precision, performance, network traffic, and usability. Evaluation results suggest that our approach decreases network traffic as well as the amount of information presented to a user without requiring significantly more processing time, and that it allows creating a satisfactory navigation experience.

1 Introduction

In recent years, researchers have paid increasing attention and effort to the Semantic Web. Tim Berners-Lee and his colleagues have formulated the vision of the Web as a universal medium for data, information, and knowledge exchange [6]. The so-called ‘Semantic Web’ aims at increasing the utility and usability of the Web by utilizing semantic information on data and services [24]. Generally, Semantic Web approaches build upon specifications for modeling and expressing web semantics [17], e.g., the Resource Description Framework (RDF), different data interchange formats (RDF/XML, N3, N-Triples, etc.), the Web Ontology Language (OWL), and the forth. However, less attention is paid to embedded semantics (e.g., RDFa, eRDF, microformats, and microdata). Existing approaches, like microformats or various harvesting solutions [2, 28], are normally restricted to pre-defined and widely accepted formats with a specific focus on structure of the data or to specific analysis techniques like text summarization or latent semantic analysis etc. [19].

On the one hand, to a large extent, the work done has focused on challenges regarding machine consumption of semantic data while less attention has been paid to the perspective of human consumption of web semantics, e.g., the enhancement of web navigation. Yet, observable efforts aiming at the provision of user-friendly means
for displaying and browsing available semantic data either address expert level users (i.e., developers) or content publishers with data-centric approaches (e.g., the generation or validation of data mashups). Therefore the key challenge how to utilize Semantic Web technologies to improve end-user functionality remains uncovered.

On the other hand, although there exists a variety of client side applications, like the Firefox add-on named “Operator” which detects and extracts embedded information, the restricted resources (i.e., limited memory, screen size, internet bandwidth, processing power, etc.) of mobile and embedded devices available within Ubiquitous Computing (UbiComp) [34] environments make extraction of semantic data from webpages a non trivial task; in particular if pages include a high amount of multimedia content as well as textual, informational and structural elements. The Web is supposed to be the main information source for UbiComp environments, hence it is important to ensure accessibility for different devices with varying technical characteristics.

![Figure 1: The Semantic Web Component.](image)

Above and beyond, this paper aims at using embedded semantic data to enable and enhance ubiquitous web navigation and accessibility by following a hybrid approach merging document-oriented and data-oriented considerations, addressing devices with different characteristics and considering usability matters like the cognitive load of users. In order to counteract the aforementioned critical challenge, we examine how embedded semantics can be used to access and navigate web-based content according to its internal structure and semantics described by embedded data. Precisely, we propose a methodology for specifying, extracting, and presenting semantic data embedded in (X)HTML through RDFa for the end-users. In the course of this work, embedded semantic data is not only considered to be structured (i.e., with types and data type properties), but also to be linked/related (i.e., interlinked with object type properties) [8]. We further assume that web-based content embeds domain-specific knowledge (limited with subclass and type relations at the moment) which can be used to improve content navigation through basic ontology reasoning. Consequently, we describe the technical realization of the proposed methodology through a server-
sided prototype called Semantic Web Component (SWC), see Figure 1, and evaluate our approach along a concrete scenario for mobile devices and with respect to precision, performance, network traffic, and usability. Evaluation results suggest that our approach decreases network traffic as well as the amount of information presented to a user with a fair processing time, and that it allows creating a satisfactory navigation experience.

The proposed methodology enables users and devices to access and navigate websites along their semantic structure. Thus, (human and non-human) actors can interact with related information only, not being confronted with irrelevant content. The costly extraction process takes place at the server side. It reduces the size of information to be transferred and processed drastically and fosters the visualization of websites on devices with smaller displays, thereby providing increased accessibility and an efficient integration into the UbiComp environments.

The rest of paper is structured as follows. Section 2 criticizes the embedded semantics technologies with respect to the requirements of the proposed methodology while Section 3 reports on related work. Section 4 describes the overall approach for Semantic Web navigation. In Section 5, we present and elaborate on the proposed methodology and describe design and implementation of a first prototype. Then, Section 6 evaluates the computational feasibility and usability aspects of the methodology before Section 7 discusses further work and concludes the paper.

2 Embedded Semantics

Web content can be presented in two distinct facades: (a) human readable facade and (b) machine readable facade of information. Structurally separating these two facades requires information to be duplicated both in the form of HTML and in the form of RDF, XML etc. Embedded technologies use the attribute system of (X)HTML to annotate semantic information so that two facades of information are available in a single representation. Embedded semantics, eRDF, RDFa, microformats, and microdata [1, 5, 18, 20], enables publishing machine readable structured data about non-information sources (i.e., things in the world) from diverse domains such as people, companies, books, products, reviews, genes etc. within (X)HTML documents. In [1], four criteria are listed for embedding semantic information. (1) Independence & extensibility: A publisher should not be forced to use a consensus approach, as she knows her requirements better. (2) Don’t repeat your-self (DRY): (X)HTML should only include a single copy of the data. (3) Locality: When a user selects a portion of the rendered (X)HTML within his browser, she should be able to access the corresponding structured data. (4) Self-containment: It should be relatively easy to produce a (X)HTML fragment that is entirely self-contained with respect to the structured data it expresses.

Among these four criteria, independence & extensibility and self-containment are important for the overall approach. eRDF has to provide vocabulary related information in (X)HTML head while microformats either assume the client to be aware of all available syntaxes beforehand or a profile URI is to be provided for extraction. Microformats and eRDF lack self containment because it is not possible to
re-use eRDF or microformat information without requiring vocabulary specific information. On the other hand microformats lack independence & extensibility since they are based on pre-defined vocabularies and they require a community consensus. Apart from encoding explicit information to aid machine readability, support for data interlinking [8] and implicit knowledge representation (i.e., sharing domain knowledge), thus ontological analysis, or logical inference [27] are of importance, particularly, although not a strict requirement, our approach benefits from these merits for enhancing the navigation experience. Therefore implicit knowledge representation can be considered fifth criterion. Microformats do not address implicit knowledge representation, hence logical inference and ontological analysis [21] while eRDF is not fully conformant with the RDF framework. Data (inter-)linking, regarded as a sixth criterion, enables linking different data items either within the local information source, or in a broader sense, across external information sources. This requires data items included in (X)HTML to have their own identifiers (i.e., HTTP URIs), which is missing in microformat approach preventing relationship assertions between data items [8].

An evaluation of embedded semantics technologies is given in Table 1 with respect to the six criteria.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RDFa</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Microdata</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>eRDF</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Not fully</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Microformats</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>Not fully</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

3 Related Work

Related work can be discussed under two main dimensions: The first one is the navigation which can be further categorized in terms of data vs. document-oriented approaches and generic vs. domain-specific approaches. The second dimension is data extraction in terms of client-based vs. server-based approaches.

Considering the first dimension, traditional hypertext browsers follow a document-centric approach by allowing users to navigate forward and backward in a document space through HTML links. Similar to the document network of the current web, linked data creates a global data graph through RDF links connecting different data items. Result of this data-centric approach is that a user may begin navigation in one data source and progressively traverse the Web by following RDF rather than HTML links [8].

Consequently, several generic linked data browsers [7], following this data-centric approach, have been developed. Zitgist (http://zitgist.com/), Tabulator (http://www.w3.org/2005/ajar/tab), Marbles (http://marbles.sourceforge.net/), Disco (http://www4.wiwiss.fu-berlin.de/bizer/ng4j/disco/), sigma [31] (http://sig.ma), Dipper (http://api.talis.com/stores/iand-dev1/items/dipper.html), jSpace
Ubiquitous Web Navigation through Harvesting Embedded Semantics

(http://www.clarkparsia.com/jspace/), and Exhibit (http://www.simile-widgets.org/exhibit/) are the most notable ones. Although a data centric approach gives users the advantage of discovering new data sources, they are not tailored for the end-users. Users might face a heavy cognitive load, thereby loosing the focus, since navigation might involve other data sources which are not explicitly selected. Normally, a website is organized into a set of information sources (i.e., (X)HTML documents) where each information source stands as a container for non-information sources (i.e., entities). Linking through these documents is established in a user-centric way rather than a data-centric one in order to provide a smooth navigation experience. Therefore a hybrid methodology merging document-centric and data-centric approaches should be more appropriate. Furthermore, generic linked data browsers usually omit supportive metadata required to improve utility of embedded data for human users. For instance, data type properties, object properties etc. are presented to the users with their machine-targeted identifiers. Most of the time these identifiers are not well suited for human consumption (since they are not intended to be), thereby necessitating a standardized metadata layer providing required labeling and generic knowledge.

Apart from generic linked data browsers, several domain-specific viewers for linked data have also been developed. Amongst others, [10] proposes a Semantic Web portal for supporting domain users in organizing, browsing and visualizing relevant semantic data. In [4], a template-based approach is followed for improving syndication and use of linked data sources. The domain-specific nature of these approaches allows them to present data in a form suited to its characteristics (e.g., for instance geographical data can be presented in a map). However the need for the domain knowledge restricts these approaches to content producers and mashups for specific presentation environments. The goal of our work is to come up with a generic data browser that can present any type of custom or de-facto standardized embedded data on different devices.

There are several studies, out of linked data community, addressing web access through mobile devices with limited sources. Amongst others, [3] reports on website personalizers observing the browsing behaviors of website visitors and automatically adapting pages to the users. Moreover [9] examine methods to summarize websites for handheld devices. In [12], authors employ ontologies (OWL-S) and web services to realize context-aware content adaptation for mobile devices. These approaches either require authoring efforts, e.g., for creating ontologies, or are based on costly AI-based techniques.

Considering the second dimension (i.e., extraction), we highlight work particularly characterizing the basic challenges. In [28], the authors describe a web service extracting and collecting embedded information for learning resources from webpages. The harvested information is stored in a semantic database, allowing other clients to query its knowledge base through SPARQL. In the scope of earth observation, [11] apply RDFa for identifying embedded information through a browser extension [14]. The information extracted is either used to populate ontologies or stored in a semantic repository. In [11, 14, 28], it is shown that there exist different ways of harvesting embedded information. On the one hand, client side tools, such as Operator or Semantic Turkey, are used to extract or distinguish the annotated information from web content. The main drawback is that these approaches
require a client side mechanism to extract information hence computing resources of the clients are used. Furthermore, the whole content has to be downloaded to the target machine, which is problematic due to the network load. On the other hand, third-party web applications or services, as demonstrated in [28], can be utilized. In this case, semantic search services usually duplicate the information by means of storing extracted information separately, which violates the DRY principle. It also imposes a dependency on other third-party web applications or services. Clearly such approaches are not feasible for the UbiComp environments with various tiny devices.

Regarding embedded semantic technology to use, [1] proposes a mechanism for unification, namely hGRDDL, to transform microformat embedded (X)HTML into its RDFa equivalent. This mechanism aims at allowing RDFa developers to leverage their existing microformat deployments. Authors advocate that their proposal can allow RDFa to be a unifying syntax for all client-sided tools. There are two important problems in this approach. First of all, developers need to provide vocabulary and syntax for each microformat to be transformed. Although the description language that we proposed earlier [29] can solve this problem, we disagree with unification by means of a unified syntax since a decision between microformats and RDFa is a tradeoff between simplicity (i.e., usability) and functionality.

4 Proposed Solution Approach

The overall approach requires that, at the server-side, requests and the responses between the client and the server be observed. When a client initiates a semantic navigation request for a page of a website, semantically annotated information (i.e., embedded data) is filtered out instead of returning all the (X)HTML content. The extracted information is presented as an (X)HTML document (i.e., reduced content). All non-annotated information is simply discarded. This facade is still the human facade; however, it allows users to navigate through the semantic information available in a website by following data links and relevant HTML links. Each (X)HTML page might contain links referring to other pages of the site having embedded information. Such links are also annotated and called semantic links. Such an approach considers a website as a graph, and the pages of it as a set of nodes where each node represents a sub-graph containing instances of several types. Data items (i.e., instances) are composed of data type properties associating them with data literals, and object properties linking them to each other. The embedded data together with semantic links (i.e., HTML links), associating pages, and object relations (i.e., RDF links), associating data instances, create a semantic information network, as we named it.

The advantage of the current document-oriented web navigation is that each page contains conceptually related information, and enables the user to have an overview of the content, increasing content and context awareness and control [30]. However the problem is that, in each page the user is confronted with ample amounts of information. A purely a data-oriented approach has the advantage of enabling the user to iteratively filter the content in order to accesses information of interest. However applying a purely data-oriented approach to web navigation is problematic since: (1)
in data-oriented approaches the navigation is highly sequential, consequently, long data chains constructed through RDF links can easily cause users to lose provenance and get lost, (2) embedded data available in different pages of a website does not necessarily need to be related or linked. In this context, purely data-oriented approaches are more suitable to expert users for specific purposes, like ontology navigation. We follow a hybrid approach merging document-oriented and data-oriented considerations. The hybrid approach gathers the benefits of both approaches: (1) by following semantic links a user can switch focus from one information cluster/sub-graph (i.e., webpage) to another at once, hence navigation experience is not highly sequential, while content and context awareness and control are maintained, and (3) by following data links within a webpage, the users can access information of interest through iteratively filtering the content rather than being confronted with abundant information.

The approach requires every instance, type, data type property, and object property (i.e., relationship) to be annotated with human consumable metadata for presentation purposes. The minimal requirement is assignment of human readable labels. A thumbnail and a short description are optional, but quite enhancing. Although additional domain knowledge can be utilized if available, the approach is domain independent and does not necessarily rely on the existence of such information. RDFa allows our approach to make use of the full potential of the RDF framework (i.e., types, item relationships, class relationships etc.) while the applicability of our approach with microformats is limited to typed instances (i.e., structured data) with a highly linear navigation experience. Any information source which includes embedded information in RDFa, microdata, microformats, or eRDF is supported by our proposed solution, as far as the basic requirement is satisfied (i.e., appropriate labeling). Several improvements for navigation have also been explored, for example, based on the number of available instances of a class or cardinalities of relationships. These are to prevent long request chains (see Section 5). A server-sided mechanism is preferred in order to isolate end-user devices from computational load of the extraction.

Two example scenarios, based on the semantic information network map depicted in Figure 2, are introduced to demonstrate the applicability and benefits of the proposed approach. A cinema company provides recommendations for the movies of the season through its website. The site includes the pages ‘Events’ and ‘Reviews’. Each movie is considered as an event in the ‘Events’ page. ‘Reviews’ page contains the reviews about the movies. Reviews are provided by registered reviewers, and users are subscribed to receive reviews. Additionally, address information for branches are given at the main page. Events, people, reviews, and addresses are particularly important entities, since they construct the essence of its content. Therefore, corresponding instances have been annotated in RDFa and widely known vocabularies such as iCal, vCard, and hReview are employed.

Scenario-1: A user wants to see a movie tonight. He does not have much time to surf through the website to find a proper movie. Furthermore, he only has his mobile phone around. However his mobile device’s connection and screen properties are at a low level. Since the website is hosted by a server which is SWC enabled, the user simply sends a request through his mobile phone. His browser implicitly tells the server that it only requests annotated information. If the index page is requested, the
server returns the list of semantic links and information available in the index page: ‘Address’ data type, ‘Events Page’, and ‘Reviews Page’. The user follows the ‘Reviews Page’, and the server returns the list of available types: ‘Reviews’, and ‘People’. The user selects the ‘Reviews’ type (see Section 5.1 for labeling and class names) and its instances are retrieved. The instances are presented by the titles of the movies and possibly associated with a small thumbnail and a short description if available. The user selects a review about a movie of interest and reads the review. He wants to see who wrote it to be sure that the quality of this information can be relied upon. The user follows the data link to the corresponding reviewer to access the reviewer instance for details. Then he navigates to the ‘Events’ page to see the schedule related information. This basic scenario is meant to address any kind of client (e.g., mobile, stationary etc.) having access to the Internet and any HTML browser with basic capabilities.

![Semantic information network map referring to semantic structure of a website.](image)

**Figure 2:** Semantic information network map referring to semantic structure of a website.

Scenario-2: Another user is visually handicapped, and uses a screen reader to navigate the Web. Information on websites is often abundant. Therefore she has to spend a lot of time to access the information of interest. She accesses a SWC enabled website. Only the semantic information constituting the essence of the site is downloaded. Furthermore, this data is not presented as a whole but presented in a hierarchical manner. On the one hand, the amount of content is reduced hence the total amount of text to be read. On the other hand, since the data is presented in a simple hierarchy, she can access a particular piece of information leaving non-essential and non-interested items unread.

In the following the advantages of the proposed approach are summarized with respect to related work presented in Section 3. (1) **Direct and seamless access** to different facades of the information without imposing any burden to the client side, e.g., no need for data extraction. (2) **Enhanced user experience:** users are usually lost
in the abundant information space [25] of the Web where valuable information is hidden in the ‘information sea’ and as part of presentational and structural elements. Users can simply access the desired information. (3) Increased accessibility and ubiquity: mobile and embedded devices in the UbiComp environments can use both facades of the information. (X)HTML representation of the reduced information will enable them to deliver web information to anyplace while the machine-readable form of the information will enable devices to process and use the web information. (4) Higher network efficiency: the devices do not need to retrieve all the (X)HTML content from the server, hence the amount of information travelling in the network decreases. (5) Semi-centralized and a generic solution: the fact that different embedded semantics technologies are available requires unifying the use of these technologies. In this paper, this diversity is advocated and unification is considered to happen through the server-sided extraction mechanisms rather than by opting for a single technology. (6) Low entry barriers: Web publishers do not need high investment in development, content-authoring, and hardware, and since the requirements for the end-user devices are basic, the users do not need high-cost devices.

5 Methodology, Design, and Implementation

A sample HTML page with embedded linked data is used through this section in order to exemplify the proposed methodology.

Figure 3: Partially extracted ontology from the sample website; data type properties are omitted for the sake of brevity.

The example HTML page has been taken from the linked data tutorial published online at [16]. Exact details of the embedded data as well as the original sample document can be found in [16]. The original HTML document is divided into sub-pages, namely Home (1), Products (2) and People (3), in order to demonstrate
The sample site provides basic information about a company, its products, and team members. Particular name spaces are used: ‘gr’ (GoodRelations) and ‘tools’ (a self-defined domain vocabulary) to present company and product related information, ‘vcard’ to present business card of the company, and ‘foaf’ to present information about people. The partial ontology of the website is depicted in Figure 3. It is partial since the domain knowledge available is limited to the information revealed by the extracted instances.

5.1 Document preparation

The embedded semantic data needs to be adapted in three levels: (1) metadata level describes how instances, classes etc. needs to be annotated with human understandable textual and visual elements; (2) domain knowledge level describes how additional domain knowledge can enhance the navigation; and (3) navigation level describes how navigational hierarchy can be constructed and fine tuned.

Metadata level: Embedded information within the (X)HTML document needs to be accompanied with appropriate metadata in order to facilitate user consumption. Table 2 shows the list of required and optional metadata elements.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name space: RDF Schema, Element: rdfs:label</td>
<td>Provides human readable labels for: types (i.e., classes), instances, data type properties, and object properties (i.e., relations).</td>
</tr>
<tr>
<td>Optional: No</td>
<td></td>
</tr>
<tr>
<td>Name space: RDF Schema, Element: rdfs:comment</td>
<td>Provides human readable short descriptions for: types, instances, data type properties, and object properties.</td>
</tr>
<tr>
<td>Optional: Yes</td>
<td></td>
</tr>
<tr>
<td>Name space: Dublin Core, Element: dc:description</td>
<td>Provides thumbnail images for: types, instances, and object properties.</td>
</tr>
<tr>
<td>Optional: Yes</td>
<td></td>
</tr>
</tbody>
</table>

Embedded data is often presented by using original type, property, and relationship identifiers as it appears in a vocabulary or ontology. In other cases, it is assumed that the presentation environment has knowledge about the domain and provides appropriate visual elements for the humans. On the one hand, original names are not meant for regular end-users and mostly do not convey any meaning for them. On the other hand, assuming availability of domain pre-knowledge is not realistic within the context of generic semantic web browsers. Therefore embedded data needs to be accompanied with basic presentational metadata, primarily textual labels, while short descriptions and thumbnails are essentially enhancing.

Naming convention in a vocabulary or ontology is singular, since a class represents a real world concept. However, during navigation, the user perceives a class as a set of instances of a particular concept. Hence, the following conventions are suggested. We use plural label values for classes, (e.g., People for foaf:Person). While labeling data type and object type properties, rather than using the classical naming used in
vocabularies or ontologies, i.e., ‘verb’ – ‘range class name’ such as ‘gr:hasPriceSpecification’, target class name, or if not appropriate, another noun should be used without attaching a verb (e.g., ‘Price’ instead of ‘hasPrice’). Plural label values should be used for Object type relations, if the cardinality is higher than 1, otherwise singular label values are appropriate (e.g., ‘Price’ for gr:hasPriceSpecification, and ‘Members’ for ‘foaf:member’).

**Domain knowledge level:** Indeed, every application, maintains a conceptual model of its domain. Such conceptual models might be implicit (i.e., encoded in application code), or explicit but informal (i.e., not machine readable, e.g., documentations). Once a conceptual model is made explicit and formal, it can be used for knowledge share and reasoning. Embedded data within an (X)HTML document does not only serve domain instances, but also formalizes a part of domain knowledge through embedded data instances. Although the revealed knowledge is sufficient for our proposal, additional domain knowledge can be explicitly provided through embedding it along with data instances. In our context, subclass relationships (among others such as domain, range constructs etc.) are particularly beneficial. This is because domain knowledge can enhance the navigation experience, for instance, by classifying data instances with respect to a class hierarchy. Otherwise content navigation might be almost linear (object relations prevent it from being fully linear), and might overload the users with long lists of instances.

Considering the partial ontology presented in Figure 3, a publisher could specify that ‘tools:Hammer’, ‘tools:Screwer’, and ‘tools:Fubar’ are subclasses of ‘tools:AllPurposeUtility’, and that ‘tools:AllPurposeUtility’ is a subclass of ‘gr:AllProductServiceInstance’. Then, a major question arises concerning the amount of knowledge to be provided, for instance, after saying that ‘instance A’ is type of ‘tools:Hammer’, is it necessary to explicitly state that ‘tools:Hammer’ and ‘tools:Screwer’ are subclasses of ‘tools:AllPurposeUtility’, and ‘instance A’ is type of ‘tools:AllPurposeUtility’ and ‘gr:AllProductServiceInstance’? In the context of the proposed methodology, a publisher can choose to provide subclass relationships, and leave ‘subclass’ and ‘type’ relationships to be inferred through ontology reasoning. That being possible, it compromise the performance since inference takes additional time. Nevertheless, in a typical webpage, the amount of domain knowledge and instances are expected to be limited, hence the inference process should not cause long delays. Indeed, another possibility is to extract useful domain knowledge without requiring the publisher to provide it explicitly. This is possible by employing an ontology learning mechanism using clues revealed by the existing data instances. However, this might be costly and not sharply precise and accurate. The possible benefits and drawbacks will be discussed in Section 5.2. We presently do not utilize an ontology learning mechanism.

**Navigation level:** Our methodology intertwines document and data hierarchies. The former outlines the navigation while the latter determines access paths to content items. Data hierarchy is constructed through object relations and class-subclass relationships, while document hierarchy is constructed through HTML links. The main strategy, when a page is accessed, is to first list all available classes, which do not have a parent class, together with semantic links; this step constructs the front layer of the navigation hierarchy. Then the user is enabled to navigate through by selecting a class, subclass, instance, relation, range class and so on. For instance,
consider that there are only two classes, ‘class A’ and ‘class B’, where ‘class B’ is subclass of ‘class A’ or an instance of ‘class A’ has an object relation with an instance of ‘class B’. In the latter case, with the first request of the page, both classes will be presented to the user. On navigating through the ‘class A’, the user will also encounter the ‘class B’. In the former case, only ‘class A’ will be presented, and ‘class B’ will only be accessible through ‘class A’.

Neither a fully linear nor a fully hierarchical representation of the content is appropriate for the web navigation. Here, it should be possible to break the hierarchy and move a particular branch to the front, or to remove/hide a particular class from the front. Precisely: (1) a class might be hidden/removed from the front layer, if the existence of this class strongly depends on another class (e.g., similar to weak entities in relational databases), (2) a class might be moved to front, if it contains instances which need to be immediately available through the front layer. An example is depicted in Figure 4.

**Figure 4**: Navigation hierarchy constructed for the Products page through object relations, class-subclass relations, ‘swc:hide’, and ‘swc:front’.

It shows classes available in the navigation hierarchy, at different levels, for the Product page of the sample site. This hierarchy is based on the following labeling: ‘Company’ for ‘gr:BusinessEntity’, ‘Offers’ for ‘gr:Offering’, ‘Prices’ for ‘gr:UnitPriceSpecification’, ‘Amounts’ for ‘gr:TypeAndQuantityNode’, ‘All Purpose Utilities’ for ‘tools:AllPurposeUtility’, ‘Products’ for ‘tools:AllPurposeUtility’, ‘Fubars’ for ‘tools:Fubar’, ‘Hammer’ for ‘tools:Hammer’, and ‘Screwers’ for ‘tools:Screwer’. Each area between bold horizontal bars represents a navigation layer (the top one being the first layer). A class might appear in different layers due to object properties. In this example, ‘Company’, ‘Offers’, ‘Prices’, ‘Products’, and ‘Amounts’ appear in the front layer since they do not have any parent class. However, it is quite logical to remove ‘Prices’ and ‘Amounts’ from the front layer of the navigation hierarchy. This is because, in the sample, ‘Amounts’ are associated with ‘Offers’ through an object relation (see Figure 3), hence every instance of ‘Amounts’ is associated with an instance of ‘Offers’. In the context of the sample website, every ‘Amounts’ instance is only meaningful with an ‘Offers’ instance which makes direct access to an ‘Amounts’ instance useless and similarly for ‘Prices’. Since the ‘Company’ class is the primary class of the data hierarchy, it might be appropriate to remove/hide it from navigation hierarchy since it might lead to a fully hierarchical experience. Consider that a new class ‘New Products’, being the direct subclass of
‘Products’, is added in order to advertise newly added products. The publisher may want to inform the customers as quickly as possible about new products by pushing this class to the front, where it can be directly accessed from the front layer as well as from its original position in the navigation hierarchy.

Two new data properties support this purpose, ‘swc:hide’ and ‘swc:front’, in our application’s ‘swc’ name space. ‘swc:hide’ and ‘swc:front’ can only be applied to classes, and their value should be assigned to ‘yes’. Another matter, concerning the navigation hierarchy, is interweaving HTML navigation and data navigation as demonstrated in Figure 2. To support annotation of HTML links targeting sub-pages of the website containing embedded semantic data, a new class has been introduced, ‘swc:SemiaLink’. Every link targeting another page containing embedded semantic data, should be annotated as an instance of ‘swc:SemiaLink’.

5.2 Extraction, Reasoning, and Presentation

Extraction and reasoning processes are conducted at the server side while the presentation related process can either take place at the client (through JavaScript calls to the server) or at the server side (through HTML links). In any case, aforementioned processes exist under two layers, namely the extraction & reasoning layer and the presentation layer. Both of the layers are instance-based, that is a class or an object property etc. are only accepted to exist, if there is at least one instance associated with it.

*Extraction & Reasoning:* At this layer, three core services are provided: (1) ‘getClasses’ is responsible of retrieval of classes in a particular level of navigation hierarchy, more specifically, (a) all non-hidden classes, having no parent class or those are pushed to the front layer, available at a given HTML document, (b) classes that are direct subclasses of a particular class, (c) all non-hidden classes having a particular relationship with a particular instance, and (d) all classes that are direct subclasses of a particular class and having a particular relationship with a particular instance. This service requires ontology reasoning to be applied which we restrict to only subclass and type inference (i.e., excluding inference for range, domain, inverse of, sub-property etc.). Subclass and type inference are required to provide a hierarchical access to instances. For example, if one asserts that ‘instance X’ is type of ‘class C’, ‘class C’ is subclass of ‘class B’, and ‘class B’ is subclass of ‘class A’, then followings are inferred: ‘class C’ is subclass of ‘class A’, and ‘instance X’ is type of ‘class B’ and ‘class A’. This implies ‘class B’ and ‘class A’ have to be visited before ‘class C’. (2) ‘getInstances’ is responsible of retrieval of instances in a particular level of the navigation hierarchy, more specifically, (a) all the direct instances of a particular class, (b) all the direct instances of a particular class, and are in a particular relationship with a particular instance. (3) ‘getInstance’ is responsible of retrieval of a particular instance in a particular level of the navigation hierarchy, more specifically, all the properties of a particular instance. For consistency and performance matters, once embedded data is extracted, it is temporally stored during session life-time. Since only ‘getClasses’ service requires inferred data and others operate on non-inferred data pool, extracted information is stored both as is and with inferred data.
**Presentation layer:** The navigation starts with loading all classes having no parent class, and without imposing any specific relation with any instance. Once the results are received from the extraction & reasoning layer, after invoking the ‘getClasses’ service, the class related presentational metadata is listed as a menu item for each class. Specific actions are hooked to each item invoking ‘getClasses’ (retrieving direct subclasses of selected parent class) and ‘getInstances’ (retrieving direct instances of the selected class); this is repeated for any subclass selected. While listing instances, if the instance being listed is a ‘SemanticLink’ type instance, it is hooked with a specific action invoking the ‘getClasses’ service, and the instance URI becomes the new URL of the navigation provenance. If the instance being listed is not a semantic link type, the instance is hooked with an action invoking the ‘getInstance’ service in order to retrieve and present all the properties of the selected instance. If a property being presented is not a data type property but an object property, then an action invoking the ‘getClasses’, ‘getInstances’, or ‘getInstance’ service is hooked to it. Deciding which action to be hooked to an object property is subject to some heuristics for enhancing the user experience.

These heuristics are constructed by following an approach similar to a possible ontology learning mechanism based on embedded data. However it is fundamentally different in the sense that findings are mostly pseudo and specific to a single session. For instance, as mentioned in Section 5.1, rather than explicitly providing a class hierarchy, it can be constructed through analyzing the existing data. Nevertheless, an (X)HTML document with embedded data represents only a single portion of possible instances, and most of the domain knowledge deduced from this single set cannot be generalized due to the Open World Assumption (OWA) which is quite natural to follow in our context. Since the presentation approach followed by the proposal is situated on existing data rather than a reference domain ontology, some pseudo deductions might lead to useful heuristics which are demonstrated in Figure 5. It is possible to shorten navigational chains by making use of object property characteristics such as cardinality and range. Normal behavior, after a particular relation is being selected, is to present instances in a class hierarchy (by default assuming that the cardinality is more than 1 with a multiple range). Consider the child-path P2 of P. To access price information of a particular offer, a user has to pass
5 navigation levels. However, since there is only one instance associated with this relation (pseudo cardinality 1), rather than listing the class hierarchy and then the instances (which is only one), the navigation can directly jump to the instance description. Such an approach shortens the navigation path 2 levels, and in cases where the real cardinality is exactly one, it removes a possible confusion by not following a singular relation label with a plural class label. Considering the child-path P1 of P, to access amount information of a particular offer, a user has to pass 5 navigation levels. However, since there is only one class having at least one instance associated with this relation (pseudo range 1), rather than listing the class hierarchy (which is only one), the navigation might directly jump to the list of instances. Such an approach shortens the navigational path 1 level. As demonstrated, pseudo cardinalities and ranges can lead to useful practices and omitting unnecessary navigation levels becomes possible due to the fact that textual description of a relation itself already reveals sufficient information about the coming presentational level. Moving back to the ontology learning discussion, we do not attempt to learn a class hierarchy, because, differences between deduced pseudo class hierarchies and original ones may not be accepted by the users or confuse them.

5.3 Architecture

The proposed architecture (see Figure 6) for SWC consists of three modules for a HTTP server.

![Figure 6: SWC architecture consists of three modules for HTTP servers: Mod Semantic, Mod GRDDL, and Mod SWC.](image)

(1) *Mod Semantic* is responsible for extracting contextual information from the request header. It detects the device type or extracts an explicit semantic navigation
request from the request header encoded with a specified parameter. This parameter can be further adjusted for directly accessing machine readable information (e.g., RDF). The module reads the requested (X)HTML document from the application pool and forwards it to ‘Mod GRDDL’, if semantic navigation request is active, otherwise to the client.

(2) *Mod GRDDL*: This module is responsible for extracting embedded semantic data from the (X)HTML. It stores the data, once extracted, to the session store temporarily, in RDF form, during the client’s session life time. If inference over extracted data is demanded, it applies ontological reasoning and stores a new data set separately. File identifiers are created through hashing source URL, session id of the client, and ‘inferred’/‘noninferred’ parameters for quick and error free access.

(3) *Mod SWC*: This module is responsible for preparing and maintaining the state of the presentation. It detects the state of navigation (i.e., the active navigation level) and extracts the requested navigation level. It queries the session store, with SPARQL, for the corresponding URL and the parameters. On directly requesting machine readable information, it returns RDF data to Mod Semantic, otherwise it generates the requested presentation level in (X)HTML.

A listener is associated with Mod Semantic in order to direct incoming client requests to the SWC. Once Mod SWC delivers back the final output, Mod Semantic forwards it to the client. There might be other modules processing the content, for instance a script interpreter for dynamic content. In this case Mod Semantic needs to be placed in appropriate order within the module queue.

6 Evaluation

SWC has been implemented to prove its applicability. A first and simpler variant was restricted to microformat [29]. The current version supports RDFa, eRDF and microformats (as long as the requirements are satisfied) and is based on PHP. It behaves like a proxy. This design decision has been made due to the simplicity of the implementation and in order to provide a demonstrator for the community. The demonstrator is available through the following link http://www.ahmetsoylu.com/pubshare/icae2011/.

Two versions are available currently: The first version is based on PHP and JavaScript for providing enhanced user experience through a dynamic presentation layer. The extraction & reasoning layer is implemented as a server-sided component and the presentation layer is developed as a client-sided component. This prototype caches the visited navigation layers at the client. Hence, it does not require executing server calls more than once. However this first prototype is only suitable for devices capable of executing JavaScript and having sufficient amount of browser cache. Although today most of the mobile devices have these capabilities, a second version has been developed as a fully server-sided component. An example navigation session of a user seeking a particular offer is depicted in Figure 7.
6.1 Performance

Two performance tests for SWC have been conducted in order to validate the computational feasibility of the proposed approach. The tests measure the performance for data extraction, inference, and presentation processes. The first test is applied to sample (X)HTML documents containing increased amount of embedded linked data instances. The time spent for extraction, for inference, and the total amount of time spent for delivery of the first request and for the second request have been traced for each (X)HTML document. The results are shown in Table 3.

The number of triples extracted increase from 89 (roughly 10 instances) to 1324 (roughly 150 instances). Normally, total number of instances available in a single typical (X)HTML page is not expected to exceed 300 triples. Measurements done for higher numbers have been conducted to demonstrate the feasibility and the limits for larger data-sets.

Results show that the most expensive operation is the extraction taking roughly 5.27 seconds for 1324 triples. The total amount of time spent for delivering the requested content (including extraction, inference, and presentational process) is about 5.61 seconds where a minor amount of time is spent for presentational processing (which includes SPARQL querying), and a comparatively higher amount of time is spent for inference even though no triples are inferred. These results confirm that the approach is computationally feasible for even larger data sets because the extraction process is only executed for the first request of a page. Thereafter, the
semantics embedded within the page is extracted and stored for subsequent requests during the session. The total amount of time spent for 2nd request (1324 triples) lasts only 0.173 seconds, which is a reasonable value compared to the 5 seconds required for the initial request. Although the time spent for inference is minor, defining an optional parameter for the document header indicating whether inference is required can eliminate it.

Table 3: Performance results for SWC with no inferred triples (all time measurements in seconds).

<table>
<thead>
<tr>
<th># of extract.</th>
<th># of infer.</th>
<th># of triples</th>
<th>t for extraction</th>
<th>t for inference</th>
<th>total t for 1st request</th>
<th>t for loading 2nd request</th>
<th>total t for 2nd request</th>
</tr>
</thead>
<tbody>
<tr>
<td>89</td>
<td>0</td>
<td>89</td>
<td>0.28298115</td>
<td>0.02677989</td>
<td>0.31824708</td>
<td>0.01701521</td>
<td>0.0261170</td>
</tr>
<tr>
<td>154</td>
<td>0</td>
<td>154</td>
<td>0.47963190</td>
<td>0.04098892</td>
<td>0.52989387</td>
<td>0.02307391</td>
<td>0.0327179</td>
</tr>
<tr>
<td>284</td>
<td>0</td>
<td>284</td>
<td>0.85058518</td>
<td>0.06839990</td>
<td>0.94419312</td>
<td>0.03603792</td>
<td>0.0468919</td>
</tr>
<tr>
<td>414</td>
<td>0</td>
<td>414</td>
<td>1.25827693</td>
<td>0.09524011</td>
<td>1.34683893</td>
<td>0.04613614</td>
<td>0.0596082</td>
</tr>
<tr>
<td>544</td>
<td>0</td>
<td>544</td>
<td>1.66216015</td>
<td>0.13231914</td>
<td>1.80117702</td>
<td>0.05918693</td>
<td>0.0740292</td>
</tr>
<tr>
<td>674</td>
<td>0</td>
<td>674</td>
<td>2.09674501</td>
<td>0.15418791</td>
<td>2.23273795</td>
<td>0.07439548</td>
<td>0.0928921</td>
</tr>
<tr>
<td>804</td>
<td>0</td>
<td>804</td>
<td>2.82035994</td>
<td>0.18054699</td>
<td>3.02241993</td>
<td>0.08645002</td>
<td>0.1072847</td>
</tr>
<tr>
<td>934</td>
<td>0</td>
<td>934</td>
<td>3.49862872</td>
<td>0.21200914</td>
<td>3.77563793</td>
<td>0.09976707</td>
<td>0.1256640</td>
</tr>
<tr>
<td>1064</td>
<td>0</td>
<td>1064</td>
<td>4.06736608</td>
<td>0.24071192</td>
<td>4.31638697</td>
<td>0.11668003</td>
<td>0.1408009</td>
</tr>
<tr>
<td>1194</td>
<td>0</td>
<td>1194</td>
<td>4.68414592</td>
<td>0.27325987</td>
<td>4.99209594</td>
<td>0.14202404</td>
<td>0.1759369</td>
</tr>
<tr>
<td>1324</td>
<td>0</td>
<td>1324</td>
<td>5.27055978</td>
<td>0.30244302</td>
<td>5.51915508</td>
<td>0.13527417</td>
<td>0.1731801</td>
</tr>
</tbody>
</table>

The second test is applied to the sample HTML documents containing increased amount of embedded linked data instances and additional domain knowledge causing an increasing number of new triples to be inferred. Regarding inference, only subclass inferences for T-box (stores terminological knowledge, e.g., classes, properties etc.) and type inheritance for A-box (stores assertional knowledge, e.g., instances) are enabled. The results are shown in Table 4.

Table 4: Performance results for SWC with inferred triples.

<table>
<thead>
<tr>
<th># of extract.</th>
<th># of infer.</th>
<th># of triples</th>
<th>t for extraction (second)</th>
<th>t for inference</th>
<th>total t for 1st request</th>
<th>t for loading 2nd request</th>
<th>total t for 2nd request</th>
</tr>
</thead>
<tbody>
<tr>
<td>87</td>
<td>5</td>
<td>92</td>
<td>0.26778292</td>
<td>0.02856206</td>
<td>0.30531597</td>
<td>0.01627492</td>
<td>0.0248689</td>
</tr>
<tr>
<td>147</td>
<td>10</td>
<td>157</td>
<td>0.44807100</td>
<td>0.04376387</td>
<td>0.50182414</td>
<td>0.02059292</td>
<td>0.0303999</td>
</tr>
<tr>
<td>267</td>
<td>20</td>
<td>287</td>
<td>0.81358194</td>
<td>0.07269811</td>
<td>0.89989014</td>
<td>0.03979706</td>
<td>0.0522391</td>
</tr>
<tr>
<td>387</td>
<td>30</td>
<td>417</td>
<td>1.19771814</td>
<td>0.10265803</td>
<td>1.31669497</td>
<td>0.04740500</td>
<td>0.0653518</td>
</tr>
<tr>
<td>507</td>
<td>40</td>
<td>547</td>
<td>1.56997489</td>
<td>0.13547611</td>
<td>1.72626900</td>
<td>0.06719708</td>
<td>0.0836968</td>
</tr>
<tr>
<td>627</td>
<td>50</td>
<td>677</td>
<td>1.92737293</td>
<td>0.16101992</td>
<td>2.11321806</td>
<td>0.07345485</td>
<td>0.0983479</td>
</tr>
<tr>
<td>747</td>
<td>60</td>
<td>807</td>
<td>2.26387000</td>
<td>0.19190812</td>
<td>2.46641109</td>
<td>0.08595013</td>
<td>0.1173040</td>
</tr>
<tr>
<td>867</td>
<td>70</td>
<td>937</td>
<td>3.08285093</td>
<td>0.22520211</td>
<td>3.32474933</td>
<td>0.10141211</td>
<td>0.1382060</td>
</tr>
<tr>
<td>987</td>
<td>80</td>
<td>1067</td>
<td>3.76992201</td>
<td>0.27355003</td>
<td>4.09606311</td>
<td>0.11638498</td>
<td>0.1621799</td>
</tr>
<tr>
<td>1107</td>
<td>90</td>
<td>1197</td>
<td>4.21638107</td>
<td>0.29071710</td>
<td>4.50103014</td>
<td>0.13082194</td>
<td>0.1839830</td>
</tr>
<tr>
<td>1227</td>
<td>100</td>
<td>1327</td>
<td>4.74430990</td>
<td>0.33829903</td>
<td>5.14622592</td>
<td>0.14249491</td>
<td>0.2055420</td>
</tr>
<tr>
<td>1231</td>
<td>200</td>
<td>1431</td>
<td>4.81639003</td>
<td>0.37364699</td>
<td>5.27849888</td>
<td>0.15929596</td>
<td>0.2417650</td>
</tr>
<tr>
<td>1239</td>
<td>400</td>
<td>1639</td>
<td>4.82993100</td>
<td>0.45354890</td>
<td>5.42210185</td>
<td>0.17822194</td>
<td>0.3246259</td>
</tr>
<tr>
<td>1247</td>
<td>600</td>
<td>1847</td>
<td>4.82093501</td>
<td>0.56154394</td>
<td>5.57514977</td>
<td>0.20634293</td>
<td>0.4151589</td>
</tr>
</tbody>
</table>

The time spent for inferring 600 triples from 1247 triples is around 0.56 seconds. The inference process, indeed, is known to be expensive. However, since T-box and
A-box sizes are comparatively small, the time spent for inference is decent. T-box and A-box sizes, hence the number of inferences to be done, are not expected to be high for a typical (X)HTML page. Even for comparatively larger sizes, the results suggest that the proposal is feasible. If necessary, the number of extracted triples can be reduced by only using ‘rdfs:label’ and omitting ‘rdfs:comment’ and ‘dc:description’ elements anyway; this would reduce efforts by two triples per instance, class, and object property.

Overall, since the embedded data is divided into pages, and each HTML document contains a decent number of instances and domain knowledge, the proposed approach is found to be computationally feasible. The proposal is not tested with more data-sets and considerably bigger T-boxes, as the proposed approach does not aim at realizing a browser for knowledge bases with full-fledged inference support. The semantic reasoner used in the implementation and evaluation is not optimized for performance. Mature reasoners, optimized for best performance, are expected to provide even better performance.

6.2 Network Efficiency

To evaluate network efficiency, precision and number of requests are used as criteria. In our context, precision (P) [23] is the fraction of the size of retrieved data that are relevant to the user's information need (i.e., target instance), and number of requests refers to the total number of HTTP calls required to access the target instance. Precision is calculated by

\[ P = \frac{t}{\sum_{i=1}^{n} p_i} \]

where \( n \) denotes number of network requests, \( p_i \) denotes the size of the returned data for a request, and \( t \) denotes size of the target instance. More specifically, for normal navigation, \( p_i \) refers to the size of an (X)HTML page required to be visited to access target information, where for SWC, it refers to the size of a presentational layer required to be visited to access target information. An example trace has been shown in Table 5 for four different target instances.

Although the sample site is quite small and does not include many irrelevant content elements the precision reaches to 72% where it is only 16% for normal web navigation. The lowest precision is measured with SWC is 6% where it is 0.6% for normal web navigation. The lowest precision, although it is still 10 times higher than the normal navigation, results from the size of the target instance. The example instance, a product, only contains its name (one word, i.e., a few Bytes). A typical webpage is expected to have a bigger size, and a higher navigational depth. A typical embedded data instance is expected to have 6-7 properties resulting in 1-2 KB of target data size. In this respect, for a full-fledged website, precision is expected to be much lower for normal navigation and much higher for SWC. Regarding network calls, naturally, SWC requires more network calls than normal navigation, since it
iteratively proceeds to the target instance. However the increase in amount of network calls seems admissible since the amount of information downloaded in each call is considerably small.

Table 5: Precision and number of required requests are traced for a set of target instances.

<table>
<thead>
<tr>
<th>target page</th>
<th>target instance</th>
<th>target instan. size (KB)</th>
<th>retr. data size with SWC</th>
<th>prec. with SWC</th>
<th># of req</th>
<th>retrieved data size with normal navig.</th>
<th>precision with normal navig.</th>
<th># of requests with normal navig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>home</td>
<td>organization</td>
<td>1.81</td>
<td>2.49</td>
<td>0.72</td>
<td>3</td>
<td>11.13</td>
<td>0.160</td>
<td>1</td>
</tr>
<tr>
<td>home</td>
<td>address</td>
<td>0.79</td>
<td>1.48</td>
<td>0.53</td>
<td>3</td>
<td>11.13</td>
<td>0.070</td>
<td>1</td>
</tr>
<tr>
<td>people</td>
<td>a member</td>
<td>1.26</td>
<td>3.05</td>
<td>0.41</td>
<td>5</td>
<td>18.68</td>
<td>0.060</td>
<td>2</td>
</tr>
<tr>
<td>products</td>
<td>a product</td>
<td>0.16</td>
<td>2.54</td>
<td>0.06</td>
<td>7</td>
<td>24.60</td>
<td>0.006</td>
<td>2</td>
</tr>
</tbody>
</table>

The example provided in Table 5, being extremely optimistic for normal navigation, implies higher network efficiency. The significant reduction in transferred data size clearly favors our solution approach.

6.3 Usability

A preliminary usability evaluation has been conducted to see (1) whether our semantic approach can create a satisfactory navigation experience comparable/superior to normal navigation, (2) to find directions for developing more heuristics, and (3) to detect any major usability problems. The usability analysis is targeted to find problems inherent to the methodology itself, regardless of problems originating from the target websites (i.e., content organization).

Table 6: Profiles of the test users.

<table>
<thead>
<tr>
<th>User</th>
<th>#years using Internet</th>
<th>frequency of Internet use</th>
<th># years using mobile Internet</th>
<th>level of expertise</th>
<th>frequency of mobile Internet use</th>
<th>age group</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>daily</td>
<td>0</td>
<td>Regular user</td>
<td>never</td>
<td>25-30</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>daily</td>
<td>1</td>
<td>Regular user</td>
<td>often</td>
<td>20-25</td>
</tr>
<tr>
<td>3</td>
<td>14</td>
<td>daily</td>
<td>3</td>
<td>Developer</td>
<td>occasionally</td>
<td>25-30</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>daily</td>
<td>2</td>
<td>Computer Sci.</td>
<td>sometimes</td>
<td>30-35</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>daily</td>
<td>0</td>
<td>Regular user</td>
<td>never</td>
<td>25-30</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>daily</td>
<td>1</td>
<td>Regular user</td>
<td>often</td>
<td>30-35</td>
</tr>
</tbody>
</table>

As suggested in [33], 5-6 users are normally sufficient to cover major usability problems while 15-16 users provide the highest benefit. This part of the evaluation is iterative and expected to provide a basis for the future work. Hence we opted to conduct a preliminary study first with 6 test users. The profiles of the test-users are given in Table 6.

A think-aloud test was conducted. The test-users were asked to conduct four tasks through the sample website with SWC: (1) find a particular person, (2) find
company’s contact details, (3) find a particular product, and (4) find a particular price information.

Regarding the navigation experience, three users managed to complete given tasks without any observable significant confusion in their first try. ‘User 3’, ‘User 6’ and ‘User 5’ could not complete the first task in their first try (due to content organization) but completed all other tasks in their first try. Users stated that they were quite satisfied with their navigation experience and they did not have any confusion or uncertainty, but had some critiques on the organization of the content. At the end of the tasks, the original pages have been shown to the users, and asked what differences they see. Users, mainly, delivered their experience with normal web navigation, stressed difficulty of finding information, and commented that it is easier to access information with our approach. They listed several websites, which they may want to access with such a mechanism.

Regarding possible suggestions for new heuristics, it has been observed that the users were quite reactant to unexpected situations, for instance, when they did see a class (e.g., menu item) containing only one instance or subclass, they immediately commented on unnecessary navigation levels. Even though the number of instances or subclasses might change (e.g., a new product or product type appears), users were apparently instance-oriented. This might require us to apply a new heuristic, by omitting classes in navigation hierarchy with only one subclass or one instance. For task 3, ‘User 2’, ‘User 4’ and ‘User 5’ tried to access price information through products and complained when they could not access it. This might require us to employ inverse properties and enable inference support for this construct.

Regarding usability problems, users reported two minor issues. ‘User 4’, ‘User 5’ and ‘User 6’ complained that pages are grouped under a category, and commented that they should be directly accessible. ‘User 2’ commented that pages should be accessible at every level of navigation. This might require us enabling semantic link instances to be directly visible without being classified under any parent class. However class based access should be maintained and particularly for websites with high number of links. Whenever users are asked to find an instance at another page, it is observed that users quickly moved back to pages option, and navigated to the correct page. Users were asked whether they know where they are, and successfully replied with the active page. ‘User 2’ commented that he knows being at ‘Products page’, but it might be hard to remember if navigation becomes deeper requiring active page information to be maintained. Users also commented that it would be good if they could search within the content. Providing a search mechanism, exploiting the structure and semantics, is definitely crucial.

Regarding the publisher side of the usability, SWC does not remove the usability problems inherent to content organization, and usefulness of the embedded data (i.e., what to structure and annotate). Publishers need to employ some measures to test usability of the content organization. Two notable measures, that we recommend, are: observed precision and perceived recall. A precision function is defined in Section 6.2 for network efficiency. Indeed it can also be considered as a measure for user cognitive load; since the rate of the presented information has an impact on the cognitive load [26]. However, the precision function defined in Section 6.2 is the expected precision assuming that the users follow the correct paths and do not get lost. Observed precision accounts the unexpected navigational levels that a user
visited. Expected and observed precision can be combined to an efficiency measure to identify usability problems regarding the content organization. The efficiency measure is defined by

\[
E = \frac{t}{\sum_{i=1}^{n} p_i/t} \sum_{j=1}^{m} p_j
\]

where \( n \) denotes number of expected requests, \( m \) denotes number of observed requests, \( p_i \) denotes the size of returned data for an expected request, \( p_j \) denotes the size of the returned data for an observed request, and \( t \) denotes the size of the target instance.

Regarding perceived recall (R) [23], it is the fraction of relevant data size that is retrieved. Recall can be calculated by asking users which information they deem relevant, and comparing it with scope of embedded instances. Perceived recall can allow publishers to fine-tune their decision on what is relevant and essential for the users visiting their website.

7 Conclusion

In this paper, a methodology has been proposed for enabling end-users and devices in UbiComp environments to navigate websites along their semantic structure and domain knowledge. Proposed methodology follows a hybrid approach of document-oriented and data-oriented considerations. Embedded data specification, extraction, and presentation mechanisms have been defined. Several heuristics have been introduced to make use of domain knowledge, with ontology reasoning, for generating user-friendly navigation experiences. A prototype, named SWC, and its architecture have been described. Several metrics have been applied or introduced for evaluation of the proposed approach and for an efficient semantic content organization. The approach has been evaluated along a concrete scenario and with respect to precision, performance, network traffic, and usability. The evaluation results suggest that the proposed approach decreases network traffic as well as the amount of information presented to the users without requiring significantly more processing time, and that it allows creating a satisfactory navigation experience.

The future work, firstly, involves investigation of new heuristics for enhancing navigation experience. Secondly, from the interaction point of view, annotating interactional elements (i.e., HTML forms) will lead us to full realization of our approach; techniques for re-formation of annotated interactional elements will be investigated.

Publishers need to be supported with appropriate tools in order to automate the annotation process. Such tools might employ database schemas [15, 35], domain
ontologies (possibly interweaved with database schema [32]), or the website itself by detecting data items [13], even relations, within the (X)HTML documents. Finally, we emphasize that the proposed approach can be used for various devices; it has been evaluated for a mobile scenario for simplicity. However, in UbiComp environments, support for different modalities (e.g., haptic interfaces etc.) are important, automated approaches are required for adapting the navigation experience with respect to modality of the end-user interface (e.g., [22] for IPTV).

Acknowledgments. This research is conducted within the project ‘Harnessing collective intelligence in order to make e-learning environments adaptive’ (IOF KP/07/006). Partially, it is also funded by the EC’s IST-FP7 under grant agreement no 231396 (ROLE project).

References


2.4 Mashups by Orchestration and Widget-based Personal Environments: Key Challenges, Solution Strategies, and an Application

**Authors:** Ahmet Soylu, Felix Mödritscher, Fridolin Wild, Patrick De Causmaecker, and Piet Desmet

**Published in:** Program: Electronic Library and Information Systems, volume 46, issue 3, 2012. (in press)

I am the first author and only PhD student in the corresponding article. I am the main responsible for its realizations. The co-authors provided mentoring support for the development of the main ideas.

**Earlier versions were published in:**


Mashups by Orchestration and Widget-based Personal Environments: Key Challenges, Solution Strategies, and an Application

Ahmet Soylu¹, Felix Mödritscher², Fridolin Wild³, Patrick De Causmaecker¹, and Piet Desmet¹

¹ KU Leuven, Kortrijk, Belgium
² Vienna University of Economics and Business, Vienna, Austria
³ The Open University, Milton Keynes, United Kingdom

Mashups have been studied extensively in the literature; nevertheless, the large body of work in this area focuses on service/data level integration and leaves UI level integration, hence UI mashups, almost unexplored. The latter generates digital environments in which participating sources exist as individual entities; member applications and data sources share the same graphical space particularly in the form of widgets. However, the true integration can only be realized through enabling widgets to be responsive to the events happening in each other. We call such an integration widget orchestration and the resulting application mashup by orchestration. This article aims to explore and address challenges regarding the realization of widget-based UI mashups and UI level integration, prominently in terms of widget orchestration, and to assess their suitability for building web-based personal environments. We provide a holistic view on mashups and a theoretical grounding for widget-based personal environments. We identify the following challenges: widget interoperability, end-user data mobility as a basis for manual widget orchestration, user behavior mining - for extracting behavioral patterns - as a basis for automated widget orchestration, and infrastructure. We introduce functional widget interfaces for application interoperability, exploit semantic web technologies for data interoperability, and realize end-user data mobility on top of this interoperability framework. We employ semantically enhanced workflow/process mining techniques, along with Petri nets as a formal ground, for user behavior mining. We outline a reference platform and architecture, compliant with our strategies, and extend W3C widget specification respectively - prominently with a communication channel - to foster standardization. We evaluate our solution approaches regarding interoperability and infrastructure through a qualitative comparison with respect to existing literature, and we provide a computational evaluation of our behavior mining approach. We realize a prototype for a widget-based personal learning environment for foreign language learning to demonstrate the feasibility of our solution strategies. The prototype is also used as a basis for the end-user assessment of widget-based personal environments and widget orchestration. Evaluation results suggest that our interoperability framework, platform, and architecture have certain advantages over the existing approaches and proposed behavior mining techniques are adequate for the extraction of behavioral patterns. User assessments show that widget-based UI mashups with orchestration (i.e., mashups by orchestration) are promising for the creation of
personal environments as well as for an enhanced user experience. This article provides an extensive exploration of mashups by orchestration and their role in the creation of personal environments. Key challenges are described, along with novel solution strategies to meet them.

1 Introduction

The plethora of applications and information available on the Web is overwhelming and calls for efficient approaches to access and organize them. There has been a long-standing research on adaptive systems that can reflect on the needs and contexts of users. Acquired techniques have been applied successfully on the development of a variety of adaptive web systems [1] providing context-tailored user experiences in terms of content (e.g., [2]), as well as, presentation (e.g., interface [3]), and behavior (e.g., [4]). Adaptation and personalization, however, is only one side of the coin in which the user environment is considered to be a mere input for the user experience. On the other side, with a constructivist approach [5-6], Wild et al. [7] consider the environment as an output of the user experience. Moreover, taking the open nature of context into account [8], it is impossible to define adaptation rules for all eventualities [9]. In keeping with this perspective, providing end-users with appropriate means to reflect on their own state of affairs becomes an integral requirement. In this article, we are interested in the realization of personal (digital) environments. We define a personal environment as an individual's space of applications, data sources etc. in which she is engaged on a regular basis for personal and/or professional purposes. The individual at the centre modifies her environment through interacting with it, intending to positively influence her social, self, methodological, and professional competences and to change her potentials for future action and experience [7].

Considering traditional web experience, users are either tied to a single application (cf. [10]) or they have to manage a set of applications, data sources etc. on their own with limited scaffolding support (e.g., iGoogle) or lack thereof (e.g., with bookmarking [11]). To fill this gap, we consider exploiting the mashup approach. Although the idea of mashups is not new, nowadays it attracts researchers and practitioners more. This is mainly due to the shift and advancements in web technologies, such as Web 2.0, RESTful services, the Semantic Web, widgets etc. (cf. [12-14]). The mashup era has emerged in response to the challenge of integrating existing services, data sources, and tools to generate new applications and gained an increasing emphasis due to the ever-growing heterogeneous application market. In this respect, user interface (UI) mashups play a scaffolding role to enable the creation of personal environments and to support cognitive processes, like fostering reflection, awareness, control and command of the environment. Nevertheless, the large body of work in this area focuses on service/data level integration and leaves UI level integration, hence UI mashups, almost unexplored. Mashups are usually realized either through a seamless integration, in which only the resulting application is known by the end-users, or through integration of original services, data sources, and tools, particularly in terms of widgets, into the same graphical space, in which participating applications and data sources are identifiable by the end-users. The former composes a unified functionality or data presentation/source from the original sources. The latter
generates digital environments in which participating sources exist as individual entities. However, the true integration can only be realized through enabling widgets to be responsive to the events happening in each other. We call such an integration widget orchestration and the resulting application mashup by orchestration.

This article aims at exploring and addressing challenges regarding the realization of widget-based UI mashups and UI level integration, prominently in terms of widget orchestration (i.e., mashups by orchestration), and to assess their suitability for the creation of web-based personal environments. To this end, we provide a holistic view on mashups and a theoretical grounding for widget-based personal environments. We identify the following challenges: widget interoperability, end-user data mobility as a basis for manual widget orchestration, user behavior mining - for extracting behavioral patterns - as a basis for automated widget orchestration, and infrastructure.

We introduce functional widget interfaces (FWI) for application interoperability, exploit semantic web technologies for data interoperability, and realize end-user data mobility on top of this interoperability framework. We employ semantically enhanced workflow/process mining techniques, along with Petri nets as a formal ground, for user behavior mining. We outline a reference platform and architecture, compliant with our strategies, and extend W3C’s widget specification respectively - prominently with a communication channel - to fill the standardization gap and to foster the re-usability of widgets and the development of standardized widget-based environments.

We evaluate our solution approaches regarding the interoperability and infrastructure, through a qualitative comparison with respect to existing literature, and provide a computational evaluation of our behavior mining approach. We realize a prototype for a widget-based personal learning environment (WIPLE) for foreign language learning to demonstrate the feasibility of our solution strategies. The prototype is also used as a basis for the user assessment of widget-based personal environments and widget orchestration.

The rest of the article is structured as follows. Section 2 elaborates on the mashup landscape, widgets, widget-based personal environments, widget orchestration, and notable challenges. In Section 3, the related work is presented. The proposed solution strategies are described in Section 4 and evaluated and discussed with respect to relevant literature in Section 5. Finally, Section 6 concludes the article and refers to future work.

## 2 Mashups, Widgets and Personal Environments

The essence of UI mashups, for personal environments, is that they provide a unified interaction experience over a common graphical space for a collection of distributed digital entities and necessary affordances to blend functionalities of these entities and to stimulate exploration of new possibilities. Therefore, UI mashups are anticipated to play a key role for the realization of personal environments. Being completed with orchestration, they are intended not only to enhance but also to augment the end-user experience. There are particular characteristics which we deem important for personal environments based on UI mashups [15-17]: (1) open: a user can add/remove an entity to his environment at anytime, (2) clustered: a user can organize entities in his
environment into different groups, (3) demand-driven: the behavior of the environment and entities are based on the explicit or implicit demands of the user, and (4) loosely coupled: entities available in the environment are independent from each other. In this context, we consider widgets, grounded on standards, as the building blocks of UI mashups and personal environments due to their promising suitability to meet the aforementioned characteristics. In what follows, a meta-level understanding of the overall picture, with a futuristic perspective, is depicted.

2.1 The Mashup Landscape

Mashups can be categorized into different types. We set forth two particular interlinked perspectives here. On the one hand, we categorize mashups into two types from an end-user point of view: the box type (cf. service/data mashups) and the dashboard type mashups (cf. UI mashups) [15].

![Figure 1: The mashup landscape](image)

The former is realized through a seamless integration combining different applications and data sources into a single user experience, in which only the resulting application is known and perceived by the end-users. The latter is realized through the integration of original applications and data sources, particularly in terms of widgets, into the same graphical space (e.g., browser), in which participating applications and data sources can be perceived and identified by the end-users. On the other hand, from a technical point of view, we categorize mashups with respect to the source and integration approach as depicted in Figure 1. The source-wise categorization includes
(1) service mashups (e.g., [18]), (2) data mashups (e.g., [19]), (3) tool mashups (e.g., [20]), and (4) hybrid mashups (e.g., [21]). Service and data mashups are based on integration of services and data sources respectively. Tool mashups are similar to the service mashups; however, they are based on end-user applications with GUIs and the integration is carried out by extracting and driving the functionality of applications from their end-user interfaces (e.g., HTML forms). Hybrid mashups combine these three sources.

The integration-wise categorization is linked with and similar to the end-user perspective and includes (1) mashups by composition and (2) mashups by orchestration (see Figure 2).

![Figure 2: Integration-wise categorization of mashups.](image_url)

The difference with the end-user perspective lays in its emphasis upon the form of functional integration rather than the end-user perception. The former (i.e., by composition) composes a unified functionality or data presentation/source through a seamless functional integration of aggregated sources. The resulting mashup is a new application and participating sources are components of this new application. The latter (i.e., by orchestration) refers to the integration of aggregated resources into the same graphical space in a way that they are independent from each other in terms of functionality and interface. The source applications and data sources are aggregated possibly in terms of widgets through a manual or automated widgetization process. In this respect, if the end application is a widget, an instance of mashup by composition might become an element of an instance of mashup by orchestration. The resulting mashup is a digital environment in which participating sources exist as individual entities. The true functional integration can only be realized through enabling widgets to be responsive to the events triggered by other widgets. We call such an integration ‘widget orchestration’. Our focus is on (semi-)automatic orchestration, that is enabling a widget platform to learn user behavioral patterns through harnessing event logs and, upon initiation of a pattern, (semi-)automatically executing the corresponding pattern flow, i.e., an ordered, parallel etc. set of widget actions. The automation process also requires communication of relevant data through the flow.

### 2.2 Widget-based UI Mashups for Personal Environments

The idea of widgets has existed in various forms such as badges, gadgets, flakes, portlets etc., and differentiates with respect to the underlying technology, the availability of backend services and so on. In this paper, we are interested in web widgets. Typically, a web widget (cf. [22-23]) is a portable, self-contained, full-
fledged, and mostly client-side application. It is hosted online and provides a minimal set of functionality (with/without backend services) through considerably less complex and less comprehensive user interfaces. Widgets are expected to be re-usable, which is achieved by enabling widgets to be embedded in different platforms satisfying certain standards and specifications (e.g., W3C widget specifications [22]). Various technologies can be used to implement widgets (notably HTML and JavaScript, Java Applets, Flash etc.), but cross-platform and device support is crucial due to re-usability considerations.

Some widgets are developed for generic purposes such as clock, calendar etc. widgets for specific platforms (e.g., for Windows 7), mostly, without re-usability and portability concerns. More advanced widgets are developed for specific purposes either from scratch or as a micro version of already existing applications (see Figure 3). As an example of the latter, Figure 3 shows a web application called ‘mediatic’ (http://www.kuleuven-kulak.be/mediatic/ - a foreign language learning tool providing video materials) and its widgetized form that we have developed. A widget platform is required to support execution of multiple widgets over a common space. A widget-based environment can be populated by an end-user or pre-populated by a programmer or super-user. Widgets can be populated into a single space, or multiple working spaces can be created to cluster related widgets (e.g., one space for language learning, one for entertainment etc.).

During a normal web navigation experience a typical user finds, re-uses, and mixes data by switching between different web applications being accessed either from different browser instances or tabs. Normally, every regular web user generates her own portfolio of applications (implicitly or explicitly - e.g., through bookmarking [11]) over time. Therefore, for a regular user, one can expect to observe behavioral patterns representing regularly performed actions and activities over the applications in her portfolio (e.g., a user watches videos in a video sharing application and
publishes the ones she likes in a social networking application). Widget-based environments can facilitate such a user experience by allowing users to access their portfolios through a common graphical space, where each application or data source is represented as a widget, and allows users to create their own personal environments. At a conceptual level, a web-based personal (digital) environment (cf. [24-26]) can be seen as a user-derived sub-ecosystem where member entities come from the Web (the ultimate medium for the supreme digital ecosystem). This is because, indeed, personal environments can be intertwined with the physical world, since the member entities are not limited to web applications and digital data sources anymore. It includes any physical entity, e.g., devices, people etc., having a digital presence (cf. [27]). A variety of devices, like mobile phones, tablet PCs, intelligent household appliances, etc. are expected to be connected to the Internet (or to local networks through wired/wireless technologies like Bluetooth etc.) and serve their functionalities through Web and Web-based technologies, e.g., via RESTful APIs [28-29]. Therefore, various Internet-connected devices can be part of the personal environments through widgets acting as a medium of virtual encapsulation. This allows users to merge their physical and digital environments into a complete ecosystem (i.e., personal and pervasive) and to organize interaction and data flow between them [21].

In widget-based personal environments, the user experience can be enhanced in several ways, notably, (1) by enabling data, provided by a user or appearing as a result of her actions in a widget, to be consumable by other widgets, particularly, in terms of end-user data mobility (i.e., enabling an end-user to copy content from one widget to another effortlessly) and (2) by automating the execution of regular user actions by learning user behavioral patterns from the logs generated as a result of users actions. We see such interplay between widgets - or web sources in general - as an orchestration process. Widget orchestration can happen in various forms in a widget-based environment. We consider followings of crucial importance: (1) user-driven: users manually copy data from one widget to the other and initiate the target widgets. The manual process can be enhanced through facilitating the end-user data mobility (e.g., select, copy, and paste with drag & drop from widget to widget), (2) system-driven: the system learns user behavioral patterns by monitoring events (each corresponds to a user action) and data emerging as a result and then handles data mapping, transportation, and widget initiation processes (semi-)automatically, (3) design-driven: a programmer, a super-user, or even an end-user pre-codes widget behaviors, e.g., which widget should react to which event and how, and (4) hybrid: similar to the system-driven orchestration, the system learns user behavioral patterns; however, the control is shared (cf. [30-31]) by means of user mediation, recommendations etc. For instance, instead of going for an immediate automation, the user is provided with recommendations and the automation only happens if the user decides to follow a particular suggestion. This paper focuses on system-driven widget orchestration.

However, system-driven widget orchestration faces us with several challenges. (1) Widget interoperability: (a) application interoperability - in order to enable widgets to be responsive to the user actions happening in other widgets, a loose functional integration is necessary. Since widgets are developed by different independent parties, standards and generic approaches are required to ensure loose coupling, (b) data
interoperability - widgets need to share data, particularly during a functional interplay. Since widgets do not have any pre-knowledge about the structure and semantics of data provided by other widgets, standards and generic approaches are required to enable widgets to consume data coming from other widgets. (2) User behavior mining: each user action within widgets can be represented with an event. Algorithms that are able to learn user behavioral patterns, i.e., structure (topology) and routing criteria, from the logged events and circulated data are required along with a formal representation paradigm to share, analyze, and present behavioral patterns. (3) Infrastructure: the abovementioned challenges require any platform, on which the widgets run, to be part of possible solution strategies (e.g., how heterogeneous applications communicate events and data). Standardization will enable different communities and parties to develop their own compliant widgets and platforms. It will enable end-users to create their own personal environments by populating heterogeneous applications and data sources and by orchestrating them. In this respect, specification of a generic communication channel for widgets is of crucial importance to enable integration. The aforementioned challenges are also in the core of the successful realization of other widget orchestration strategies and widget-based environments in general, notably, user-driven orchestration through end-user data mobility.

3 Related Work and Discussion

There exists a considerable amount of work on mashups by composition due to the popularity of web service composition, e.g., [13, 18-20, 32]; however, there is a limited work on UI mashups. At this point, we remark that approaches using visual constructs (e.g., widget like) and programming support for the construction of mashups by composition (e.g., service, data etc.) should be distinguished from UI mashups (e.g., [14, 33]). In conjunction with the rising popularity of widgets and W3C widget specifications (e.g., [22]), the use of widgets for UI mashups and personal environments gained attraction (e.g., [34-37]). Yet, first examples are proprietary such as Yahoo widgets, Google gadgets, Open Social gadgets etc. and integration is mostly limited (i.e., functional interplay). In more advanced approaches, particularly, Intel Mash Maker [38], mashArt [39], and Mashlight [14], IU mashups are developed manually by a developer or a skilled user, notably, with visual programming support (cf. [40]). The main problems with these approaches are that they are mainly design-driven (including data mappings, flow structure and routing) and not truly appropriate for naive end-users. Another handicap is that they mostly provide limited compliance on the W3C’s widget specification family or do not describe any possible extension.

Regarding approaches based on widget interoperability, in the current literature, the trend is towards inter-widget communication (IWC) (e.g., [26, 34-35, 37, 41-43]) in which basically an event is delivered to relevant widgets (i.e., unicast - to a single receiver, multicast - to multiple receivers, or broadcast – to all possible receivers) whenever a user performs an action inside a widget. Multicast and broadcast are the basis of the IWC. In the former, widgets subscribe to particular events and/or
particular widgets etc. and get notified accordingly while, in the latter, events are delivered to all widgets in a platform. In both cases, the receiving widgets decide whether and how they react to an event depending on the event type, content, etc. However, there exist some problems in the IWC. First of all, since widgets have to decide on which events to react and how, they are overloaded with extra business logic to realize responsiveness. Secondly, responsiveness is hard to be realized. Either widgets have pre-knowledge of each other, and hence semantics of the events they deliver, or widgets exhibit responsiveness through matching syntactic signatures of the events delivered. The former approach is not realistic because widgets are developed by different parties and in a broad and public manner. The latter is problematic in terms of its success, since syntactic signatures are simply not enough for a successful identification of relevant events. Thirdly, since each widget acts independently, without any centralized control, it is unlikely to achieve a healthy orchestration. Chaotic situations are most probably to arise in an open environment when several self-determining widgets respond to the same/different events in a distributed and uncontrolled way.

Wilson et al. [44], in their recent work that partially coincides with our predecessor study (cf. [36]), define three UI mashup models supporting widget responsiveness, namely Orchestrated UI mashups, Choreographed UI mashups, and Hybrid UI mashups. Orchestrated UI mashups refer to the case where interactions between widgets are defined explicitly and managed centrally. Event (notification of a user action in a widget) – operation (a widget behavior triggered by an event originating from another widget) mappings as well as data transformations and mappings are pre-defined by a developer (please note the conceptual difference that we have, that is by mashups by orchestration we refer to the general idea of interplaying widgets regardless of how it is realized). Choreographed UI mashups refer to the case where interactions between widgets (i.e., responsiveness) emerge from the individual decisions of widgets. This is achieved through ensuring that each widget complies with a reference topic ontology. Each widget publishes its events with respect to a reference topic ontology and widgets subscribe to the events of interest. Hybrid UI mashups refer to the case where widgets maintain a partial autonomy, that is, the interplay (i.e., responsiveness) between widgets is constrained through the central logic programmed by a developer. These three models inherit previously elaborated drawbacks for possible applications of the IWC approach. Regarding the first model, a central logic, pre-coded by a developer, constraints the open and demand-driven characteristics of personal environments in terms of user experience, that is although new widgets can be added to the environment by an end-user, it cannot truly be part of the user experience before a developer describes its role in the orchestration process. Moreover, the approach becomes inflexible since a developer is required to describe event – operation mappings as well as data mappings and transformations. The second model, being based on a reference topic ontology, is better than a purely syntactic approach, though it does not comply with the demand-driven characteristic of personal environments. One should also take it into consideration that a semantic match between an event and an operation does not guarantee that the emerging interplay is sound. Lastly, a distributed orchestration approach (with or without semantics) complicates the widget development as previously mentioned. Entry barriers for the widget development should be kept minimal. The third model
maintains the drawbacks described for the previous two models and is inflexible since it requires a developer to be under full possession of the environment and the widgets.

In the current examples of widget-based environments, e.g., [7, 26, 34, 37], the idea of interplay between widgets already exists; however, it is either pre-designed or purely based on syntactic or semantic similarities between widgets. Behavioral patterns, which are necessary to comply with the demand-driven nature of personal environments in an automated approach, are not exploited and, due to that, a formal ground for mashups by orchestration is not explored yet. In Srbljic et al. [45], a widget-based environment (i.e., mashups by orchestration) is used for the end-user programming of composite applications (i.e., mashups by composition). The work is particularly relevant since it aims at empowering end-users to program by demonstration which requires learning from end-user behaviors. Each source is represented as a widget and an end-user performs a set of actions over these widgets to achieve the outcome she desires. The actions of the user are monitored and a composite application is generated respectively. The algorithm employed corresponds to a part of a well known workflow mining algorithm (α-algorithm), yet a formal modeling instrument, such as Petri nets, is not utilized. Data mappings as well as the topology of composite application flow (e.g., parallel, sequence etc.) are provided manually by the end-user (with visual programming support tailored for skilled users). Coming back to Mashlight [14], though being a design-driven approach, in terms of the grounding formalism, the authors employ a process model based on a directed graph. Indeed, their work is rather an example of mashups by composition based on widgets displayed in a sequential order. The authors later introduce super-widgets which are indeed containers for multiple widgets activated in parallel. The problem with this approach, in our context, is that their grounding model is proprietary and requires means for validation, verification, and sharing of patterns. In a personal environment based on widgets, facilities related to the data mobility should be designed for naive end-users while topology and routing criteria should be extracted implicitly. A formal ground is a must for validation, verification and sharing of the behavioral patterns to avoid emergence of pathological patterns, to enable possible share between users, and for visualization of the extracted patterns.

Ngu et al. [46] propose an approach that allows composition of Web-service-based and non-Web-service-based components, such as web services, web applications, widgets, portlets, Java Beans etc. Further, they propose a WSDL based approach, enriched with ontological annotations, to describe programmatic inputs and outputs of components in order to allow searching and finding complementary and compatible components. The overall approach allows users to progressively find and add components to realize a composite application through wiring the outputs and inputs of different components. The approach is based on IBM Lotus Expeditor which includes a Composite Application Editor (CAE) and Composite Application Integrator (CAI). CAI is the run-time editor which reads and manages the layout information of composite application and responsible of passing messages between components. The CAE allows assembling and wiring components. The approach presents components with UIs in a common graphical space and follows a data-flow-oriented approach rather than a task-oriented approach (i.e., mashup by composition - which aims at putting together a single business process for the purpose of automating a specific task). In other words, there is no specific begin and end-state, a component
can start executing whenever it receives the required input, and there is no explicit control flow specified. Although the matching mechanism and the way components are put together are still more task-oriented, the data-flow oriented perspective matches the characteristics of personal environments. However, firstly, the presented approach relies on the users for designing mashups and interaction of components. Secondly, the approach does not present any specification regarding event delivery and communication; hence it remains ad-hoc. The proprietary nature of the editing and run-time environment hinders the possibility of wide acceptance of the resulting composition framework and remains weak against more ubiquitous, simple, and standard approaches, e.g., mashups, based on W3C widgets, which can simply run on any standard browser. Thirdly, the proposed approach does not provide any formal means for validation and verification of the compositions to prevent the emergence of pathological mashups. Finally, in the current shape, the proposed approach is more appropriate for creation of task-oriented mashups, mixing functional and interface integration, for enterprises and skilled users rather than personal environments for the end-users.

Regarding the architecture, the existing work is mainly repository-centric [26, 37]. The Apache Wookie server (http://getwookie.org/) is notable in this respect. Wookie does not only host W3C widgets, but also provides basic services such as inter-widget communication (over a server-sided communication mechanism), preference management etc. Widgets access services, which are provided by the widget server, through containers in which the server places them before the delivery. Such a centralized approach is inflexible and overloads the repository by aggregating services and tasks, that should normally be provided by a client-side run-time system, to itself. Such an approach is not appropriate for a heterogeneous environment since widgets coming from different repositories cannot communicate. We believe that any possible architectural decision should be taken in compliance with existing specifications. Prominently, the W3C’s widget family of specifications provides a set of standards to author, package, digitally sign, and to internationalize a widget package for the distribution and the deployment on the Web. Yet, the W3C’s widget specification and the standardization process still being active, there remains major room for extensions towards realizing mashups by orchestration. More specifically, with respect to the challenges described in Section 2.2, extensions are required for communication infrastructure, event delivery, functional integration, and data mobility. In Wilson et al. [44], the authors propose to extend the W3C’s widget specification family with communication support and also with means to enable widgets to disclose their functionality for programmed orchestration support. We agree upon these extensions as asserted by our earlier work [36] with the difference that we the extract control logic implicitly from the user interactions. These extensions will be described in detail in Section 4.

4 Solution Strategies

Traditional UI mashups (e.g., [14, 46]) are usually compositional and enterprise-oriented. They compose Web APIs and data feeds into new applications and data
sources to typically serve specific situational (short-lived) needs of the users in a task-oriented manner (cf. [47]). Each mashups by composition always have a particular task as a goal and require some manual/automated development process. Considerable amount of effort has been spent towards creating development environments, addressing skilled users to novice users (cf. [48]), to enable effortless composition, with support for data mapping, flow operations, user interface combination etc. However, for personal environments, mashups should be employed for the orchestration of a dynamic and heterogeneous set of applications, with respect to the active data-flow, in a user-oriented manner. This is comparable to traditional desktops where users run a set of applications and manually blend their functionalities. Therefore, Mahsups by orchestration follow a experience-oriented perspective and allow users to populate various applications and to orchestrate them spontaneously. There is not only a single task in mind. There is no specific start operation, ending operation, pre-defined control flow etc. In this respect, our task is not to come up with a development environment; however, what we target is a platform (e.g., operating system) which allows aggregation of applications in form of widgets and enables user-oriented widget interaction. We first provide an interoperability framework; the interoperability framework is combination of specifications for functional interfaces for widgets and semantic annotation of content, events, and interfaces in order to address application and data interoperability considerations respectively. Secondly, we propose an end-user data mobility facility, built on top of the interoperability framework, to enable the user-driven data exchange between widgets. This is particularly important since, contrary to data exchange between services, output of an application with UI (a widget in our case) is not necessarily programmatic and not all the returned content is relevant to the need of end-user. Thirdly, we propose and specify a communication channel and standard run-time services (e.g., event delivery, preference management etc.) for a widget platform, along a reference architecture to facilitate rapid realization of widget-based personal environments. Finally, we provide an algorithmic solution to enable widget platform to learn commonly executed end-user patterns to automate the interplay between widgets with respect to the events occurring as a result of user interactions at UI level.

Our perspective, for personal environments, is to enable end-users to populate their own spaces and to organize and orchestrate the available entities with respect to their changing needs. Expectedly, a design-driven model and others have their places as well (e.g., when specific experiences are required to be designed). However, with a fundamentalist approach, we first define the most generic and suitable model on which more specific models can be built. Otherwise, results are more likely to be proprietary. In this respect, the realization of an orchestration model with an open and demand-driven characteristic allows more specific models and design tools (e.g., design-driven) to be realized upon by imposing new constraints (cf. [14]). Here, the path we follow is to first empower end-users with generic facilities to realize user-driven orchestration, later to enable system to extract behavioral patterns from the user interactions to realize system-driven orchestration, and finally to enable the system to mediate with the end-user and/or to provide recommendations in order to realize a facilitated orchestration experience (cf. [30] - note that the latter is not within the scope if this article).
Technically, to realize our approach, we first empower the end-users with facilities to communicate data from one widget to another (i.e., data mobility). Each widget notifies the platform, through a communication channel, whenever a user action occurs, including data exchanges. The platform stores events into the event log and monitors the log for a certain time to extract behavioral patterns. A behavioral pattern is a partial workflow with a flow structure and routing criteria. We define functional interfaces, which allow widgets to disclose their functionalities, so that the platform can automatically execute extracted patterns. The use of domain knowledge, along ontological reasoning support, and standard vocabularies, for enhancing event signatures, functional interfaces, and widget content including interactional elements (e.g., forms), improves the pattern mining (i.e., extraction) process as well as data mobility. We prefer to use and extend the W3C’s widget family of specifications due to ubiquity, simplicity, and device-natural characteristics of its underlying technology (i.e., HTML, JavaScript), yet the overall strategy and proposed approaches remain generic. The foremost advantage of our approach is that widgets and widget development remain simple and complicated orchestration tasks are delegated to the platform. One can also build a design-driven model or a widget-driven distributed orchestration model on top of the main instruments of the proposed model (i.e., functional interfaces, data mobility and communication infrastructure) while keeping in line with a standard-oriented approach (cf. [14, 34, 44]).

4.1 Widget Interoperability and End-user Data Mobility

Regarding the application interoperability, the proposed strategy is that widgets disclose their functionalities through standardized client-sided public interfaces (e.g., JavaScript APIs) which we call functional widget interfaces (FWI) as shown in Figure 4.

![Figure 4: Functional Widget Interfaces (FWI).](image)

FWI allows the corresponding platform to control widgets through functional interfaces. Each function corresponds to a user action within a widget that generates an event when triggered. Event notifications and control requests are communicated between the platform and the widgets, through a communication channel, over services provided by the run-time system of the platform (see Section 4.2 for the
Widgets can share the functionality of their APIs with the platform through a handshake process performed over a standard interface function (e.g., with WSDL) or it can be extracted from the event logs. The latter requires functionality provided with GUls and APIs to be identical. The former is required for a design-driven approach where functionality provided by the widgets should be available to the users (e.g., programmer, super-user, end-user etc.) directly.

An example is given in Figure 5. In this example, there are two widgets in a user’s environment, namely, ‘mediatic’ and ‘flickr’ (a widget that we have developed for a web 2.0 tool that is used to store, sort, search and share photos online – see www.flickr.com). The user watches a video material from the ‘mediatic’ widget with sub-titles, and when clicking on certain words of the text (e.g., the word ‘car’ in Figure 5), the ‘mediatic’ widget delivers an event to the platform. The platform decides on an appropriate widget to react on this event based learned patterns. In this case the ‘flickr’ widget is selected. The relevant event data are extracted and communicated to the ‘flickr’ widget with the desired functionality. The ‘flickr’ widget executes the request by fetching and displaying images relevant to the word of interest.

Concerning the data interoperability, the use of domain knowledge or generic/domain-specific vocabularies enhances interoperability as well as end-user data mobility. For instance, in Figure 5, the ‘mediatic’ widget announces an event informing that the noun ‘car’ is clicked and the ‘flickr’ widget is selected to respond although it accepts strings, which are of the word type. This is because, an ontological reasoning process asserts that the noun ‘car’ is an instance of the class ‘word’ since the class ‘noun’ is a subclass of the class ‘word’ as declared in the grounding ontology. A semantic approach also enhances the behavior mining as described in Section 4.3. For this purposes, on the one hand, we enhance events and function signatures with domain ontologies or vocabularies as shown in Figure 6.
On the other hand, we annotate widget content including interactional elements (e.g., forms) with domain knowledge in order to enable end-users to copy content from one widget to another by simple clicks (i.e., data mobility). Each annotated content piece is visually marked to support end-users with necessary affordance cues.

An example is depicted in Figure 7, for two widgets, namely, ‘dafles’ (a widget that we have developed for an online French dictionary – see http://ilt.kuleuven.be/blf/) and ‘dpc’ (a widget that we have developed for an online multilingual parallel corpus – see http://www.kuleuven-kortrijk.be/DPC/). A user looks up for the meaning of a French word in the ‘dafles’ widget and decides to see example sentences as well as their English translations. Therefore, she clicks on the
marker of one of the items (data chunk) of the result list, and copies that item to the ‘dpc’ widget by clicking on the marker of the target form.

**Figure 8:** Semantic data is extracted from an annotated HTML content (source).

```html
<span about="www.dafles.com#r1" typeof="ll:verb">
<span property="ll:text">abandonner</span>
<span property="ll:language">FR</span>
</span>

**N-Triple: dafles**

5. `<dafles:r1> <rdf:type> <ll:verb>.`  
6. `<dafles:r1> <ll:text> "abandonner".`  
7. `<dafles:r1> <ll:language> "FR".``

**Figure 9:** Semantic data is extracted from an annotated HTML form (target).

```html
<span about="param:p" typeof="ll:word">
Zoek:  
<input id="woord" type="text"/>
Taal1  
<select id="taal1">  
<option value="EN">EN</option>...  
</select>
Woordsort  
<select id="woordsoort">  
<option value="noun">Noun</option>...  
</select>
</span>

**N-Triple: dpc**

19. `<param:p> <rdf:type> <ll:word>.`  

We use embedded semantics technologies for in-content annotation (cf. [49-50]), e.g., microformats, RDFa, microdata, eRDF etc. They can be used for structuring content (i.e., with types and data type properties), interlinking content elements (cf. [50] - i.e., a form of linked-data - with object type properties), and embedding high-level domain semantics (e.g., class – subclass relationships). Figure 8 and Figure 9 show excerpts from annotated HTML content (with RDFa, cf. [49]) of the ‘dafles’ and ‘dpc’ widgets and extracted semantic data in simplified N-Triples format. The excerpt shown in Figure 8 and Figure 9 belongs to the user-selected data item (cf. the
left-hand side of the Figure 7), and the target HTML form (cf. the right-hand side of the Figure 7) respectively. The visual marking of annotated content including interactional elements is handled through a specific widget plugin that we have developed. The plugin observes content changes and, upon each change in content, marks annotated content pieces. Each marking is associated with a standard event described in what follows.

In order to copy a user-selected data chunk from a source widget to a target widget, a special event ‘dataSelected’ (i.e., copy) is introduced to inform the platform. This standard event is only associated with the markings of non-interactional content pieces and communicates the selected data chunk as an event payload. The extracted data indeed forms a small RDF graph. Later, the user clicks on the marker of the target HTML form, and the target widget informs the platform with a special event ‘formSelected’ (i.e., paste). This standard event is only associated with the markings of interactional content pieces and communicates the data extracted from the target content piece (i.e., HTML form) as an event payload. The extracted data indeed can be represented as a (partially) empty graph.

Figure 10: The target HTML form is transformed into a SPARQL query for graph matching.

Consequently, data mobility is achieved through graph matching (see Figure 10). In order to do so, we first transform the empty graph into a SPARQL query (cf. Figure 10). We introduce a specific name space (http://itec-research.be/ns/param with prefix ‘param’) to define variable type resources in order to annotate form fields as shown in Figure 9. This is an open and empty name space. Identifiers of the variable
type can be defined under this name space at the time of authoring and the scope of each identifier is limited to the subject document. It is required that the form author keeps variable resource identifiers and corresponding form field identifiers identical. In this respect, the transformation starts with converting each variable type resource (e.g., “<param:p>”) in each RDF statement of the triple set into SPARQL variables (e.g., “<param:p> <rdf:type> <ll:word>” becomes “?p rdf:type ll:word.”). This transformed triple set, as a whole, is used to construct the WHERE clause of the SPARQL query and, finally, we construct the SELECT clause from the variable type objects of each RDF triple. We execute the resulting SPARQL query over the first RDF graph with ontological reasoning support. As a result, the empty graph is matched with the former RDF graph and the values of the form fields are set with the data matched from the source widget. In Figure 8 and Figure 9, the ‘ll’ prefix stands for the namespace associated with the sample language learning ontology.

Once the run-time platform resolves the target form field values, it communicates these values to the target widget through a specified FWI function (see Section 4.2 for details). This widget specific function sets the values of form fields respectively by exploiting the fact that variable resource identifiers and form field identifiers are identical; however, one can also exchange HTML form identifiers within event and control messages for more complex widgets (e.g., when there is more than one HTML form in a single interface etc.). Multiple paste events can be executed over the extracted source data (i.e., can be copied into several target forms) as far as data is not overwritten by a new copy event.

4.2 Platform, Framework and Architecture

The platform is composed of two primary layers (see Figure 11), namely a run-time system and a backend system. The run-time system resides at the client (e.g., browser) and is responsible for the operational tasks and the delivery of standard platform services (e.g., preference management, cf. [22]) to the widget instances. The backend system resides at the server side and is responsible for the persistence and decision-making.

The run-time system and backend system are composed of different components. Regarding the run-time system: (1) Widget containers (e.g., a HTML frame), in our context, hold widget instances in the user space and bridge communication ends of widget instances and the environment. Triggers for basic facilities, related to the presence of a widget instance in the environment such as remove, close, minimize, pin, move etc., are attached to the containers. (2) The environment controller manages presence related facilities, such as (absolute/relative) widget positioning, for widget instances over the widget containers and is responsible for the introduction of new sub-spaces, repositories, and widget instances (widgets from repositories or standalone widgets from the Web). (3) The communication channel allows bidirectional communication between the widget instances and the environment. Widget instances communicate events, preferably preferences and data access requests as well, to the platform, and the platform communicates data and control commands for orchestration to the widgets through the communication channel.
Figure 11: The platform architecture – (1) run-time system/environment, (2) backend system, and (3) the Web.

(4) The run-time system core provides standard system services to the widget instances, particularly, for preference management through (4a) the preference management service, for event delivery through (4b) the event management service, and for data access requests to widget backend services through (4c) the data access service using (5) the proxy agent. The core coordinates the orchestration through (4d) the adaptation controller by submitting control commands to the widgets over the communication channel. The adaptation controller handles data mediation and transportation and can utilize a light-weight (6) client-side reasoner for this purpose (e.g., JSW toolkit http://code.google.com/p/owlreasoner/, EYECClient http://n3.restdesc.org/rules/generalized-rules/). The adaptation controller can also submit re-positioning requests to the environment controller (e.g., in order to move involved widgets closer in course of an active interplay). We consider this facility particularly important, since there exists some interdependencies between location expectations of widgets as suggested by Gali and Indurkhya [51].
Regarding the backend system components, (1) the manager handles preference persistence through (1a) the preference manager and the state of the environment (e.g., widgets, widget positions etc.) through (1b) the widget manager. (1c) The context manager stores event logs and any other contextual information for context based adaptation (cf. [52]). (1d) The adaptation manager decides on adaptation rules (i.e., control commands), particularly through learning behavioral patterns, and submits them to the adaptation controller. It utilizes a (2) server-side reasoner. (3) The proxy is responsible of retrieving data from external data sources (mainly from widget back-end services) upon receiving a dispatch request, initiated by a widget, from the proxy agent of the run-time system.

We prefer to detail the specification of important components of the platform and architecture by providing examples from our own prototype.

```javascript
function CommunicationChannel()
{
    var Channel = function()
    {
        this.send = Send;
        this.addCallListener = AddCallListener;
    };

    function Send(receiver, message)
    {
        if (receiver == 'parent')
            window.parent.postMessage(message, '*');
        else if ... // to subscribed/all widgets
        else if ... // to a specific widget
    };

    function AddCallListener()
    {
        var onmessage = function(e)
        {
            var message = e.data;
            var origin = e.origin;
            var targetFunction = extractTarget(message);
            var fn = widget.shortName + '.' + targetFunction;
            eval(fn)(message);
        };
    }

    if (typeof window.addEventListener != 'undefined')
    {
        window.addEventListener('message', onmessage, false);
    } else if (typeof window.attachEvent != 'undefined')
    {
        window.attachEvent('onmessage', onmessage);
    }

    channel = new Channel();
    channel.addCallListener();
}
```

**Figure 12:** Widget end of the communication channel.

The communication channel (see Figure 12) and standard services require special attention since they need to be standardized while other components are specific to a platform. The communication channel constitutes the backbone of the platform and the personal environment. For approaches allowing direct widget communication, the
Mashups by Orchestration and Widget-based Personal Environments

A communication channel allows communication between local and remote widgets through the platform. Run-time system services (e.g., preference, data access, event delivery etc.) are mainly built on top of the communication channel, hereby allowing us to come up with a non-complex and generic platform and architecture. The communication channel consists of two ends that is a run-time end and a widget end (for each widget). In Figure 12, the widget end of the communication channel is shown. The run-time end of the communication channel is similar to the widget end. The communication channel is based on the ‘window.postMessage’ method of HTML 5 allowing cross-origin communication (realize that widget sources are mostly distributed). It provides a method named ‘channel.send’ for event and request delivery (e.g., preference, data access etc.). This method accepts two arguments, a ‘receiver’ and a ‘message’. In our context, the receiver is only the parent (that is the platform) for a widget; however, for other models, one can also distribute events to a specific widget or to a set of subscribed/all widgets.

<table>
<thead>
<tr>
<th>Message format</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ( // message body</td>
</tr>
<tr>
<td>2 &quot;message&quot;:{</td>
</tr>
<tr>
<td>3 &quot;messageType&quot;:&quot;type of message&quot;,</td>
</tr>
<tr>
<td>4 &quot;messageName&quot;:&quot;name of the message&quot;,</td>
</tr>
<tr>
<td>5 &quot;returnFunction&quot;:&quot;return function name&quot;,</td>
</tr>
<tr>
<td>6 &quot;targetFunction&quot;:&quot;target function name&quot;,</td>
</tr>
<tr>
<td>7 &quot;data&quot;:{ // message payload</td>
</tr>
<tr>
<td>8 { &quot;key&quot;:&quot;value&quot; },</td>
</tr>
<tr>
<td>9 . . .</td>
</tr>
<tr>
<td>10 {</td>
</tr>
<tr>
<td>11 &quot;entity&quot;:{</td>
</tr>
<tr>
<td>12 {</td>
</tr>
<tr>
<td>13 &quot;?x&quot;:{ URI },</td>
</tr>
<tr>
<td>14 &quot;?y&quot;:{ URI },</td>
</tr>
<tr>
<td>15 &quot;?z&quot;:{ URI OR</td>
</tr>
<tr>
<td>16 &quot;label&quot;:&quot;a label&quot;,</td>
</tr>
<tr>
<td>17 &quot;lang&quot;:&quot;lang&quot;,</td>
</tr>
<tr>
<td>18 &quot;dtype&quot;:&quot;dtype&quot;</td>
</tr>
<tr>
<td>19 }</td>
</tr>
<tr>
<td>20 },</td>
</tr>
<tr>
<td>21 . . .</td>
</tr>
<tr>
<td>22 ]</td>
</tr>
<tr>
<td>23 },</td>
</tr>
<tr>
<td>24 . . .</td>
</tr>
<tr>
<td>25 }</td>
</tr>
<tr>
<td>26 }</td>
</tr>
<tr>
<td>27 )</td>
</tr>
</tbody>
</table>

Figure 13: Message format for communication.

The message argument is indeed composite and represented in JSON format as key-value pairs, see Figure 13. A message is composed of a message body and a payload. Regarding the message body, a message type is mandatory. In our context, the following message types are required to be used by a widget: ‘event’, ‘preferenceSet’, ‘preferenceGet’, ‘handShake’, and ‘access’. Regarding the platform,
control’, ‘preference’, ‘handshake’, and ‘dispatch’ message types are required. A return function and/or target function might be required depending on the message type. The latter specifies a function/procedure (i.e., name or alias) in the receiver application that needs to take care of the received message while the former specifies a function in the sender application that the target application must return its response to.

The message types are described as follows; unless otherwise noted target and return functions are not required. The ‘event’ type messages are used to deliver user actions in widgets and an event name needs to be specified by using ‘messageName’ key. There are two special events with reserved names ‘dataSelected’ (i.e., copy) and ‘formSelected’ (i.e., paste) as stated in Section 4.1. A return function needs to be specified, for only the event named ‘formSelected’, in order to deliver values for matched data items. The ‘preferenceSet’ and ‘preferenceGet’ are used for the preference persistence, that is, for storing and retrieving respectively. The ‘handShake’ message type is used by a widget to deliver its FWI function set to the platform. The ‘access’ message type is used by a widget for data access (through the proxy of platform); a return function needs to be specified for data dispatching. The ‘control’ type messages are used by the platform to send control commands to widgets; a target function needs to be specified for control messages. The ‘preference’ type messages are used by the platform to deliver stored preference values to the requesting widgets. The ‘handshake’ message type is used by the platform to request the function set of a widget’s FWI. Finally, the ‘dispatch’ message type is used by the platform to return the requested data to the requesting widget; a target function is required.

Regarding the message payload (i.e., data), it follows the same key-value approach; however, the message payload is not limited to a syntactic key-value structure, particularly for ‘event’ and ‘control’ type messages, as explained in Section 4.1. Semantic data (i.e., in form of typed entities) can be exchanged as well with the reserved key ‘entity’ where each entity is a set of RDF statements serialized in JSON format (cf. Figure 13).

Figure 14: Functional widget interface of an example widget.

In our context, control commands are derived through analyzing the event log by the platform and a FWI is the point where the platform re-generates events with respect to the extracted patterns. Therefore, each function in a FWI (see Figure 14, for
an example) corresponds to an event (with same name and signature). We also allow widgets to send their FWI function set along the function signatures to the platform with a handshake process in order to support design-driven and other models. The function signature follows the same semantic approach used for the event descriptions. The communication channel activates a listener in order to receive messages (cf. Figure 12) and it dynamically invokes corresponding target procedures, particularly when control commands are received.

We reflect the aforementioned design to the W3C’s widget interface specification as depicted in Figure 15. We define a new attribute named ‘channel’ for communication, which is made available through initiation of communication channel (cf. Figure 12). An attribute named ‘access’ is defined for cross-origin communication and is built on top of the communication channel (cf. message type ‘access’). The preference feature already exists in the current specification; however, in our case, it is built on top of the communication channel as well (cf. message types ‘setPreference’ and ‘getPreference’). The reason is that the preference and data access requests are handled by the platform.

```java
interface Widget {
    readonly attribute DOMString author;
    readonly attribute DOMString description;
    readonly attribute DOMString name;
    readonly attribute DOMString shortName;
    readonly attribute DOMString id;
    readonly attribute DOMString authorEmail;
    readonly attribute DOMString authorHref;
    readonly attribute Storage preferences;
    readonly attribute unsigned long height;
    readonly attribute unsigned long width;
    readonly attribute Channel channel;
    readonly attribute Access access;
}
```

Figure 15: Extended W3C widget interface specification.

### 4.3 User Behavior Mining and System-driven Orchestration

We build our system-driven orchestration approach on two possible conditions (see Figure 16): (1) two or more widgets can consume the same input data, suggesting that these widgets can run in parallel (cf. Figure 5) and (2) one or more widgets can consume the output of another widget, suggesting that the consuming widgets are sequential to the source widget and parallel to each other (cf. Figure 7). In a user-driven orchestration, a user manually provides (i.e., types) the same set (or sub-set) of data to different widgets and manually copies output data from one widget to another. In the approach described herein, the goal is to learn user behavioral patterns, satisfying one of the aforementioned conditions, and appropriate rules, for firing widgets automatically, from the event log.
Mashups by Orchestration and Widget-based Personal Environments

Figure 16: Possible scenarios for orchestrating widgets: (1) input-input and (2) output-input.

In the context of this paper, a behavioral pattern is a recurring sequence of user actions connected with control-flow dependencies (e.g., sequence, parallel, choice etc.). Each event refers to an action and each action refers to a user executable function of a widget. Events are considered as being atomic and associated with data items. We investigate the use of workflow/process mining techniques [53] to discover user behavioral patterns from the event logs. A variation of the conventional α-algorithm [54] is used to detect patterns and to extract their topologies (i.e., structure). Decision point analysis (e.g., [55]), is used to find the routing criteria at decision points where routing is deterministic with respect to the value of data attributes in the flow.

Workflow mining approaches usually assume that there is a fully connected workflow (process) model to be discovered. In most cases, there is even an a priori prescriptive or descriptive model. In purely event-based approaches, the event log is assumed to be complete [54], that is the log is representative and a sufficiently large subset of possible behaviors is observed. The event log is subject to noise due to rare events, missing data, exceptions etc. The use of frequency tables is the most classical approach to deal with the noise [54]. Existing approaches are focused on the discovery of the topology (i.e., structure), and a few consider how data affects the routing (e.g., [55-56]) by means of decision mining, yet with a syntactic perspective. In behavior mining, there is not an a priori model. It is very unlikely that there is one single connected workflow, but rather there are small fragments (i.e., patterns) representing commonly repeated user behaviors. In order to emphasize this difference, we call these fragments behavioral patterns rather than models. Completeness (in another form), noise, and effect of data on routing are important issues, which we address through the use of ontologies.

In behavior mining, the completeness problem does not exist as such. In workflow mining, parts of a flow structure, existing in reality, might be missing due to unobserved activities. In behavior mining, on the contrary, the subject user constructs the reality and what exists is what we observe. We consider the completeness problem in a different form however, precisely on the basis of exploration. In order to empower users to explore and utilize the full potential of their environments, recommending possible widgets and actions holds a crucial role. Our semantics-based approach enables such recommendations by matching semantics of inputs (i.e., semantic signatures) and/or outputs (i.e., content) of the widgets. Consider an action named ‘lookforMeaning’ of a widget accepting word type entities and an action ‘searchImage’ of another widget accepting noun type entities, since each noun is also a word (i.e., subclass), one might assert that these two actions can be run in parallel.
The same approach is of use to tackle the noise problem. The semantic match between actions can contribute to the frequency analysis as a heuristic factor. The approach also enables us to exploit high-level semantics of the domain in decision point analysis for learning the routing criteria, for instance, by utilizing ontological class types. Assume that a set of consecutive actions are reported along with the syntactic representation of data consumed in each action - see the footprint in situation (1) of Figure 17 for the ‘searchFor’ actions of widgets A, B and C. And assume that this footprint is repeated in the log substantially, then one might conclude that whenever the ‘searchFor’ action is executed in widget A by a particular user, consequently, either widget B or widget C is executed (cf. topology). However, it is not possible to learn the routing criteria, since it is indeed dependent on the type of the entity being searched for. In the situation (2) of Figure 17, the event data is enriched with ontological classes, hereby one might conclude that whenever a noun type word is searched in widget A, the widget B follows next, otherwise widget C follows (cf. routing criteria). Although the class type information can be incorporated as a separate syntactic attribute, an ontology-based approach provides reasoning support such as the classification of classes.

Figure 17: Application of ontologies for behavior mining.

Once behavioral patterns are discovered along their topology and routing criteria, the next step is creating a formal representation of these patterns in order to enable validation, verification, sharing, and visualization. For this purpose, we employ Colored Petri nets (cf. [57-58]) by adopting the approaches presented in van der Aalst et al. [55] and Rozinat et al. [54]. Petri nets are a graphical and mathematical modeling tool providing the ability to create, simulate, and execute behavioral models; its sound mathematical model allows analysis (e.g., performance), validation,
and verification of behavioral models (e.g., deadlock, liveness, reachability, boundness etc.).

![Figure 18: Petri nets for widget orchestration: transitions refer to widget functions.](image)

A Petri net (see Figure 18) is a graph in which nodes are places (circles) and transitions (rectangles). Places and transitions are connected with directed arcs. Petri nets are marked by placing tokens on places and a transition fires when all its input places (all the places with arcs to a transition) have a token. Colored Petri nets [58] are a type of high level Petri nets allowing the association of data collections with tokens (i.e., colored/typed tokens). A data value attached to a token is called token color. Next to each place, an inscription determines a set of token colors acceptable for each place (i.e., colored places) and the set of possible token colors specified by means of a type called the color set of a place (see Figure 19). The inscription on the upper side of a place specifies the initial marking of that place [58]. Input arc expressions (cf. Figure 19) can be used to determine when a transition occurs and transition guards can be used to constrain the firing of a transition (cf. Figure 18).

![Figure 19: The patterns used in automated orchestration: OR and Sequence.](image)

In our context, a token corresponds to data in the flow. All the in/output messages of the widget actions are modeled as colored places, and widget actions themselves are modeled as transitions with input/output places by following adaption of Colored Petri nets for service composition, e.g., [59]. Learned routing criteria can be represented in terms of rules in the form of arc expressions or guards. The work presented in Gasevic and Devedzic [60] introduces a Petri net ontology, and can be
integrated to utilize ontological reasoning and to support the sharing of behavioral patterns (also see Vidal et al. [61]).

The specifics of our approach are presented in the following. Note that the goal is not to mine full behavioral models, but only the fragments of them, and to realize automated orchestration through the extracted fragments. These fragments need to be simple, yet sound, and short in depth. This is because the target of automation is the end-users, and it is not realistic to bring an excessive number of automated actions to a user’s attention. Hereby, in our context, a pattern consists of a triggering action and one or at most two parallel/alternative consequent actions. If there is a single follower action, it is sequential to the triggering action in terms of the execution order. If there are two follower actions, these are sequential to the triggering action in terms of the execution order, and are either parallel to each other or there is a choice in between. The choice between two alternative actions might dependent on constraints (e.g., one widget cannot process a particular type of data attribute) or on the preferences of the end-user. Therefore, we limit our approach to two types of patterns (cf. Figure 19): (1) Multi-Choice pattern (a.k.a. OR-split) and (2) Sequence pattern (cf. [62]). The Multi-Choice pattern allows execution of one or more transitions out of several available transitions. The realization of the Multi-Choice pattern is depicted at the left-hand side of the Figure 19 (e.g., \(B^\text{f1}\) and \(C^\text{f2}\), only \(C^\text{f2}\), or only \(B^\text{f1}\) – the main letter denotes the widget and the superscript denotes the widget function), and is based on input arc expressions where conditions are either mutually exclusive or overlapping (i.e., combined representation of AND-split, XOR-split and OR-split). The Sequence pattern allows execution of a single action after the triggering action without any alternative. In this respect, the goal becomes, for each possible action \(a\), to detect the most frequent two actions that can run upon the execution of the action \(a\) and to extract the decision criteria if there exists a choice in between the selected two actions.

Regarding the pattern extraction procedure, an event log is the starting point. In workflow mining, a log consists of a set of traces and each trace consists of an ordered set of events (i.e., with respect to associated timestamps). Each trace in a log corresponds to a workflow instance (a.k.a. case), and reveals a possible execution path of the target workflow model. Therefore, given a set of traces, which is a sufficiently large subset of possible behaviors, it becomes possible to derive a complete workflow model. However, in our context, the original log indeed consists of a set of user sessions and in every session different pattern instances coexist including arbitrary actions in the form of a continuous series of events. For this reason, the log needs to be processed to generate a meaningful set of traces for each action. Input-input match based patterns and output-input match based patterns are processed separately to extract traces for the sake of simplicity. For the former, we split the log into a set of fragments for every possible triggering action \(a\). More specifically, fragments for action \(a\) are constructed by taking each occurrence of event \(a\) in the log along \(z\) number of predecessor events and \(z\) number of successor events (\(z\) is the window size that can be defined with respect to the total number of actions, \(z=2\) in our experiments). Data associated with each event in a fragment is matched with data associated with action \(a\) where an event should consume the same or a subset of the triggering action \(a\)'s data. Typed entities are compared with the subsumption test where a subclass relationship should hold (either direction). Events that do not match,
any repetitive events, and any re-occurrence of event $a$ are removed from the corresponding fragment. Each resulting fragment represents a trace for action $a$. Traces based on output-input matches are special, and they are based on specific events: ‘dataSelected’ (i.e., copy) and ‘formSelected’ (i.e., paste) (cf. Section 4.1). This is because these two events are one of the best means to capture patterns based on output-input matches. Considering that output of a widget is mostly not a single data chunk, rather a set of data chunks, an output–input match detection would only be possible through comparing all output data of a widget with the candidate widget’s input attributes. However, the end-user data mobility facility does not only enhance the user experience but also allows us to detect output-input matches by enabling us to identify the user selected data chunk. The resulting traces are in the form of a series of events consuming the same set (or subset) of data for the input-input match based traces and in the form of a series of paste events following a single copy event (e.g., ‘dataSelected’, ‘formSelected’, ‘formSelected’…) for output-input match based traces. The trace extraction process also eliminates the noise emerging from the arbitrary user events.

After the trace extraction process, the task is to identify the most frequent two follower events for each action, hence actions associated with these events. For this purpose, we employ a substantial variation of the frequency analysis used for the $\alpha$-algorithm [63]. Let $W_a$ be a trace set, extracted from the user log $W$ for the action $a$, over the set of actions $L$. Let $a, b \in L$; $a \gg \gg \gg w b$ if and only if there is a trace $\sigma_a = t_1 t_2 t_3 \ldots t_{n-1}, i \in \{1,\ldots,n-2\}$, and $j \in \{2,\ldots,n-1\}$ such that $\sigma_a \in W_a$, $t_i = a$, $t_j = b$, and $i < j$. $a << \gg w b$ if and only if there is a trace $\sigma_b = t_1 t_2 t_3 \ldots t_{n-1}, i \in \{1,\ldots,n-2\}$, and $j \in \{2,\ldots,n-1\}$ such that $\sigma_b \in W_a$, $t_j = b$, $t_i = a$, and $i < j$. A metric indicating the strength of the frequency between action $a$ and action $b$ (denoted with $\#A \rightarrow B$), where $a$ is the triggering action and either $a \gg \gg \gg w b$ or $a << \gg w b$ holds, is calculated as follows. For the input-input based traces, if action $a$ occurs before action $b (a \gg \gg \gg w b)$ or action $b$ occurs before action $a (a << \gg w b)$ in a trace and $n$ is the number of intermediary events between them, $\#A \rightarrow B$ frequency counter incremented with a factor of $\delta^n$ ($\delta$ is frequency fall factor $\delta$ [0.0…1.0], in our experiments $\delta = 0.8$). The contribution to the frequency counter is maximal 1, if action $b$ occurs directly after or before action $a$ ($n=0$ and $\delta = 1$). For the output-input based traces only the $a \gg \gg \gg w b$ relation exists where $a$ is the triggering copy action. After processing the whole trace set of action $a$, the frequency counter is divided by overall frequency of the action $a$ denoted by $\#A$. This process is repeated for every action over its associated trace set. For each triggering action $a$, two follower actions, with the highest frequency factor above a specified threshold $th$, is selected (in our experiments $th=0.45$). The value of $th$ can be adjusted with respect to the total number of actions or can be set for each triggering action individually (possibly with respect to the number of potential follower actions). It is also possible that no action or only one action is above the threshold. For the former, no follower action is selected and for the latter the pattern becomes a sequence in terms of the execution order (i.e., in a input-input based pattern, theoretically a follower action is parallel to the triggering action; however, it is sequential to the triggering action in terms of the execution order since it is only executed once the triggering action occurs). For every action $a$, for which at least one follower action is selected, a decision point analysis (cf. [55]) is conducted for determining the firing conditions. Following
this idea, every decision point becomes a classification problem, that is classifying a given event instance into a class, representing a follower action with respect to previous observations (i.e., training data). However, the decision point analysis employed by Rozinat et al. [55] does not consider Multi-Choice patterns (only XOR – a.k.a. Exclusive Choice); hence, the authors approaches the case as a single label classification problem. We consider the problem as a multi-label classification problem where multiple target class labels can be assigned to an instance. Multi-label classification problems can be handled through problem transformation or algorithm adaptation methods. Problem transformation methods force the learning problem into the traditional single-label classification where algorithm adaptation methods adapt an algorithm to directly perform multi-label classification [64].

\begin{verbatim}
@attribute type {verb, noun}
@attribute lang {en,fr,nl}
@attribute target {dafsels,dpc,dpc-dafsels}

<table>
<thead>
<tr>
<th>Type</th>
<th>Language</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>verb</td>
<td>en</td>
<td>dpc</td>
</tr>
<tr>
<td>verb</td>
<td>fr</td>
<td>dafsels</td>
</tr>
<tr>
<td>noun</td>
<td>en</td>
<td>dpc</td>
</tr>
<tr>
<td>noun</td>
<td>fr</td>
<td>dpc-dafsels</td>
</tr>
<tr>
<td>verb</td>
<td>en</td>
<td>dpc</td>
</tr>
<tr>
<td>verb</td>
<td>fr</td>
<td>dafsels</td>
</tr>
<tr>
<td>noun</td>
<td>en</td>
<td>dpc</td>
</tr>
<tr>
<td>noun</td>
<td>fr</td>
<td>dpc-dafsels</td>
</tr>
<tr>
<td>noun</td>
<td>nl</td>
<td>rest</td>
</tr>
</tbody>
</table>

IF lang=en THEN dpc
IF lang=fr AND type=verb THEN dafsels
IF lang=fr AND type=noun THEN dpc-dafsels

Figure 20: Multi-label classification – the resulting decision tree is transformed into a rule set.

We follow the problem adaptation method, when more than one follower action is selected, and apply a well known data mining algorithm, namely C4.5, for the classification. C4.5 [65] is used to generate decision trees. In the single label classification, a being the triggering action and b, c being the follower actions, an event instance a can be classified into either only b or only c (that is either action b or c follows); however, with the multi-label classification, an event can be classified into only b, only c, or b and c. The approach we use, for the problem transformation, is a variation of the label combination approach [66], which considers events with multiple labels as a new class (e.g., b, c, and b-c). We also add a fourth class named rest referring to negative examples that are not associated either with b or with c. An example training data set, its classification, and rule generation are exemplified in Figure 20 for the triggering action a and selected follower actions b and c. In the training data set (cf. left-hand side of the Figure 20), each line corresponds to an event instance associated with action a, and each column represents a data attribute. The final column represents the action that followed this event instance. If an event a is followed by both action b and c in the corresponding trace (directly or indirectly), the class label for this event instance is set to the combination of labels of follower action
classes (cf. Figure 20). Given a training data, a C4.5 classifier is trained to classify unlabeled instances. Then, the generated decision tree is transformed into a rule set (cf. the right hand side of the Figure 20). Each path from the root node of the decision tree to a leaf node represents a different rule. Note that if a single follower action is selected, we conduct a binary classification with C4.5.

The obtained rules are committed to the run-time platform (cf. Section 4.2). Whenever an event occurs, the associated event data is compared with the antecedents of the rules available in the rule set of the corresponding event. If a rule is satisfied, the action it specifies in its consequent is executed through submitting a control command to the corresponding widget. If the matching rule has a single label as its consequent, other rules in the rule set are also checked. If the matching rule has a multi-label consequent, the remaining rules are not checked. Regarding the subsumption checking for the entity type constraints taking place in the rule antecedents, dedicated approaches such as class hierarchy encoding technique described in Preuveneers and Berbers [67] can enhance the client-side performance.

5 Evaluation and Discussion

We have implemented a partial prototype to prove the applicability of our approaches. Several widgets have been developed for language learning, by adopting the W3C widget specification, with respect to the extensions that we propose; hence, a widget-based personal learning environment (WIPLE) is realized. The widgets we have developed are cross-platform (e.g., Opera platform - https://widgets.opera.com – enables widgets to run on desktop). A demo can be watched online from the following web address http://www.ahmetsoylu.com/pubshare/program2012/.

An example scenario is depicted in Figure 21 where a user watches a video material in the ‘mediatic’ widget and clicks on a particular word in the subtitle. For this particular user and widget action, the mostly used follower actions belong to the ‘dafles’ widget and the ‘dpc’ widget (i.e., input-input match). However, there is a choice in between ‘dafles’ and ‘dpc’ with respect to certain rules (cf. (1) in Figure 21). If the clicked word is in English, only ‘dpc’ follows (‘dpc’ accepts words in English, French, and Dutch whereas ‘dafles’ is a French dictionary accepting only French words); if the word is in French and of the verb type, ‘dafles’ follows; and if the word is a French noun, both widgets follow. Assume that the word the user clicked on is a French noun, e.g., ‘voiture’ (car); in that case, both ‘dafles’ and ‘dpc’ widgets are automatically executed. Afterwards, the user clicks on an output item of the ‘dafles’ widget. For the ‘dafles’ widget, there is only one follower action which belongs to the ‘flickr’ widget (i.e., output-input match) with a condition constraining the type of selected output entity to a noun (cf. (2) of Figure 21). Since the ‘flickr’ widget is not at the near vicinity of the ‘dafles’ widget, first it is automatically moved closer to the ‘dafles’ widget (cf. [51] and (3) of Figure 21). Then, the selected output data chunk is copied to the target ‘flickr’ HTML form (note that the user does not need to click on the marker of the target form). At this point, the user activates the ‘flickr’ widget (i.e., clicks on ‘zoek’ (search) button), and relevant images are retrieved and displayed (cf. (4) of Figure 21).
Figure 21: An example scenario for system-driven widget orchestration.

The orchestration process is semi-automatic because, for output-input matches, the end-user needs to activate the target widget after data is copied. Indeed, this episode can be fully automated or the same approach can be applied for input-input matches (i.e., copying input data to the target widgets without activating the widget action). Regarding the former, it must be checked that the copied data is sufficient to execute the corresponding actions. However, if the target actions realize sensitive operations such as insert, delete etc. over data, the latter approach might be more appropriate. Application of both approaches is independent of the platform, and depends on the FWIs of individual widgets. Note that semi-automatic widget orchestration is not considered to be hybrid orchestration since the latter requires the involvement of users while selecting appropriate actions to execute.

Regarding the graphical environment, each widget container is associated with a set of presence related operations (cf. (5) of Figure 21) such as move, minimize, close, pin, settings etc., and the corresponding visual elements appear when the user moves the cursor over a widget. There exists a task bar where widgets can be minimized to (cf. (6) of Figure 21). The task bar also includes a ‘Widget Store’ tab for adding new widgets, repositories etc. Different workspaces can be created and accessed through the task bar; an alternative can be the use of the browser tabs that will allow a more natural access to different workspaces. Each workspace can be accessed through a distinct URL over different browser tabs.

5.1 Qualities of the Approach

There exist several approaches for widget orchestration. Prominent ones, namely user-driven, design-driven, distributed, system-driven, and hybrid approaches, have
been introduced in Section 2.2 and Section 3. A qualitative comparison of these approaches is given in Table 1 in terms of several major interlinked properties.

Table 1: Comparison of personal environments based on different orchestration approaches.

<table>
<thead>
<tr>
<th></th>
<th>User-driven</th>
<th>Design-driven</th>
<th>Distributed</th>
<th>System-driven</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demand-driven</td>
<td>++++</td>
<td>-</td>
<td>-</td>
<td>+++</td>
<td>+++</td>
</tr>
<tr>
<td>Open</td>
<td>++++</td>
<td>++</td>
<td>+</td>
<td>+++</td>
<td>+++</td>
</tr>
<tr>
<td>Loosely coupled</td>
<td>-</td>
<td>+++</td>
<td>++++</td>
<td>+++</td>
<td>+++</td>
</tr>
<tr>
<td>Clustered</td>
<td>++++</td>
<td>++++</td>
<td>++++</td>
<td>+++</td>
<td>+++</td>
</tr>
<tr>
<td>Simple (Widget Dev.)</td>
<td>++++</td>
<td>+++</td>
<td>++</td>
<td>+++</td>
<td>+++</td>
</tr>
<tr>
<td>Effortless (Orch.)</td>
<td>+</td>
<td>++++</td>
<td>++++</td>
<td>+++</td>
<td>+++</td>
</tr>
<tr>
<td>Sound (Orchestration)</td>
<td>++++</td>
<td>++</td>
<td>+</td>
<td>+++</td>
<td>+++</td>
</tr>
<tr>
<td>Autonomous (Orch.)</td>
<td>-</td>
<td>-</td>
<td>++++</td>
<td>+++</td>
<td>+++</td>
</tr>
</tbody>
</table>

An end-user driven approach (without any end-user data mobility support) is fully demand-driven since the control of orchestration is totally held by the end-users where design-driven and distributed approaches lack this characteristic, since the control logic depends on the perception of the skilled users or programmers. System-driven and hybrid approaches maintain the demand-driven characteristic implicitly because the automation logic is extracted from the end-user logs.

We consider openness in terms of (1) platform and (2) end-user experience. We evaluate openness of the platform in two folds, (a) end-user’s ability to add new widgets to her environment and (b) entry barriers (i.e., commitments) that widget/component providers need to overcome. We evaluate openness of the end-user experience as the level of freedom that an end-user has in the orchestration process. A user-driven approach is expected to exhibit a high level of openness since lesser technical commitments and obligations are required (to standards and specifications) and the experience is driven by the end-user herself. A design-driven approach is weak regarding the openness of the end-user experience since the experience is mostly pre-designed. In terms of the platform, it requires more, albeit not substantial, commitments. A distributed approach is similar to a design-driven approach with respect to the openness of the end-user experience; however, it is ranked lower in terms of openness of the platform since each widget also has to implement its own orchestration logic. System-driven and hybrid approaches are more open compared to other approaches, excluding the user-driven orchestration, in both perspectives. This is due to the fact that the end-user experience is driven by the system with implicit user control. In terms of the platform, they require more, albeit not substantial, commitments.

Loose coupling can be evaluated at several levels, such as physical coupling, communication style, type system, interaction pattern, control of process logic, service discovery and binding, and platform dependencies, as described in Krafzig et al. [68]. For a user-driven approach, we consider loose coupling as a non-existing property because there is no explicit interaction (e.g., data exchange, message communication etc.) between widgets in a user-driven scenario. The level of coupling in other approaches might vary depending on the underlying implementation approach; however, most of the approaches presented in Section 3 follow some blueprints to achieve loose coupling. More specifically, there is no direct physical link between widgets, the communication style is mainly asynchronous, payload semantics
is employed rather than strongly typed interface semantics, a data centric interaction pattern is used etc. The system-driven and hybrid approaches are ranked slightly lower due to the central control logic (which is indeed mined, yet in short-term, these approaches are vulnerable to the structural changes in event payloads and if the central control logic fails the whole system fails). The design-driven approach is ranked lowest mainly due to the static functional binding and highly centralized control logic. All the approaches are based on the existence of standardized widget engines guaranteeing platform and OS independence.

The simplicity of the widget development is indeed dependent on the amount and complexity of the required commitments. A user-driven approach can be ranked higher in this respect since the required commitments are minimal. Except the distributed approach, the remaining approaches can be ranked slightly lower due to higher commitments. A distributed approach is ranked lowest since each widget also has to implement its own logic of responsiveness. This becomes particularly difficult due to synchronization problems (i.e., widgets are independent – cf. Section 3). Note that the commitment to standards and specifications is inevitable primarily due to interoperability considerations; however, the more complex and voluminous a standard/specification, the more difficult it will produce simple and open approaches. Therefore a minimalist approach is required.

We consider effortlessness of orchestration from the end-user perspective (i.e., the level of explicit user effort required for the orchestration). A user-driven approach requires end-users to drive their own experience manually where in other approaches orchestration takes place automatically. Therefore one can consider orchestration process in other approaches easier to be realized compared to a user-driven approach (neglecting the appropriateness of the automated actions for the moment). However, in a hybrid approach, control is shared between the end-user and the system; therefore, a hybrid approach is ranked slightly lower compared to other approaches.

The soundness of orchestration refers to the appropriateness of the automated actions in an orchestration process. A user-driven approach can be ranked highest, since it is the end-user who directly chooses the next action. A design-driven approach and distributed approach is ranked lowest among others because the end-user has no implicit or explicit control in the orchestration process. A design-driven approach is ranked slightly higher compared to a distributed approach, since at least a programmer or skilled user (i.e., a human actor) evaluates possible scenarios and designs the control logic accordingly. System-driven and hybrid approaches are based on implicit user control (cf. behavioral patterns); therefore, they can be ranked higher compared to design-driven and distributed approaches. Yet, compared to a user-driven approach, they are ranked lower, since mined patterns cannot be considered equal to the explicit user request in terms of their reliability and accuracy. A hybrid approach is slightly better than a system-driven approach, since control is shared between the end-user and the system and at certain points explicit user input is requested.

Autonomy of orchestration is regarded as the capacity of an orchestration process to be self-organized, that is its control logic is not explicitly driven by an end-user or programmer. This does not hold for user-driven and design-driven approaches. Other approaches are not directly dependent on a human user for the orchestration logic. A
hybrid approach is ranked slightly lower since at certain points it requires human involvement.

Given the aforementioned comparisons and discussion, system-driven and hybrid approaches are ranked highest in overall. There exist a trade of between two approaches in terms of soundness and simplicity of the orchestration. Since a hybrid approach mediates with the end-user at some points (e.g., in a case of uncertainty), the resulting automation can be more reliable. However, control is shifted towards the end-user. Note that user involvement (cf. [9, 30]) in automation is required at some degree, particularly under ambiguity and when severe implications are probable (e.g., delete, insert actions etc.). A hybrid approach compromises autonomy and easiness of orchestration (from end-user point of view) for the soundness of orchestration.

5.2 Patterns and Decision Points

One of the problems with Petri nets and some of the process mining techniques is the representation and the identification of advanced patterns, for our context, particularly Multi-Choice patterns; the \( \alpha \)-algorithm can only mine some kind of OR patterns while there are other algorithms, such as a multi-phase miner and region based techniques, that can mine larger classes of process models [69-70]. Nevertheless, Multi-Choice patterns even might be reduced to XOR-splits or AND-splits by most of the noise elimination techniques. In our approach, we adopted frequency analysis (that is normally used for noise elimination) to detect the mostly appearing follower actions. The topology and the routing criteria are identified with decision point analysis. Therefore, the limitations regarding Multi-Choice pattern do not apply to our approach. Although Petri nets do not provide any means for explicit representation of OR-splits, since patterns subject to our work are small scale (i.e., at most 3 transitions), the Petri net representation based on AND-split and arc expressions are sufficient (cf. Figure 19).

Probably due to the aforementioned considerations, the decision mining approach, employed for decision point analysis in Rozinat et al [55] and Rozinat and van der Aalst [56], omits OR-splits (i.e., Multi-Choice). The label combination approach that we follow, for multi-label classification, might be problematic since it may lead to data sets with a large number of classes and few examples per class [64]. Let \( L \) be the set of disjoint labels; the number of class labels in a combined label set \( L_c \) is the sum of \( k \)-combinations of \( L \) for every possible value of \( k \) where \( 1 \leq k \leq |L| \). That is combined labels set \( L_c \) increases in polynomial magnitude with respect to \( |L| \):

\[
|L_c| = \sum_{k=1}^{|L|} C(|L|, k)
\]

However, this does not affect our approach due to the small size of \( L \), i.e., \( |L|=2 \) and \( |L_c|=3 \). The affect can be analyzed through two concepts, namely label cardinality and label density, introduced in Tsoumakas and Katakis [64]. Let \( D \) be a multi-label data set with \( |D| \) number of multi-label examples \((x_i, Y_i)\), \( i=1…|D| \) where \( x_i \) refers to a
particular instance and \( Y_i \) refers labels associated with \( x_i \). The label cardinality and label density of \( D \) is defined as follows:

\[
\text{LC}(D) = \frac{1}{|D|} \sum_{i=1}^{|D|} |Y_i| \quad \text{and} \quad \text{LD}(D) = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{|Y_i|}{|L|}
\]

The label cardinality of \( D \) is the average number of labels of the examples in \( D \), and the label density of \( D \) is the average number of labels of the examples in \( D \) divided by \(|L| \) [64]. Our case is a 2-class multi-label classification problem where \(|L|=2\). Each example in \( D \) is associated with at least one label. Therefore, in one extreme (the target pattern is an XOR-split), every example will be associated with one label which will make \( \text{LC}(D)=1 \) and \( \text{LD}(D)=0.5 \), and in the other extreme (the target pattern is an AND-split), every example will be associated with two labels which will result in \( \text{LC}(D)=2 \) and \( \text{LD}(D)=1 \).

**Table 2:** Density comparison of multi-label data sets in which \(|L|>2\) with 2-class multi-label data sets in which \(|L|=2\) (min. density 0.5).

<table>
<thead>
<tr>
<th>Data set</th>
<th>Instance #</th>
<th>Attr. #</th>
<th>Label #</th>
<th>Cardinality</th>
<th>Density</th>
<th>Ratio 0.5/LD(D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bibtex</td>
<td>7395</td>
<td>1836</td>
<td>159</td>
<td>2.402</td>
<td>0.015</td>
<td>33.3</td>
</tr>
<tr>
<td>bookmarks</td>
<td>87856</td>
<td>2150</td>
<td>208</td>
<td>2.028</td>
<td>0.010</td>
<td>50.0</td>
</tr>
<tr>
<td>CAL500</td>
<td>502</td>
<td>68</td>
<td>174</td>
<td>26.044</td>
<td>0.150</td>
<td>3.3</td>
</tr>
<tr>
<td>corel5k</td>
<td>5000</td>
<td>499</td>
<td>374</td>
<td>3.522</td>
<td>0.009</td>
<td>35.6</td>
</tr>
<tr>
<td>corel16k</td>
<td>13811</td>
<td>500</td>
<td>161</td>
<td>2.867</td>
<td>0.018</td>
<td>27.8</td>
</tr>
<tr>
<td>delicious</td>
<td>16105</td>
<td>500</td>
<td>983</td>
<td>19.020</td>
<td>0.019</td>
<td>26.3</td>
</tr>
<tr>
<td>emotions</td>
<td>593</td>
<td>72</td>
<td>6</td>
<td>1.869</td>
<td>0.311</td>
<td>1.6</td>
</tr>
<tr>
<td>enron</td>
<td>1702</td>
<td>1001</td>
<td>53</td>
<td>3.378</td>
<td>0.064</td>
<td>5.3</td>
</tr>
<tr>
<td>EUR-Lex (dc)</td>
<td>19348</td>
<td>5000</td>
<td>412</td>
<td>1.292</td>
<td>0.003</td>
<td>166.7</td>
</tr>
<tr>
<td>EUR-Lex (sm)</td>
<td>19348</td>
<td>5000</td>
<td>201</td>
<td>2.213</td>
<td>0.011</td>
<td>45.5</td>
</tr>
<tr>
<td>EUR-Lex (ed)</td>
<td>19348</td>
<td>5000</td>
<td>3993</td>
<td>5.310</td>
<td>0.001</td>
<td>500.0</td>
</tr>
<tr>
<td>genbase</td>
<td>662</td>
<td>1186</td>
<td>27</td>
<td>1.252</td>
<td>0.046</td>
<td>10.9</td>
</tr>
<tr>
<td>mediannl</td>
<td>43907</td>
<td>120</td>
<td>101</td>
<td>4.376</td>
<td>0.043</td>
<td>11.6</td>
</tr>
<tr>
<td>medical</td>
<td>978</td>
<td>1449</td>
<td>45</td>
<td>1.245</td>
<td>0.028</td>
<td>17.9</td>
</tr>
<tr>
<td>rcv1v2 (subset1)</td>
<td>6000</td>
<td>47236</td>
<td>101</td>
<td>2.880</td>
<td>0.029</td>
<td>17.2</td>
</tr>
<tr>
<td>rcv1v2 (subset2)</td>
<td>6000</td>
<td>47236</td>
<td>101</td>
<td>2.634</td>
<td>0.026</td>
<td>19.2</td>
</tr>
<tr>
<td>rcv1v2 (subset3)</td>
<td>6000</td>
<td>47236</td>
<td>101</td>
<td>2.614</td>
<td>0.026</td>
<td>19.2</td>
</tr>
<tr>
<td>rcv1v2 (subset4)</td>
<td>6000</td>
<td>47229</td>
<td>101</td>
<td>2.484</td>
<td>0.025</td>
<td>20.0</td>
</tr>
<tr>
<td>rcv1v2 (subset5)</td>
<td>6000</td>
<td>47235</td>
<td>101</td>
<td>2.642</td>
<td>0.026</td>
<td>19.2</td>
</tr>
<tr>
<td>scene</td>
<td>2407</td>
<td>294</td>
<td>6</td>
<td>1.074</td>
<td>0.179</td>
<td>2.8</td>
</tr>
<tr>
<td>tmc2007</td>
<td>28596</td>
<td>49060</td>
<td>22</td>
<td>2.158</td>
<td>0.098</td>
<td>5.1</td>
</tr>
<tr>
<td>yeast</td>
<td>2417</td>
<td>103</td>
<td>14</td>
<td>4.237</td>
<td>0.303</td>
<td>1.7</td>
</tr>
</tbody>
</table>

Accordingly, in our context, the label density is minimum 0.5. We compare this minimum label density with the densities of twenty-two open-access multi-label data sets (\(|L|>2\)), from diverse domains, which are available at http://mulan.sourceforge.net/datasets.html (Mulan [71] - an open-source Java library for learning from multi-label datasets). Information regarding the data sets and comparison results are shown in Table 2. The final column (2C ratio) presents 2-class minimum density ratio to the individual densities of the data sets. According to the
results, after excluding the extreme cases, EUR-Lex (ed) and EUR-Lex (dc) data sets, the average ratio is 19.8 (i.e., on average a 2-class multi-label data set is 19.8 times denser than a multi-label data set with |L|>2). The results confirm that, in our context, any 2-class multi-label data set will probably be denser than other multi-label data sets with higher number of labels. This also holds in terms of attribute numbers where a large number of attributes is another reason for sparse data sets. In the data sets presented in Table 2, the number of attributes varies from min. 72 to max. 49060; a widget action is not expected to consume more than 6-7 attributes. Therefore we expect that the approach will not encounter a severe sparse data set problem.

If one wants to perform multi-label classification with a higher number of labels, e.g., where more than two follower actions are necessary, the use of algorithm adaptation methods is preferable; the Mulan library [71] includes a variety of state-of-the-art algorithms for performing several multi-label learning tasks.

At the moment we use an offline learning approach, that is, after collecting a substantial amount of data, patterns and decision rules are generated. However, we face a concept drift problem (cf. [72]) since the distribution underlying the instances or the rules underlying their labeling might change over time (e.g., user’s preferences change). This is because, after certain patterns and rules are learned and put into effect, the event and data occurring later will be the result of the automated actions themselves and changes in user preferences and the environment will not be reflected. Therefore, it is a must to develop methods and techniques to alleviate this problem and probably to enable end-users to communicate inappropriate automations without disrupting the end-user experience by putting end-users under an excessive load. We are also interested in the possibility to generate patterns and rules with on-line learning through data stream mining (cf. [73]) by extracting patterns from continuous data records. There already exist algorithms and frameworks to support data stream mining (e.g., [74]).

5.3 End-user Experiment and Assessment

We conducted a preliminary user experiment and survey in order to test the effectiveness of our approach. The user experiment and survey were performed with four widgets and six users. The profiles of the test users are given in Table 3 (1 for ‘very poor’, 5 for ‘very good’).

The first goal of the user experiment was to test the performance of the mining approach. The experiment was realized in three sessions for each user individually (1.5 hours in total for each user). A five minutes introduction to WIPLE and widgets was given to each user, and users were given the opportunity to get familiar with the WIPLE and widgets before the experiment. The first session consisted of four cycles; at each cycle, the users were given fifteen English words, and asked to comprehend the word by using the existing widgets (i.e., the ‘flickr’ widget that retrieves images associated with a particular word, the ‘mediatic’ widget that allows watching videos with subtitles, the ‘dpc’ widget that retrieves example sentences which include a specified word, and a new widget named ‘engDict’ that is an English-to-English dictionary). However, each cycle had a specified widget that the user had to start off with for each word (e.g., for cycle 2, the user always had to use the ‘DPC’ widget
first, for each word, before using any other widgets) in order to ensure even data distribution. A total of sixty English words were used at the first session. The second session had four cycles as well and was similar to the first session; however, only ten words were given per cycle. A total of forty words were used in the second session. The words, which were used at the first and second sessions, were selected at a difficulty level above the ability level of the test users in order to ensure use of multiple widgets for each case (i.e., word). The first session was used for generating training data regarding the usage behaviors of the test users. After the first session, patterns were mined for each user. The data gathered at the second session was used as test data.

Table 3: Profiles of the test users.

<table>
<thead>
<tr>
<th>User</th>
<th>Occupation</th>
<th>Age group</th>
<th>#years using Internet</th>
<th>Frequency of Internet use</th>
<th>Familiarity with mashups</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>Teacher</td>
<td>20-25</td>
<td>7</td>
<td>daily</td>
<td>2</td>
</tr>
<tr>
<td>User 2</td>
<td>Engineer</td>
<td>25-30</td>
<td>12</td>
<td>daily</td>
<td>3</td>
</tr>
<tr>
<td>User 3</td>
<td>Teacher</td>
<td>30-35</td>
<td>14</td>
<td>daily</td>
<td>1</td>
</tr>
<tr>
<td>User 4</td>
<td>Student</td>
<td>20-25</td>
<td>8</td>
<td>daily</td>
<td>1</td>
</tr>
<tr>
<td>User 5</td>
<td>Student</td>
<td>20-25</td>
<td>7</td>
<td>daily</td>
<td>2</td>
</tr>
<tr>
<td>User 6</td>
<td>Student</td>
<td>15-20</td>
<td>4</td>
<td>daily</td>
<td>1</td>
</tr>
</tbody>
</table>

Recalling our approach, at the first stage, we select the mostly used two following actions for each action and perform a 2-class multi-label classification to mine the selection criteria at the second stage. We have analyzed the second stage in Section 5.2 and we want to evaluate the overall two-stage approach here. The overall approach itself can be considered as a multi-label classification problem as well where \( L \) is the set of widget actions excluding the triggering action itself which is subject of the classification. The \( L \) should not be confused with the label set used in 2-class multi-label classification at stage two which is simply the subset of \( L \) denoted here. In this respect, we can use evaluation metrics used for multi-label classification problems to evaluate the overall approach. In multi-label classification a result can be fully correct, partly correct, or fully incorrect [66]. For instance, say ‘engDict’ is found to be most frequent for the ‘mediatic’ widget along with an empty decision criterion (i.e., follows in any case) for a particular user. In the test session, after using the ‘mediatic’ widget, the test user might execute ‘engDict’ (fully correct), ‘dpc’ (fully incorrect), ‘engDict’ and ‘dpc’ (partly correct), ‘engDict’ and ‘flickr’ (partly correct) etc. The following metrics [64] were used for performance evaluation: (1) Hamming Loss [75], (2) accuracy, (3) precision, and (4) recall [76].

Let \( D \) be a multi-label test data set with \(|D|\) multi-label examples \((x_i, Y_i)\) where \(i = 1 \ldots |D|\) and \(Y_i\) subset of \(L\); let \(H\) be a multi-label classifier and \(Z_i = H(x_i)\) be the set of labels predicted by \(H\) for \(x_i\) [64]. Accordingly, Hamming Loss, accuracy, precision, and recall are defined in the following where \(\Delta\) stands for the symmetric difference of two sets, \(m\) is the total number of widget actions, and \(|L| = m-1\).

\[
\text{Hamming Loss}(H, D) = \frac{1}{|D|} \sum_i |Y_i \Delta Z_i|, \quad \text{Accuracy}(H, D) = \frac{1}{|D|} \sum_i \frac{|Y_i \cap Z_i|}{|L|},
\]

\[
\text{Precision}(H, D) = \frac{1}{|D|} \sum_i \frac{|Y_i \cap Z_i|}{|Y_i \cup Z_i|}, \quad \text{Recall}(H, D) = \frac{1}{|D|} \sum_i \frac{|Y_i \cap Z_i|}{|L|}.
\]
Precision($H, D$) = $\frac{1}{|D|} \sum_{i} \frac{|Y_i \cap Z_i|}{|Z_i|}$, and Recall($H, D$) = $\frac{1}{|D|} \sum_{i} \frac{|Y_i \cap Z_i|}{|Y_i|}$

The evaluation results are shown in Table 4. Although the approach is yet to be experimented with larger user groups and a higher number of widgets in different contexts, the experiments resulted in extraction of different patterns due to varying characteristics of the test users; therefore the results suggest that the mining approach is indeed promising.

**Table 4**: Analysis of the test results.

<table>
<thead>
<tr>
<th>Metric</th>
<th>User 1</th>
<th>User 2</th>
<th>User 3</th>
<th>User 4</th>
<th>User 5</th>
<th>User 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>HammLoss</td>
<td>0.20</td>
<td>0.15</td>
<td>0.15</td>
<td>0.13</td>
<td>0.13</td>
<td>0.19</td>
</tr>
<tr>
<td>Accuracy</td>
<td>0.72</td>
<td>0.77</td>
<td>0.80</td>
<td>0.80</td>
<td>0.78</td>
<td>0.72</td>
</tr>
<tr>
<td>Precision</td>
<td>0.77</td>
<td>0.86</td>
<td>0.90</td>
<td>0.80</td>
<td>0.81</td>
<td>0.80</td>
</tr>
<tr>
<td>Recall</td>
<td>0.83</td>
<td>0.91</td>
<td>0.81</td>
<td>0.80</td>
<td>0.91</td>
<td>0.76</td>
</tr>
</tbody>
</table>

The user experience with the automated system (the third session which is described in what follows) revealed that indeed the accuracy, precision, and recall can deviate from what we have observed. This is due to the possibility that users might feel uncomfortable or comfortable with an automated action whether it does or does not follow his intention. Therefore, in future experiments we plan to measure perceived accuracy, precision and recall along their observed values. In order to acquire perceived values, another controlled session with automation can be conducted. After each automation users can be asked to comment if they see the automated widgets appropriate and if not which widgets they were about to use.

The third session of the experiment was used as a basis for the end-user assessments and the usability analysis by means of an experience survey, and realized through a think-aloud manner. In Virzi [77], the author suggests that five-six test users are usually sufficient to find major usability problems (around %85); therefore, the aim of this preliminary user study was to detect major usability problems in early stages of our prototype and to provide input for the ongoing development process. In this third session, the generated rules were put in force in the platform and users were asked to use the system again with a different set of words. During the session, users were asked five Likert-scale questions: (1) ‘How useful do you find the mashup idea regardless of the prototype that you have just used?’, (2) ‘How successful do you find the system you have just used?’, (3) ‘How do you like the data mobility facility?’, (4) ‘How do you like the orchestration facility?’, and (5) ‘How do you like the dynamic widget relocation?’. Users were first asked to comment on the question including any recommendations and then to give a rank between 1 (very poor) and 5 (very good). The survey results are shown in Table 5.

Test users argued that the widget-based mashup idea quite useful and promising; most of the users immediately commented on possible uses and scenarios. Regarding our WIPLE implementation, expectedly, users found it yet to be improved. Users mainly demanded a more uniform interface and a higher degree of customization such as widget sizes, colors etc. Users mainly found the data mobility facility useful; two users commented that rather than using mouse clicks for copying
data, they would prefer a drag and drop facility supported with more visual cues (e.g., an animation). The widget orchestration facility was found very useful and users mostly reported on their past experiences in which they had needed such a facility. Widget relocation was mainly found useful; however, two users did not prefer sliding effect since they found it time consuming. One user commented that this could be customized (e.g., glow effect, sliding etc.) and user preferences on widget locations could also be learned and reflected while moving interacting widgets closer. The results suggest that the approach does not include any major usability problems; however, the platform interface has to be improved along with a customization support.

Table 5: End-user survey results.

<table>
<thead>
<tr>
<th>Concept</th>
<th>User 1</th>
<th>User 2</th>
<th>User 3</th>
<th>User 4</th>
<th>User 5</th>
<th>User 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Mashups</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>(2) WIPLE</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>(3) Data mobility</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>(4) Orchestration</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>(5) Relocation</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

During the first and second session, we observed that there might be various factors, other than the match between widgets, affecting the preferences of a user while combining functionalities of different widgets. For instance, two users, with comparatively lower language levels, usually omitted the ‘dpc’ widget, since they did find the example sentences provided by ‘DPC’ widget pretty long and complex. These two users were also indifferent to the word types and firstly preferred checking the meaning of the word and then possibly its image if it is a concrete noun. Another user with comparatively higher level language skills usually differentiated nouns and verbs and preferred checking the image of the word more often; however, the user found most of the retrieved images irrelevant and did not visit this widget as much as expected. These observations confirm that an orchestration approach, which is only based on syntactic or semantic match between widgets, is far from satisfying user preferences; however, the match between widgets can be quite useful to guide naive end-users to explore different possibilities.

6 Conclusion and Future Work

In this paper, we first have provided a broad overview of the mashup landscape with a holistic approach and set the links between widget-based UI mashups with orchestration, which we named mashups by orchestration, and web-based personal environments. We have discussed several prominent approaches for the realization of mashups by orchestration and opted for a system-driven approach where the system (i.e., the widget platform) learns user behavioral patterns from the user logs and automates the interplay between widgets accordingly. We have identified several generic challenges concerning most of the possible approaches and specific challenges mainly concerning a system-driven orchestration approach. We have described and addressed these changes in three main folds: (1) widget
interoperability, (2) platform and architecture, and (3) user behavior mining. We have investigated widget interoperability in two levels, namely application interoperability and data interoperability. We have introduced Functional Widget Interfaces (FWI), with which widgets disclose their functionality, for application interoperability and employed semantic web technologies, particularly ontologies and embedded semantics (e.g., eRDF, RDFa, microformats, microdata), for data interoperability. We have built an end-user data mobility facility on top of this interoperability infrastructure; this facility allows end-users to copy data from one widget to another. For this purpose, we have specified techniques for annotating HTML forms and matching user selected data with the HTML form elements. We have specified a reference platform and architecture, introduced a communication channel and message format, and extended W3C’s widget specification respectively. We have employed workflow mining and multi-label classification techniques within a two-stage approach for mining behavioral patterns in terms of their topology and routing criteria. We have employed Colored Petri nets for representation of the behavioral patterns. Finally, we have provided a preliminary evaluation of our approach in terms of its qualities, performance of the mining approach, and its usability with a prototype. The results suggested that our approach is promising.

The future work firstly includes maturation of the prototype and larger scale experiments for different application domains. Investigation of methods and techniques for dealing with concept drift problem and the exploration of applicability of an on-line pattern mining approach based on data stream mining are also of crucial. We believe that with the standardization of widget technologies, e.g., widgets, platforms (e.g., run-time system/environment, development frameworks etc.), and reference architectures, the widgetization of existing web applications (particularly dynamic approaches) will be of crucial importance due to the growing interest for personal environments and widget-based UI mashups in different domains (e.g., [78]). In this respect, we are interested in designing means for the automated widgetization of existing applications through harvesting application semantics from the interfaces of applications (i.e., with embedded semantics, cf. [79]). The interface annotation can be automated if the original application is developed with a Model Driven Development approach employing ontologies as a modeling paradigm (cf. [9, 80]). Finally, an interesting application of widget-based UI mashups is to enable end-users to program their personal and pervasive environments in which digital and physical entities are encapsulated by means of widgets (cf. [21, 27, 43]). In other words, the aim is to allow end-users to generate mashups by composition through demonstration to program their pervasive spaces (cf. [45, 81]). Our infrastructure and learning approach can be adapted for such purposes.

Acknowledgments. This article is based on research funded by the Industrial Research Fund (IOF) and conducted within the IOF Knowledge platform ‘Harnessing collective intelligence in order to make e-learning environments adaptive’ (IOF KP/07/006). Partially, it is also funded by the European Community's 7th Framework Programme (IST-FP7) under grant agreement no 231396 (ROLE project).
Mashups by Orchestration and Widget-based Personal Environments

References


Chapter 3
Conclusions and Future Research

This chapter concludes the work that was presented in this thesis, summarizes our main contributions, and indicates possible directions for future research.

The work presented in this thesis mainly focused on adaptive and (personal) pervasive environments from a user-centric point of view. It investigated how high level abstractions and semantics, varying from generic vocabularies and metadata approaches to ontologies, can be exploited for the creation of such environments and to enrich and augment the end-user experience. We reviewed the pervasive computing domain along its links with adaptive systems to grab the overall picture. Keeping in line with a user-centric approach based on abstractions and semantics, we reviewed the end-user aspects of Pervasive Computing in more detail along the recent state of the art intersecting KR, Software Engineering, Logic, and the Semantic Web. Our reviews suggest that the use of abstractions, particularly ontologies, is promising for the development and run-time adaptation of individual context-aware applications and for the aggregation and orchestration of these applications to form personal and pervasive environments. The design and development of individual adaptive and pervasive applications has been addressed at conceptual level. Regarding personal and pervasive environments, in this thesis, two practical studies were built on top of our conceptual work. The first study concerns the widgetization of traditional applications, in a broader perspective in terms of ubiquitous web navigation and access, by harvesting semantics embedded into the interfaces of web applications. The second study concerns the realization of a standard and open platform along an interoperability framework, based on the semantic web technologies, for the creation of widget-based personal environments. We also provided methods and techniques for two notable orchestration approaches, namely user-driven and system-driven orchestration, to enable interplay between widgets.

3.1 Contributions

This thesis makes a number of research contributions at conceptual and practical level.

The conceptual body of this thesis is based on two review articles and forms our main research perspective and the trajectory. The reviews also support us with main know-how and directions while realizing our practical contribution. We believe them
to be useful for other researches conducting or intending to conduct research on pervasive and adaptive computing systems. The conceptual perspective, that we derived, is based on end-user involvement and awareness at individual application level and collective level. The approach we employ is based on using high level abstractions, particularly ontologies, for the acquisition of domain knowledge and semantics at the first stage. Afterwards, on the one hand, the goal is to use the resulting ontology for run-time reasoning for the sake of dynamic adaptations, end-user awareness, intelligibility, self-expressiveness, and user control. On the other hand, the goal is to automatically generate and re-generate (i.e., requirement adaptability) the application code and other related software artifacts, by using the same ontology, through iteratively deriving more concrete sub-models from the source ontology. This approach also enables the share of application knowledge and semantics over the interfaces of the applications (e.g., HTML), since annotations are directly dependent on the source ontology. We built an approach that allows us to widgetize annotated applications effortlessly. This moved us from an individual application perspective to a perspective based on the collective operation of distributed applications. We provided a platform and interoperability framework, enhanced with ontologies and semantics, allowing end-users to manually and automatically blend the functionalities of the member applications, devices etc.

According the aforementioned approach and methodology our contributions can be summarized as follows:


1. We provided a review of the Pervasive Computing domain. This review allowed us to synthesize our research challenges and the aforementioned vision and approach at individual and collective level.


2. According to results of the first review, we provided a second review in which, at individual application level, we:
   a. identified the main problems and elements of software intelligence from development and end-user perspective;
   b. provided a conceptual approach where ontologies are used for automated development, run-time adaptation, and for meeting the end-user considerations (software intelligibility, situation awareness, end-user involvement etc.).
Conclusions and Future Research


3. We proposed an approach, by building on the ontology-driven approach presented in our second review, in which, at collective application level, we provided:
   a. an approach enabling ubiquitous access to applications which are semantically annotated. The annotation process is straightforward, if the target application is developed with respect to the ontology-driven approach presented in the second review;
   b. specifications for specifying, extracting, and presenting the embedded semantic information;
   c. a set of heuristics to enable end-user consumption of the extracted semantic information;
   d. a prototype to prove the feasibility of the proposed approach;


4. We proposed a platform, interoperability framework, and orchestration approach for the realization of personal and pervasive environments at collective level. More specifically, we provided:
   a. an interoperability framework based on a standardized communication channel, messaging format for event delivery and communication, functional widget interfaces for functional integration, and semantic annotations of content, widget interfaces, and events, for data mobility and enhanced semantic interoperability (with respect to the approach presented in first practical article);
   b. an open platform and a reference architecture for widget-based UI mashups with run-time and backend systems along specification of standard platform services and components;
   c. a method for end-user data mobility facility based on interoperability framework, for user-driven widget orchestration;
   d. an algorithmic approach, based on workflow mining techniques, for learning behavioral user patterns in order to realize automated widget orchestration;
   e. generic extensions to W3C’s widget specifications, with respect to the proposed interoperability framework, particularly in terms of communication infrastructure and access to platform services.

Regarding the evaluation of the practical body of our work, for the ubiquitous access/widgetization, we provided a prototype named SWC and conducted performance tests in order to prove that our approach is computationally feasible. We defined metrics such as expected and observed precision to evaluate the effectiveness of our approach in terms of information access, and finally conducted a usability test.
to validate usability aspects of our approach. Regarding the widget-based personal environments, we developed a prototype and implemented user-driven and system-driven orchestration approaches. We realized a personal learning environment, for language learning named WIPLE, and evaluated different orchestration approaches through a comparative analysis of their qualities. We evaluated the efficiency of our mining approach through computational analysis and user experiments. We conducted a user study to evaluate usability aspects of the proposed approach.

3.2 Discussion and Open Problems

We consider expressive abstractions, being completed with necessary tool support, as a key instrument that can close the gap between end-users and machines, in terms of end-user involvement, and between machines themselves, in terms of interoperability. This is why, through this thesis, we addressed end-user considerations on the ground of high level semantics and abstractions. In this respect, our first practical contribution brought end-users and semantics together and showed that end-users indeed can consume semantic data. Our second practical contribution demonstrated that high level abstraction and ontologies can be of use to monitor and automate user interaction with better precision and to enable users to command heterogeneous and distributed software agents.

**Developing User-Centric Pervasive Software.** The research domain clearly underestimates the role of end-users. This is mainly due to the strong machine-oriented perspective which envisions an intelligent digital world that can accommodate and satisfy the needs of human beings. End-user involvement, from a traditional perspective, is mostly considered as a development-time paradigm where end-users provide input during the development cycle. With the increasing software complexity, approaches, based on enabling end-users to program their own applications, did appear. However, approaches, which put the end-users directly into the role of programmers, even with advanced visual programming support, are mostly likely to fail. Learning from the end-users is a prominent approach; however, it is crucial to let users know what is really happening and to give a chance to interfere. In some cases, it might be simply enough to support the users with an appropriate amount of contextual information and let them to take actions accordingly. Regarding the development, till the time when software can “code itself”, design and development is the integral part of the software market. Increasing software complexity even hinders the small enterprises and causes considerable amount of loss of sources. The current software development practices and tools are the result of evolving abstractions. Ontologies can be considered as the next step which is indeed already taken with MDD through the use of less expressive abstractions, i.e., models. In this respect, knowledge engineers can be the fact of future’s software development.

**Ubiquitous Web Access.** The proposed approach is indeed a realization of abstract interfaces and allows interface semantics to be delivered through the interfaces themselves. Regarding the practicality of our approach, the most expensive processes are extraction and reasoning; however, experiment results, with comparatively high number of triples, revealed that our approach is feasible. The evaluation results suggested that the proposed approach decreases the network traffic as well as the
amount of information presented to the users without requiring significantly more processing time. More importantly, end-user experiments showed that the approach is promising to create a satisfactory navigation experience comparable to a normal navigation experience. The evaluation of our approach revealed more heuristics that can enhance the generated experience. The proposed approach and prototype only support annotation of non-interactional content elements at the moment; however, to some extent, we addressed the annotation of interactional elements (i.e., forms) in our following practical study (widget platform and interoperability framework). The proposed approach indicates that domain knowledge and semantics can be quite enhancing for the end-user experience; however, publishers usually do not opt for putting effort on complex ontology development process. Although our approach also supports the use of simpler vocabularies, a unified approach, in which applications are derived from an ontology automatically, can be convincing. This is because annotation process becomes automated and publishers can benefit from the automation of development as well. Secondly, the true integration of the Semantic Web with the current web technologies can motivate the use of semantic web technologies. In our approach, we proposed that web application servers should be able to deliver semantic information directly without needing any third party services or client-side extraction mechanisms.

**Widget-based Personal Environments.** The proposed approach, in contrast to design-driven approaches, opts for learning from the end-user and automating the collective behavior of the widgets accordingly. The qualitative evaluation of our approach, with respect to other orchestration approaches, showed that a system-driven approach and hybrid approach have certain advantages over others. A hybrid approach is expected to offer recommendations to the end-users when the system is not confident with the next action to be selected (i.e., probability of being next action is not high). We did not provide a hybrid approach yet; however, a hybrid approach can be built on top of our system-driven approach. Ontology-driven nature of our approach can facilitate identification of possible recommendations. The evaluation results, regarding the performance of the mining approach, showed that the approach is promising. At the moment, the approach is based on offline learning. We would like to work towards an online approach. We also require techniques for tackling with concept drift problem which happens when the distribution of instances underlying the mined patterns change over time. Usability analysis, conducted over a prototype for language learning, indicated that automated widget orchestration and widget-based UI mashups are promising for the construction of widget-based personal environments. More practical work is required to truly enable pervasiveness of the proposed approach in terms of coupling widgets and physical devices (width digital presence, preferably with web presence, cf. [105]), ensuring the accessibility of platform over devices with limited screen size (e.g., mobile devices), and enabling platform to support widget instances distributed over different devices (e.g., one widget runs on your TV and another widget runs on your mobile phone).

Many research problems remain open. Among others, at individual application level, approaches and methods need to be developed to realize end-user situation awareness, software intelligibility, and end-user control. At collective level, enabling inexperienced end-users to “program” their own environments, in a task oriented manner, is required. Note that, by programming we do not refer to the scenario where
end-users directly program (including scenarios with visual programming support), rather we refer to scenario where the specifics of tasks are demonstrated by the end-users.

## 3.3 Future Research

Our future work is mainly built upon the perspective constructed by our reviews and complementary to the conceptual and practical work presented in this thesis. We aim at investigating and developing frameworks for the following main objectives: (1) *end-user situation awareness and control* (cf. [11]), (2) *intelligibility and self-expressiveness* (cf. [53]), and (3) *environment programming* (cf. [106]).

The first two challenges are interlinked and address the practical aspects of end-user considerations, which we addressed in the conceptual part of this thesis, at individual application level. It also includes the realization of a unified development approach utilizing ontologies as development-time and run-time artifacts. We aim at exploring methods and techniques to use ontologies to communicate relevant contextual information to the end-user, to acquire user feedback (when system is the primary decision taker), to provide feedback to the end-users (when the end-user is primary decision taker), to explain reasoning behind adaptations, and to make behavior of applications more understandable by end-users without any explicit system effort. We have already initiated an interdisciplinary research track for this purpose (see [107]). E-learning has been selected as the application domain. Three research domains are combined, namely instructional science, methodology, and computer science. The goal is to develop an item-based language learning environment consisting of simple questions that can be combined with hints and feedback. An adaptive system is expected to be developed from a domain ontology for item-based learning environments. Ontology is also expected to be used for the dynamic adaptation of item sequencing mechanism (cf. [108]). Learners are expected to be supported with adaptive feedback, awareness of the execution context, and causal information regarding the adaptation logic.

Regarding the third challenge, we aim at empowering the end-users to “program” their environments including digital entities and physical entities having digital presence. We have already attempted towards this direction (see [109]). Our first attempt was based on facilities enabling end-users to connect inputs and outputs of different widget-like applications in a similar manner to Yahoo Pipes and Deri Pipes (i.e., through wires) (cf. [110]). We employed a metaphor similar to a movie maker application, including scenes and timelines, due to our observation that many naive users can successfully create short videos using such applications. With respect to our first end-user tests, approach and facilities proposed in our first attempt could not succeed to meet our expectations and failed for the naive end-users; it is a challenging task to realize end-user programming (we refer to visual programming). We, as already discussed, concluded that a programming approach is not truly appropriate for the end-users. Rather than asking users to describe and actually program a task, it might be better to ask them to demonstrate the task and so that system can learn from it. Therefore, our future attempts will be based on programming by demonstration idea (cf. [60, 111]). The very same platform that we have developed for the widget-
based personal environments, in this thesis, is quite appropriate for this goal. It is required to adapt and extend the employed pattern mining approach for programming by demonstration purpose.

3.4 Concluding Thoughts and Trends

We believe that computing will further enhance the quality of life, but this will not be only because of more ‘intelligent’ machines but also because of the fact that computing technologies will be more and more ubiquitous and will extend our physical (e.g., remote controls), sensory (e.g., digital sensors) and mental (e.g., automated analyses, simulations) abilities. In this respect, approaches merging human intelligence and machine processing power are important. On the one hand, it is beyond one’s capacity to enumerate every possible use case and to design tailored end-user experiences. On the other hand, smart technologies should not make people dumb and adapt activities and environment to this dumbness [112]. These necessitate approaches enabling end-users to design and manage their own experiences rather than purely design-driven approaches. We do not claim that adaptive experiences are not useful but simply are not a panacea for all user needs.

The digital ground for the end-user experience has been expanded drastically, since the notion of environment, for the end-users, has changed. The user environment is not based on the vicinity and physical connectedness anymore. A user environment encompasses any digital and physical entity, location etc., which the end-user is physically and/or digitally connected with and have capability to affect. The Web of Things (cf. [61]) vision is quite important in this respect which aims at using existing web technologies and standards (URI, HTTP, REST, HTML etc.) to access functionality of everyday devices connected to the Internet. Devices are expected to serve their functionality through web applications, coupled with their internal functionalities, published through embedded application servers or gateways. Several successful attempts have been done (e.g., [113]) for embedding web server functionality to the devices, yet more effort on standardization is required (e.g., interface, publishing etc.).

A personal and pervasive environment includes a variety of entities in varying types; therefore, the analysis and visualizations of interactions between these entities becomes important for supporting cognitive processes, reflection and awareness, analyzing user interactions, finding and explaining patterns and characteristics, providing visual feedback etc. (cf. [114]). The use of one-mode networks is quite common in the literature for such purposes. The most popular approach for exploring networked structures in such ecosystems is social network analysis (SNA) which focuses on the relationships (edges) among social entities i.e. humans (nodes) (cf. [115]). However, one-mode networks result in loss of information when analyzing complex interactions including more than one type of entities. K-mode networks provide more information; however, considering huge number of entities, visualizing k-mode networks becomes difficult. For this reason, k-mode networks are rarely used in the literature. Pattern based approaches are proposed to tackle with this problem (e.g., [114]). A pattern based approach allows zooming specific point of the visualization, hence facilitates the analysis of visualizations. Nevertheless, a
considerable amount of effort is required before k-mode networks can be effectively used (e.g., [116]).


List of Publications

International Journal Articles


International Conference and Workshop Papers


2. Visualization of Networked Collaboration in Digital Ecosystems through Two-mode Network Patterns. Felix Mödritscher, Wolfgang Taferner, Ahmet


Books and Book Chapters


National Conference Papers


Technical Reports


Biography

Ahmet Soylu was born in Elazig, Turkey on 26 November 1984. He received his BSc degree in Computer Science from the Işık University (with second rank), Istanbul, Turkey, in 2006, and his MSc degree in 2008 from the same university. He was a research assistant between 2006 and 2008 in IRDC (Informatics Research and Development Center) research group of Işık University. Since 2008, he is a PhD candidate and research assistant in the ITEC-IBBT (Interdisciplinary Research on Technology, Education and Communication) and CODeS (Combinatorial Optimization and Decision Support) research groups at the Department of Computer Science of the KU Leuven KULAK. He has been involved in several EU level research projects such as iCamp (IST FP6/STREP), LEFIS (Erasmus), and TREE (Socrates). His research interests include Pervasive Computing, Context-aware Computing, Adaptive Computing Systems, e-Learning, Human-machine Interaction, End-user Development, Meta-data and Semantics, Ontological Engineering, The Semantic Web, Formal Modeling, Knowledge Representation, Software Engineering, and Model Driven Development.
Try not to become a man of success but rather to become a man of value.

Albert Einstein