Formal modelling, knowledge representation and reasoning for design and development of user-centric pervasive software: a meta-review

Ahmet Soylu* and Patrick De Causmaecker

K.U. Leuven,
Department of Computer Science,
ITEC-IBBT, CODES,
Kortrijk, Belgium
E-mail: Ahmet.Soylu@kuleuven-kortrijk.be
E-mail: Patrick.DeCausmaecker@kuleuven-kortrijk.be
*Corresponding author

Davy Preuveneers and Yolande Berbers

K.U. Leuven,
Department of Computer Science,
Heverlee, Belgium
E-mail: Davy.Preuveneers@cs.kuleuven.be
E-mail: Yolande.Berbers@cs.kuleuven.be

Piet Desmet

K.U. Leuven,
Department of Linguistics,
ITEC-IBBT,
Kortrijk, Belgium
E-mail: Piet.Desmet@kuleuven-kortrijk.be

Abstract: 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 modelling 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 behavioural 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 realise a merged approach.

Keywords: MDD; model driven development; ontologies; logic; reasoning; pervasive computing; formal modelling; adaptive systems; KR; knowledge representation; software engineering; Petri nets; user-machine interaction; user control; software intelligibility.

Reference to this paper should be made as follows: Soylu, A., Preuveneers, D., Berbers, Y., De Causmaecker, P. and Desmet, P. (2011) 'Formal modelling, knowledge representation and reasoning for design and development of user-centric pervasive software: a meta-review', *Int. J. Metadata, Semantics and Ontologies*, Vol. 6, No. 2, pp.96-125.

Biographical notes: Ahmet Soylu is a PhD candidate in Computer Science at K.U. Leuven (Belgium) since 2008. He holds his BS (2006) and MSc (2008) Degrees in Computer Engineering and Science from Isik University (Istanbul, Turkey). He is presently a full time researcher in ITEC research group at K.U. Leuven (Kortrijk, Belgium). The aim of his PhD is to exploit metadata, ontologies and semantics to design and enhance (new) end-user experiences for pervasive computing environments.

Davy Preuveneers is a Postdoctoral Researcher at the DistriNet research group of the Department of Computer Science at the Katholieke Universiteit Leuven in Belgium. He received his PhD Degree in 2009, his MSc in artificial intelligence in 2003, and his MSc Degree in Computer Science in 2002 from the Katholieke Universiteit Leuven. His research interests include middleware and service oriented architectures for context-aware service interaction and application adaptation, and software engineering for embedded software in particular for mobile and ubiquitous computing.

Yolande Berbers is a Professor at the Department of Computer Science of the Katholieke Universiteit Leuven (Belgium), where she is a member of the research group DistriNet (Distributed Systems and Computer Networks). In general her research interests include software engineering for embedded and real-time systems, middleware infrastructure, ambient intelligence and ubiquitous computing, system support for distributed applications, distributed and parallel systems, distributed computing, multi-agent systems with emergent behaviour and computer architecture.

Patrick De Causmaecker is currently a Professor of Computer Science at the Subfaculty of Sciences at Katholieke Universiteit Leuven, Campus Kortrijk, Belgium. He is the head of the CODeS Research group in the Department of Computer Science and Coordinator of the ITEC research group, Kortrijk, Belgium. His current research interests include optimisation, planning and scheduling, distributed scheduling applications, agent technologies, and pervasive computing.

Piet Desmet is Full Professor of French and Applied Linguistics at the Faculty of Arts at K.U. Leuven and ITS Campus Kortrijk. He is co-directing ITEC research group. His current research interests include computer-assisted language learning, French language teaching, and educational technology.

1 Introduction

Although the emergence of Pervasive Computing goes back to the early 1990s (Weiser, 1991), 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, miniaturisation, energy consumption, materials, sensors, etc. (see Bick and Kummer, 2008; Cook and Das, 2007). However, Pervasive Computing (Satyanarayanan, 2001; Weiser, 1991) 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 customise 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 (Bettini et al., 2010; Dey, 2001, 2009a; Preuveneers, 2010; Schilit et al., 1994). Context-awareness is defined as the capability of being able to perceive (through physical, virtual, etc., sensors (Indulska and Sutton, 2003)) the dynamic computing context and to respond collectively, proactively (Coutaz et al., 2005), properly, and seamlessly to better serve users

without creating any distraction at the user’s side (Soylu et al., 2009). In this respect, context is any information (e.g., device screen size, etc.) characterising the situation (e.g., characteristics, requirements, capabilities, etc.) of any entity (e.g., applications, users, devices, etc.) (Dey, 2001; 2009a).

The emergence of complex systems, particularly with the rise of the Pervasive Computing era (Weiser, 1991) and the impact of Adaptive Systems (Brusilovsky et al., 2007), 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 (Soylu et al., 2009) 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., personalisation), or in a broader sense, customisation 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 behaviour spaces (i.e., a set of available context elements and a set of possible adaptive behaviours respectively) (Soylu et al., 2009). 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 behaviour spaces of software substantially and, consequently, hardens the management of hard-coded mappings between the context and behaviour spaces and implicit reasoning logic,

as well as validation and verification of structural and behavioural 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 realised as of yet, due to the ever-growing context and behaviour 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 behaviour. User involvement necessitates software to ensure intelligibility (Dey, 2009b) of its behaviour/decisions which is possible through user situation awareness (Spiekermann, 2008), which is realised 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' behaviour by automatically enforcing university regulations on the administrative and academic operations realised 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 realised 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 behaviour 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 complains 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 behaviour 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 behaviour/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 behavioural (in terms of models), and use of semi/fully automatic mechanisms for transformation from application knowledge to application code. This allows the realisation 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 behavioural 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 formalised 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 behaviour 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 Katasonov and Palviainen (2010), Knublauch (2004), Parreiras and Staab (2010), Soyulu and De Causmaecker (2009) and Valiente (2010). On the one hand, in the software engineering domain, automated development of complex software products from higher abstract models has gained great momentum (Ayed et al., 2007), 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 utilised as run-time and development time software artefacts 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 (Dey, 2009b), self-expressiveness, user involvement, and user situation awareness
- 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 behavioural 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 behaviour. 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 realise the aforementioned merged approach.

The rest of the paper is structured as follows. In Section 2, we present our motivation in four respective sub-sections; 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 emphasise 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 realise 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 modelling, KR and reasoning can offer to counter the elaborated issues while giving glimpses of an approach merging KR and software modelling instruments and fundamentals.

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 behaviour. 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 behaviour of the application; the latter focuses on models that formalise the application semantics and the operational context. A model-driven development approach is envisioned to blend software models like UML diagrams and MDD design artefacts, with knowledge models like ontologies and rules to generate context-aware adaptive applications. This model-based design process allows software developers to embed a formalised representation of the characteristics and behaviour 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.

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 behaviour through anticipation and adaptation, the end-user should also understand the application’s capabilities and its adaptation behaviour. 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 behaviour 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 behaviour 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 behaviour of the system.

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

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

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 (Alfons, 2007; Murch, 2004; Preuveneers and Berbers, 2008a). An increasing demand for large and complex software systems that are able to adapt to dynamically changing computing settings has appeared (Geihs et al., 2009) with

- the rise of mobile devices and computing and, later, the emergence of Pervasive Computing (Bick and Kummer, 2008; Krumm, 2009; Satyanarayanan, 2001; Weiser, 1991)

- the increasing need for adaptive, particularly user-adaptive (i.e., personalised) software systems (Brusilovsky et al., 2007; Salehie and Tahvildari, 2009; Schneider-Hufschmidt et al., 1993).

The dynamic and heterogeneous nature of pervasive computing settings (Preuveneers and Berbers, 2008b) 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 personalised computer-based instruction is superior to the traditional approaches (Kadiyala and Crynes, 1998). In other words, the pervasive computing vision manifests an unobtrusive, anytime and anywhere (Bick and Kummer, 2008) user experience which requires expansion of the personalisation 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 (Dey, 2001; 2009a) simply represents formalised snapshots 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 (Alfons, 2007). In contrast, the new perspective (see Figure 3) considers computing as a continuous process of recognising user's goals, needs, and activities and mapping them adaptively onto the population of available resources that responds to the current context (Garlan et al., 2002) (i.e., context-awareness (Baldauf, Dustdar and Rosenberg, 2007; Bettini et al., 2010; Krumm, 2009; Perttunen et al., 2009; Preuveneers, 2010; Schilit et al., 1994; Soylu et al., 2009)).

Figure 3 The new computing perspective: context and computing setting

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’ behaviour (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 (Soyly et al., 2009, 2010a). Contextual information is mainly collected through physical sensors acquiring real world data, and through virtual sensors acquiring transactional data through the application logs (Cook and Das, 2007; Indulska and Sutton, 2003). This type of contextual information (i.e., acquired through sensors) is called low-level context (Du and Wang, 2008) 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 (Padovitz et al., 2008), 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) (Du and Wang, 2008). The resulting contextual information is called high-level context. Contextual information is often imperfect (Bettini et al., 2010; Dey, 2009b; Perttunen et al., 2009; Soyly et al., 2009; Strang and Linnhoff-Popien, 2004) 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 Baldauf et al. (2007), Bettini et al. (2010), Du and Wang (2008), Padovitz et al. (2008), Preuveneers (2010), Soyly et al. (2009), Strang and Linnhoff-Popien (2004) for further analytical and conceptual information on contextual inference and reasoning. The final phase is usually the definition of adaptive behaviours and their mappings to the identified contexts. Context-behaviour mapping follows a similar inference procedure, i.e., mapping a set of related context elements – probably each having a different weight – to particular behaviour(s). One can further abstract such sets of context elements in terms of ‘situations’ (Dey, 2009b; Padovitz et al., 2008) 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 (Banavar et al., 2000) 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 behaviour spaces can be encoded in the application itself and dynamic adaptations can be handled through hard-coded behaviour-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 behaviours, as well as mappings between ever-growing context and behaviour spaces. Accordingly, it is necessary to maintain an extensible and formalised conceptualisation 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 (Lieberman et al., 2006), might also enable end-users to program and control their own environments in the future (i.e., through adjusting the application behaviour or introducing new context-behaviour mappings). This is what we call environment programming (Helal, 2010; Soyly and De Causmaecker, 2009; Soyly et al., 2010b) (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: customisation with explicit user input) without being identified at design time. In Wild et al. (2008), it is criticised 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.) (Geijs et al., 2009) 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 formalised form (Banavar et al., 2000) closer to the human-level language, is important to avoid repetitive coding of the application by automatically transforming application knowledge to software artefacts, 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 behavioural) 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 build machines that can achieve human levels of intelligence (Dreyfus, 1993; Dreyfus and Dreyfus, 2000; Kasabov, 2008; McCarthy, 2007; Winograd and Flores, 1987; Zadeh, 2008) 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 (Zadeh, 2008) (opposed to the more optimistic views (McCarthy, 2007)). There are years of research ahead of us, most probably only with limited achievement in terms of real intelligence (Kasabov, 2008). 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 (Kasabov, 2008), 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 Zadeh (2008).

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 Tribus and Fitts (1968), 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 (Zadeh, 2008). Computational systems are good at gathering and aggregating data and humans are good at recognising contexts and determining (e.g., to reason) on what is appropriate (Erickson, 2002).

Today, ‘intelligent’ systems and applications are ‘intelligent’ only to the extent of completeness of the real-world contexts modelled by the developers. Their ‘intelligence’ is strictly based on strong logical assumptions or computational and algorithmic procedures prepared by the developers (Maher et al., 2006). Hence, we prefer to call such ‘intelligence’ machine encoded human intelligence/simulated intelligence (i.e., weak AI or pseudo-intelligence (Constantine, 2006; Searle, 1980), and we prefer AI alone to refer to strong AI or human level of AI (McCarthy, 2007; Zadeh, 2008)). This is because it is only a limited reflection of the human intelligence, that 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/artefact 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 (Cook and Das, 2007)). 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 (Dey, 2009b), 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 behavioural 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 (McCarthy, 2007) 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 standardised 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 formalisation 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 realised through the 'attachedTo' property (which is symmetric, i.e., if (x, y) holds then (y, x) also holds). In an implementation without such formalised conceptualisation, 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.

Although the approach is of use for manageability and for application development, imperfectness of the contextual information (Cook and Das, 2007; Dey, 2009b; Strang and Linnhoff-Popien, 2004) 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 formalised conceptualisations can be used to some extent for consistency checking and verification and validation of structural and behavioural

properties of the software. Furthermore, various AI techniques can be applied to alleviate imperfectness (Anagnostopoulos and Hadjiefthymiades, 2009; Bardram, 2005; Binh An et al., 2005; Haghighi et al., 2008; Ranganathan et al., 2004; Zhongli and Yun, 2004); 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 (Anderson, 2005; Georges, 2004; Helmreich, 2000).

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.) and (b) without semantics, developers have to enumerate every possible concept while constructing the logic rules

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 (Constantine, 2006). 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* (Hassenzahl and Tractinsky, 2006): It refers to the ability of a system to attract and hold the attention of the users (Chapman et al., 1999).

In O'Brien and Toms (2008), the authors remark that successful technologies are not just usable; they engage the users, given the increased emphasis on user experience (Blythe et al., 2003).

- *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 (Constantine, 2006). The absence of trust introduces inefficiency, demanding added vigilance, encouraging protective and unproductive actions, and complicating interaction (Constantine, 2006).
- *User acceptance* (Spiekermann, 2008): 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 (Brown and Cairns, 2004; Constantine, 2006; O'Brien and Toms, 2008; Spiekermann, 2008). User involvement can be considered both as a:
 - Development time issue (Begier, 2010) 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 behaviour 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 Spiekermann (2008), the author points out that perceived control is the conviction that one can determine the sequence and consequences of a specific event or experience (Zimbardo and Gerrig, 1996). Control over a system might be totally held by the system or the user. Alternatively, it can be shared (Corbalan et al., 2008) 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 (Das and Roy, 2008; Dey and Mankoff, 2005; Mankoff et al., 2000; Roy et al., 2009) (i.e., user feedback to machine) or adaptive user guidance/advisement (Bell and Kozlowski, 2002; Pervez and Ryu, 2008) (i.e., machine feedback to user). Considering adaptive advisement, we argue that adaptive application behaviours 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 (Korpipää et al., 2004). Considering user meditation, as previously mentioned, adaptive behaviours are realised by means of predefined behaviours mapped to possible contexts of the setting and use. However, imperfectness of context information decreases the reliability of adaptive behaviours. 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 behaviours, while the ideal case is placing fewer demands on the user's attention (Hagras, 2007; Henriksen et al., 2002). 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 Spiekermann (2008), 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?" (Woods, 1996). 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 (Spiekermann, 2008). It is further argued in (Spiekermann, 2008) that the reason for these questions and resistance is lack of situation awareness (Endsley, 1996). 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 (Peng and Silver, 2005), while delegating too much control over machines causes lack of situation awareness (Spiekermann, 2008). 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 (Dey, 2009b; Henriksen and Indulska, 2006; Jameson and Schwarzkopf, 2002). In this context, the assistance systems have an important place, in terms of enhancing user perception, interpretation of data, feedback, and motivation (Adell et al., 2008; Wandke, 2005). The solution lies in providing the right level of balance between automatic system decisions and user involvement. Apparently, such optimisation 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 behaviour 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 modelling, 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 (Henderson-Sellers, 2011; Pidd, 2000). 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 (Meservy and Fenstermacher, 2005; Singh and Sood, 2009). It eventually approaches human language through the use of representation formalisms with a higher degree of abstraction (Mellor et al., 2003) – any program code is simply another, albeit low level, abstraction (Meservy and Fenstermacher, 2005) – thereby enabling programming less bound to underlying low-level implementation technology (Booch et al., 2004; Selic, 2003). A basic development process in MDD starts with the identification of target platforms. Afterwards, it is important to select appropriate meta-models (Fritzsche et al., 2009), which provide basic primitives (i.e., constructs) for developing models belonging to a specific subject area (i.e., any realisation of source meta-models or target meta-models), and an appropriate language for formalisation. 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) (Booch et al., 2004), 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 (Schmidt, 2006). Prominently, UML (OMG, 2009b) as a modelling formalism is in the core of MDA. MDA utilises a meta meta-model which allows construction of different meta-models belonging to different subject areas. The MDA process consists of five main stages (Asadi and Ramsin, 2008; Meservy and Fenstermacher, 2005; Singh and Sood, 2009):

- 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 (Assmann et al., 2006a).

The benefits of MDD can be discussed under two main and interrelated categories: abstraction and automation. We first consider abstraction (Selic, 2003). A model has multiple views, some of which are revealed (Mellor et al., 2002). 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 (Booch et al., 2004; Mellor and Balcer, 2002). One of the goals of MDD is to enable sensitivity to inevitable changes that affect a software system (Atkinson and Kuhne, 2002; Gitzel et al., 2007). In Atkinson and Kuhne (2003), four fundamental forms of change are identified for a software: personnel (i.e., developer), requirements, development platforms, deployment platforms. In Mellor et al. (2003), 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 in 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 (Selic, 2003)). 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 (Gitzel et al., 2007). When considering automation, portability (Asadi and Ramsin, 2008) 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 (Singh and Sood, 2009). This provides a faster and cheaper migration (Meservy and Fenstermacher, 2005). In Gitzel et al. (2007), it is pointed out that code generation results in high reusability and increased productivity, since repetitive coding for the application is not required (Meservy and Fenstermacher, 2005). MDD increases the quality of the software. Firstly, errors are reduced by using automated tools for transforming models to application code (Booch et al., 2004). Besides, it is possible to verify consistency of the models through formalised models (Selic, 2003). 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 (Tetlow et al., 2006). 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 (Studer et al., 1998) define ontology as a formal and explicit specification of a shared conceptualisation where a conceptualisation 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) (Hofweber, 2004). An ontology refers to an engineering artefact, constituted by a specific vocabulary and set of assumptions (based on FOL) regarding the intended meaning of the vocabulary's words (Guarino, 1998). 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 (Gomez-Perez et al., 2003; Noy and McGuinness, 2001), and in the simplest case, an ontology describes a hierarchy of concepts in subsumption relationships (Guarino, 1998). A typical heavyweight ontology requires suitable axioms in order to express more sophisticated relationships between concepts and constrain their intended interpretation (Guarino, 1998), 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 categorised as follows (Gomez-Perez et al., 2003; Soylu et al., 2009):

- 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 (Gomez-Perez et al., 2003). 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 (Baader et al., 2003). 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 summarised as follows: reduced ambiguity, knowledge share, interoperability, re-usability, knowledge acquisition, communication between human-human and human-machine, inference and reasoning and natural authoring (Besnard et al., 2008; Gruninger and Lee, 2002; Ruiz and Hilera, 2006; Uschold and Gruninger, 1996; Uschold and Jasper, 1999).

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 (Gomez-Perez et al., 2003), and there is a trade-off between expressiveness and reasoning power (Levesque and Brachman, 1985). 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) (Eiter et al., 2008b).

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 (Horrocks, 2002). DL languages belong to a family of KR formalisms based on FOL (Esposito, 2008). The reasoning tasks supported in DL are subsumption and realisation. Subsumption determines sub-concept/super-concept relationships of concepts occurring in a TBox, where realisation computes whether a given individual necessarily belongs to a particular concept (Lin et al., 2005). F-logic (Kifer et al., 1995) 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 (Kifer, 2005; Motik et al., 2006). 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 (Hitzler et al., 2005; Kifer, 2005). 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 Boley et al. (2007), rules are categorised 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) (Berstel et al., 2007) describe the reactive behaviour 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 Boley et al. (2007) for more information. Rules, as a logic paradigm, are quite important, as they provide the capability of explaining why a particular decision is reached (Besnard et al., 2008; Diouf et al., 2007; Lehmann and Gangemi, 2007). 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 formalising engineering activities (Sicilia et al., 2009) and artefacts. Our focus is on software as an engineering artefact. There has been a considerable debate on the formal/informal form of software specifications (Colburn, 2000) gathering structural knowledge (about the components which comprise the design object and their relations), behavioural knowledge (about the behaviour 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 behaviours and goals of the artefact itself) (Colombo et al., 2007).

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 modelling formalism in MDD (Knublauch, 2004; Ruiz and Hilera, 2006; Tetlow et al., 2006) or employing MDA modelling instruments –

particularly UML, the UML meta-model and OCL – as a representation formalism to develop ontologies (Achilleos et al., 2010; Djuric et al., 2005; Gomez-Perez et al., 2003; Henricksen et al., 2002; Pan et al., 2006; Wang and Chan, 2001). 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 (OMG, 2006) (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 (Diouf et al., 2007; Noguera et al., 2010; Parreiras and Staab, 2010; Rodriguez et al., 2010). 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 (Pahl, 2007). In Daconta et al. (2003), 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 Knublauch (2004), by giving valuable insights, the author points out potential benefits of using domain models not only for code generation but also as executable software artefacts 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 organisations 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 visualising OWL and RDF KR constructs. In Guarino (1998), 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”. Ruiz and Hilera (2006) elaborates on the use of ontologies in Software engineering from various points, by covering the temporal

perspective suggested by Guarino (1998) as a core. The authors point out that, for development time, a set of reusable ontologies can possibly be organised 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. Katasonov and Palviainen (2010), Knublauch (2004), Soyulu and De Causmaecker (2009) and Valiente (2010) favour 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 (Knublauch, 2004). However, realisation 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 (Valla, 2010), 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 realised.

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 behaviour of the system, current OWL-based approaches do not (Noguera et al., 2010; Parreiras and Staab, 2010; Rodriguez et al., 2010; Silva Parreiras et al., 2007). The ability to model dynamic behaviour of a system is crucial for the automated development of pervasive and adaptive systems and applications, since behavioural models (including constraints imposed) are central to the adaptation process. UML’s ability to specify dynamic behaviour leads researchers to investigate means of combining power of the ontologies and UML.

In Parreiras and Staab (2010) 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 behavioural 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 Rodriguez et al. (2010) the authors present how processes modelled with SPEM (the Software & Systems Process Engineering meta-model of OMG (OMG, 2008) 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 behaviour 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 behaviour 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 behaviour 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 behavioural properties of the model. One response to this consideration is the integration of Petri nets (Murata, 1989), particularly high level Coloured Petri Nets (CPN) (Jensen and Kristensen, 2009), to the MDD process (Gasevic and Devedzic, 2006; Noguera et al., 2010). Petri nets are a graphical and mathematical modelling tool providing the ability to model, simulate and execute behavioural models; they are a sound mathematical model allowing analysis (e.g., performance), validation, and verification of behavioural 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 behaviour. 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 Gasevic and Devedzic (2006), 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 Noguera et al. (2010), the authors introduce a methodological framework called AMENITIES employing UML, OWL, and CPN to advance modelling and analysis of collaborative systems. The methodology is based on a collaborative process domain ontology allowing representation of processes (i.e., behavioural aspects) and relevant entities along with their relationships (i.e., structural aspects). Collaborative processes are modelled 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 behavioural constructs. The mapping from UML to OWL results in the formalisation of process models, and hence, enables ontological reasoning and validation. A mapping from ontological entities involved in description of behavioural aspects of the process model to the entities in CPN meta-model is also defined to exploit advanced behavioural 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 summarised 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 standardised 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 customised 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 modelling paradigms with respect to three prominent properties

	OWL	UML	Petri net	UML + OWL	UML + Petri net	OWL + Petri net
Extensible visual constructs	~	✓	~	✓	✓	~
Reasoning and semantic validation	✓	x	x	✓	x	✓
Behavioural analysis and validation	x	x	✓	x	✓	✓

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 modelling 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 behavioural analysis, validation and verification. This approach requires defining mapping schemas, and realising 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. A logic layer is required, since it is not possible to realise 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 behavioural 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.

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

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 conceptualised and formalised 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 (Assmann et al., 2006a; Fonseca, 2007). The overall approach is based on the understanding that ontologies are boarder than models in terms of semantics and the reality they describe (Fonseca and Martin, 2007), 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 (Gonzalez-Perez and Sellers, 2007; Henderson-Sellers, 2011). Ontologies are primarily used to describe domains (Sicilia et al., 2009) while models are used to prescribe systems (Assmann et al., 2006a). 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 (Henderson-Sellers, 2011)) from a philosophical perspective, since at this stage we are more interested in

practical issues we refer interested readers to Assmann et al. (2006a), Devedzic (2002), Gonzalez-Perez and Sellers (2007) and Henderson-Sellers (2011).

Figure 7 An objective-based merger of model-driven and ontology-driven approaches

Regarding a possible methodology towards integrating ontologies and MDD based on the third approach, the following has been proposed in Soylyu and De Causmaecker (2009) and Soylyu et al. (2009), 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 (Singh and Sood, 2009):

- 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 artefacts.

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 artefact, so that the transformation of the given ontology to the specific software artefact can be realised.

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 (Baldauf et al., 2007; Chen et al., 2004; Economides, 2009; Henze et al., 2004; Nicklas et al., 2008; Perttunen et al., 2009). A recent approach (Serral et al., 2010) 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 modelling pervasive computing systems on which automated code generation is based. The behavioural aspects of the system are modelled through state transitions diagrams (non-executable). An ontology for PervML is manually developed, and exchange/transformation between the PervML model and ontology is realised. 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 behavioural system properties due to insufficient formality of the modelling paradigm used.

Figure 8 A possible methodology merging MDD and ontologies

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 (Tetlow et al., 2006)
- 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 (Ruiz et al., 2004; Uschold and Jasper, 1999))
- a smooth requirements elicitation and modelling phase (Girardi and Faria, 2003), 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 (Besnard et al., 2008; Lehmann and Gangemi, 2007)
 - 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 (Ranganathan et al., 2003)
- alleviation of imperfectness through consistency checking of the context
- interoperability at semantic and syntactic levels (Ruiz and Hilera, 2006).

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 Ruiz and Hilera (2006), the authors point out that a typical ontology-driven information system consist 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 Chaari et al. (2006), Perttunen et al. (2009), Soylu et al. (2009) – 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 (Gu et al., 2005; Soylu et al., 2009) 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 artefacts 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 (Kifer et al., 1995), under the logic programming paradigm, and OWL (Bechhofer et al., 2004), 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 (Kifer, 2005), while OWL, from the semantic web (Shadbolt et al., 2006) family, targets ontologies for the web. F-logic has already been used successfully for ontology modelling, software engineering, agent-based systems, etc. (Kifer, 2005). Although F-logic is based on object-oriented primitives (Motik et al., 2006), and has strong links with logic programming, and availability of mature development environments, notably Ontobroker (already commercial), TIPPLE, 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 (Gomez-Perez et al., 2003) 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 (Soylu et al., 2010a).

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

(Eiter et al., 2008b; Horrocks et al., 2005; Motik and Rosati, 2010). 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 artefacts (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 (Eiter et al., 2008b). 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) (Bechhofer et al., 2004). OWL Full OWL provides maximum expressivity, but computational guarantees do not exist, since it has non-standard semantics (Motik et al., 2006). In particular, decidability is an important criteria, since complete and undecidable algorithms can get stuck in infinite loops (Hitzler et al., 2005). 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 (Krotzsch et al., 2006). 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 (Hatala et al., 2005; Perttunen et al., 2009). 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 (Boley et al., 2007). This is the central task in the current research (Antonioni et al., 2005; Assmann et al., 2006b; Eiter et al., 2008b; Perttunen et al., 2009). OWL DL is based on DLs, with an RDF syntax (Horrocks et al., 2003), which can be considered as a decidable fragment of FOL (Hitzler et al., 2005) (i.e., a DL knowledge can be equally translated into FOL (Motik et al., 2006)). FOL follows an OWA and employs monotonic reasoning, while logic programming follows the CWA and allows non-monotonic reasoning (Esposito, 2008; Gomez-Perez et al., 2003; Perttunen et al., 2009). Several reasons can be listed for integrating logic programming and OWL – extended from (Motik et al., 2006).

- *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 (Grosz et al., 2003; Motik et al., 2005). 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 (Esposito, 2008).

- *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 (Gomez-Perez et al., 2003). However, in practice higher arity predicates are encountered, such as ‘connect’ property, which is ternary, stating that a road connects two cities (Gomez-Perez et al., 2003; Motik et al., 2006)
- *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 Motik et al. (2006) 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 (Gomez-Perez et al., 2003). However, adaptive and pervasive systems require non-monotonic reasoning, since the dynamic nature of the context requires retraction and negation of the existing facts (Perttunen et al., 2009).
- *Integrity constraints*: Integrity constraints, which are non-monotonic tasks, cannot be realised in OWL (Motik et al., 2006; Reiter, 1992) 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 (Horrocks and Patel-Schneider, 2004) which combines OWL and Rule Markup Language (RuleML) (Boley et al., 2001) (i.e., an XML based mark-up language for rules (Eiter et al., 2008b)), however, it does not support non-monotonic reasoning and it is undecidable (Boley et al., 2007; Esposito, 2008). There are several reasoners supporting SWRL such as KAON2 and Pellet (for known decidable fragments – i.e., DL safe – of SWRL (Motik et al., 2005)), RacerPro (uses a SWRL like syntax and supports closed world semantics) (Eiter et al., 2008a). It is worthwhile mentioning the Jena 2 semantic web framework (Hewlett-Packard Development Company, 2011) 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 realises 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 Dao-Tran et al. (2009), Drabent (2010), Eiter et al. (2009), Eiter et al. (2008a, 2008b), Esposito (2008), Hitzler et al. (2005), Horrocks et al. (2005), Krotzsch et al. (2006), Motik and Rosati (2010) and Motik et al. (2005) for the current research towards improving OWL with the expressiveness of logic programming for interested readers. Similar to the standardisation of the ontology layer of the semantic web through OWL, rule layer also needs to be standardised, to enable use of ontologies and rules for innovative applications (Hatala et al., 2005) and exchanging rule based knowledge (Boley et al., 2007). W3C already initiated a working group for developing a standard exchange format, namely Rule Interchange Format (RIF) (Boley et al., 2009), for the rules. In Boley et al. (2007) the authors remark that the development of RIF includes two phases, where the first phase includes realisation of stable backbone and the second phase includes extensions with first order rules, logic programming rules, production rules, etc. We refer to Boley et al. (2009), Boley and Kifer (2010) and Boley et al. (2007) for further details, syntax, and semantics.

4.2 *The semantic web: ontologies to software artefacts*

Existing work in the literature for transforming ontologies to software artefacts, including application code, is of great use for the proposed approach in terms of tool support. In Eberhart (2002), the author introduces two cross compilers, namely, OntoJava and OntoSQL to realise 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 Goldman (2003) 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 Kalyanpur et al. (2004), the authors focus on OWL, which is a more expressive DL than RDFS, by building on the existing work of Eberhart (2002), 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 minimising 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 realised 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.) realised through constraint checker

classes registered as listeners on the properties; and property associations (e.g., including multiple domains and ranges) realised through Java interfaces and listeners. In Völkel (2006) 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
- realisation of multiple inheritance in a type-safe manner
- realisation 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 Völkel (2006), the earlier work of Eberhart (2002) lacks some basic features, such as multiple inheritance and properties with multiple domains and in Kalyanpur et al. (2004) the OWL type system is only supported through raised exceptions. Considering the relational databases, in Gali et al. (2004) and Astrova et al. (2007), 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 object-relational database management systems (Astrova et al., 2007). In Gali et al. (2004), 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 Vysniauskas and Nemuraite (2006), the authors describe a set of algorithms, based on the work of Gali et al. (2004), 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 Astrova et al. (2007), the authors refer to the related literature for OWL to the relational schema mapping (e.g., Astrova and Kalja (2007), Gali et al. (2004), Vysniauskas and Nemuraite (2006)) and claim that they suffer from one or more of the following problems:

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

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 Vysniauskas et al. (2010), 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 Athanasiadis et al. (2007) 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:
 - a 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 artefacts to ontologies (e.g., relational databases to ontologies (Astrova, 2009)). Although this is not the focus of this paper, such efforts are helpful to reveal the semantic gap between the software-related modelling 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 artefacts 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 realise 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 (Bezivin et al., 2008). In Sicilia et al. (2006), the authors point out that organisations managing their processes with ontology-enabled tools and methods would benefit from a flexible infrastructure prepared for inference and partial automation of processes. Pisanelli et al. (2002) 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 (Perttunen et al., 2009). 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 Brockmans et al. (2004), Mehroolhassani and Elci (2009). 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 standardised 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 visualising 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 (OMG, 2009a) 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 modelling formalism and languages provide extensible visual support and are easier to use. Although there are some distinctions between modelling 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 modelling 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 criteria affecting decisions in favour 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 artefact(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 artefacts for the reasoning purposes.

Limited research and tool support exists for automated transformations of OWL ontologies to the application artefacts (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 uses 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 artefacts (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 artefact (e.g., ontology) and the final artefact (i.e., the software artefact). 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 artefacts (e.g., Eberhart, 2002; Vasilecas and Bugaite, 2007; Vasilecas et al., 2009) 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) (Kazakov

et al., 2007). 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 (Fonseca and Martin, 2007).

6 Conclusions 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
- 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 realise 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 (Soylyu et al., 2010c)) to provide a personalised, any-time and any-where learning experience.

Acknowledgements

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 K.U. Leuven.

References

- Achilleos, A., Kun, Y. and Georgalas, N. (2010) 'Context modelling and a context-aware framework for pervasive service creation: a model-driven approach', *Pervasive and Mobile Computing*, Vol. 6, No. 2, pp.281–296.
- Adell, E., Varhelyi, A., Alonso, M. and Plaza, J. (2008) 'Developing human-machine interaction components for a driver assistance system for safe speed and safe distance', *Iet Intelligent Transport Systems*, Vol. 2, No. 1, pp.1–14.
- Alfons, J.S. (Ed.) (2007) 'Intelligent computing everywhere', *Intelligent Computing Everywhere*, Springer, London, pp.3–23.
- Anagnostopoulos, C. and Hadjiefthymiades, S. (2009) 'Advanced inference in situation-aware computing', *IEEE Transactions on Systems Man and Cybernetics Part a-Systems and Humans*, Vol. 39, No. 5, pp.1108–1115.
- Anderson, M.L. (2005) 'Why is AI so scary?', *Artificial Intelligence*, Vol. 169, No. 2, pp.201–208.
- Antoniou, G., Damasio, C.V., Grosz, B., Horrocks, I., Kifer, M., Maluszynski, J. and Patel-Schneider, P.F. (2005) *Combining Rules and Ontologies: A Survey*, Technical Report IST506779/Linköping/13-D3/D/PU/a1, Linköping University.
- Asadi, M. and Ramsin, R. (2008) 'MDA-based methodologies: an analytical survey', *4th European Conference, ECMDA-FA 2008. Model Driven Architecture – Foundations and Applications*, 9–13 June, Berlin, Germany, pp.419–431.
- Assmann, U., Henriksson, J. and Maluszynski, J. (2006b) 'Combining safe rules and ontologies by interfacing of reasoners', *4th International Workshop, PPSWR 2006. Principles and Practice of Semantic Web Reasoning*, 10–11 June, Budva, Montenegro.
- Assmann, U., Zschaler, S. and Wagner, G. (2006a) 'Ontologies, meta-models, and the model-driven paradigm', in Calero, C., Ruiz, F. and Piattini, M. (Eds.): *Ontologies for Software Engineering and Technologies*, Springer, Berlin, pp.175–196.
- Astrova, I. (2009) 'Rules for mapping SQL relational databases to OWL ontologies', in Sicilia M.A. and Lytras M.D. (Eds.): *Metadata and Semantics*, Springer, New York, pp.415–424.
- Astrova, I. and Kalja, A. (2007) 'Automatic transformation of OWL ontologies to SQL relational databases', *IADIS European Conf. Data Mining (MCCSIS)*, 5–7 July, Lisbon, Portugal.
- Astrova, I., Korda, N. and Kalja, A. (2007) 'Storing OWL ontologies in SQL relational databases', *International Journal of Electrical, Computer, and Systems Engineering*, Vol. 1, No. 4, pp.242–247.
- Athanasiadis, I. N., Villa, F. and Rizzoli, A. E. (2007) 'Ontologies, JavaBeans and relational databases for enabling semantic programming', *The Thirty-First Annual International Computer Software and Applications Conference, Compsac 2007*, 23–27 July, Beijing, China, pp.341–346.
- Atkinson, C. and Kuhne, T. (2002) 'The role of metamodeling in MDA', *International Workshop in Software Model Engineering*, Dresden, Germany.
- Atkinson, C. and Kuhne, T. (2003) 'Model-driven development: a metamodeling foundation', *IEEE Software*, Vol. 20, No. 5, pp.36–41.
- Ayed, D., Delanote, D. and Berbers, Y. (2007) 'MDD approach for the development of context-aware applications', *6th International and Interdisciplinary Conference, CONTEXT 2007. Modeling and Using Context*, 20–24 August, Roskilde, Denmark, pp.15–28.
- Baader, F., McGuinness, D., Nardi, D. and Patel-Schneider, P. (2003) *The Description Logic Handbook: Theory, Implementation and Applications*, Cambridge University Press, Cambridge.
- Baldauf, M., Dustdar, S. and Rosenberg, F. (2007) 'A survey on context-aware systems', *International Journal of Ad Hoc and Ubiquitous Computing*, Vol. 2, No. 4, pp.263–277.
- Banavar, G., Beck, J., Gluzberg, E., Munson, J., Sussman, J. and Zukowski, D. (2000) 'Challenges: an application model for pervasive computing', *Sixth Annual International Conference on Mobile Computing and Networking, MobiCom 2000*, 6–11 August, Boston, MA, USA, pp.266–274.
- Bardram, J.E. (2005) 'The Java Context Awareness Framework (JCAF) – a service infrastructure and programming framework for context-aware applications', *Third International Conference, Pervasive 2005. Pervasive Computing*, 8–13 May, Munich, Germany, pp.98–115.
- Bechhofer, S., Van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F. and Stein, L.A. (2004) *OWL Web Ontology Language Reference*, W3C, Obtained through the internet: <http://www.w3.org/TR/owl-ref/> [Accessed 2011].
- Begier, B. (2010) 'Users' involvement may help respect social and ethical values and improve software quality', *Information Systems Frontiers*, Vol. 12, No. 4, pp.389–397.
- Bell, B.S. and Kozlowski, S.W.J. (2002) 'Adaptive guidance: enhancing self-regulation, knowledge, and performance in technology-based training', *Personnel Psychology*, Vol. 55, No. 2, pp.267–306.
- Berstel, B., Bonnard, P., Bry, F., Eckert, M. and Patranjan, P.L. (2007) 'Reactive rules on the web', *3rd International Summer School 2007. Reasoning Web*, 3–7 September, Dresden, Germany.
- Besnard, P., Cordier, M.O. and Moinard, Y. (2008) 'Ontology-based inference for causal explanation', *Integrated Computer-Aided Engineering*, Vol. 15, No. 4, pp.351–367.
- Bettini, C., Brdiczka, O., Henriksen, K., Indulska, J., Nicklas, D., Ranganathan, A. and Riboni, D. (2010) 'A survey of context modelling and reasoning techniques', *Pervasive and Mobile Computing*, Vol. 6, No. 2, pp.161–180.
- Bezivin, J., Moreno, A.V., Molina, J.G. and Rossi, G. (2008) 'Presentation: MDA at the age of seven: past, present and future', *UPGRADE*, Vol. 11, No. 2, pp.4–6.
- Bick, M. and Kummer, T.F. (2008) 'Ambient intelligence and ubiquitous computing', in Adelsberger, H.H., Kinshuk, Pawlowski, J.M. and Sampson, D. (Eds.): *Handbook on Information Technologies for Education and Training*, Springer, Berlin, <http://scis.athabasca.ca/scis/staff/faculty.php?id=kinshuk>, pp.79–100.
- Binh An, T., Young-Koo, L. and Sung-Young, L. (2005) 'Modeling and reasoning about uncertainty in context-aware systems', *IEEE International Conference on e-Business Engineering*, 18–20 October, Beijing, China.

- Blythe, M., Overbeeke, K., Monk, A.F. and Wright, P.C. (2003) *Funology: From Usability to Enjoyment*, Kluwer, The Netherlands.
- Boley, H. and Kifer, M. (2010) 'A guide to the basic logic dialect for rule interchange on the web', *IEEE Transactions on Knowledge and Data Engineering*, Vol. 22, No. 11, pp.1593–1608.
- Boley, H., Hallmark, G., Kifer, M., Paschke, A., Polleres, A. and Reynolds, D. (2009) *RIF Core Dialect*, W3C, Obtained through the internet: <http://www.w3.org/TR/rif-core/> [Accessed 2011].
- Boley, H., Kifer, M., Patranjan, P.L. and Polleres, A. (2007) 'Rule interchange on the web', *Third International Summer School 2007, Reasoning Web*, 3–7 September, Dresden, Germany.
- Boley, H., Tabet, S. and Wagner, G. (2001) 'Design rationale of RuleML: a markup language for semantic web rules', *First Semantic Web Working Symposium*, Stanford University, Stanford California.
- Booch, G., Brown, A., Iyengar, S., Rumbaugh, J. and Selic, B. (2004) 'An MDA manifesto', *MDA Journal*, Vol. 5, pp.2–9.
- Brockmans, S., Volz, R., Eberhart, A. and Löffler, P. (2004) 'Visual modeling of OWL DL ontologies using UML', *Third International Semantic Web Conference, Iswc 2004. The Semantic Web*, 7–11 November, Hiroshima, Japan, pp.198–213.
- Brown, E. and Cairns, P. (2004) 'A grounded investigation of game immersion', *Conference on Human Factors in Computing Systems*, 24–29 April, Vienna, Austria, pp.1297–1300.
- Brusilovsky, P., Kobsa, A. and Wolfgang, N. (Eds.) (2007) *The Adaptive Web*, Springer, Berlin, Heidelberg.
- Chaari, T., Ejigu, D., Laforest, F. and Scuturici, V.M. (2006) 'Modeling and using context in adapting applications to pervasive environments', *ACS/IEEE International Conference on Pervasive Services*, 26–29 June, Lyon, France.
- Chapman, P., Selvarajah, S. and Webster, J. (1999) 'Engagement in multimedia training systems', *Thirty-Second Annual Hawaii International Conference on System Sciences, HICSS-32*, 5–8 January, Maui, HI, USA, p.1084.
- Chen, H., Finin, T., Joshi, A., Kagal, L., Perich, F. and Chakraborty, D. (2004) 'Intelligent agents meet the semantic web in smart spaces', *IEEE Internet Computing*, Vol. 8, No. 6, pp.69–79.
- Colburn, T. (2000) *Philosophy and Computer Science*, M.E. Sharpe, New York.
- Colombo, G., Mosca, A. and Sartori, F. (2007) 'Towards the design of intelligent CAD systems: an ontological approach', *Advanced Engineering Informatics*, Vol. 21, No. 2, pp.153–168.
- Constantine, L.L. (2006) 'Trusted interaction: user control and system responsibilities in interaction design for information systems', *18th International Conference, CAiSE 2006. Advanced Information Systems Engineering*, 5–9 June, Luxembourg, pp.20–30.
- Cook, D.J. and Das, S.K. (2007) 'How smart are our environments? An updated look at the state of the art', *Pervasive and Mobile Computing*, Vol. 3, No. 2, pp.53–73.
- Corbalan, G., Kester, L. and Van Merriënboer, J.J.G. (2008) 'Selecting learning tasks: effects of adaptation and shared control on learning efficiency and task involvement', *Contemporary Educational Psychology*, Vol. 33, No. 4, pp.733–756.
- Coutaz, J., Crowley, J.L., Dobson, S. and Garlan, D. (2005) 'Context is key', *Communications of the ACM*, Vol. 48, No. 3, pp.49–53.
- Dacosta, M.D., Obrst, L.J. and Smith, K.T. (2003) *The Semantic Web*, Wiley, Indianapolis.
- Dao-Tran, M., Eiter, T. and Krennwallner, T. (2009) 'Realizing default logic over description logic knowledge bases', *10th European Conference, ECSQARU 2009. Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, 1–3 July, Verona, Italy.
- Das, S.K. and Roy, N. (2008) 'Learning, prediction and mediation of context uncertainty in smart pervasive environments', *Otm 2008 Workshops. On the Move to Meaningful Internet Systems*, 9–14 November, Monterrey, Mexico.
- Devedzic, V. (2002) 'Understanding ontological engineering', *Communications of the ACM*, Vol. 45, No. 4, pp.136–144.
- Dey, A.K. (2001) 'Understanding and using context', *Personal and Ubiquitous Computing*, Vol. 5, No. 1, pp.4–7.
- Dey, A.K. (2009a) 'Context-aware computing', in Krumm, J. (Ed.): *Ubiquitous Computing Fundamentals*, CRC Press, pp.321–352.
- Dey, A.K. (2009b) 'Modeling and intelligibility in ambient environments', *Journal of Ambient Intelligence and Smart Environments*, Vol. 1, No. 1, pp.57–62.
- Dey, A.K. and Mankoff, J. (2005) 'Designing mediation for context-aware applications', *ACM Transactions on Computer-Human Interaction*, Vol. 12, No. 1, pp.53–80.
- Diouf, M., Maabout, S. and Musumbu, K. (2007) 'Merging model driven architecture and semantic Web for business rules generation', *1st International Conference, RR 2007, Web Reasoning and Rule Systems*, 7–8 June, Innsbruck, Austria.
- Djuric, D., Gasevic, D., Devedzic, V. and Damjanovic, V. (2005) 'A UML profile for OWL ontologies', *European MDA Workshops: Foundations and Applications, MDFA 2003 and MDFA 2004. Model Driven Architecture*, 26–27 June, Twente, The Netherlands, 10–11 June, 2004, Linköping, Sweden.
- Drabent, W. (2010) 'Hybrid reasoning with non-monotonic rules', *6th International Summer School 2010. Reasoning Web: Semantic Technologies for Software Engineering*, 30 August–3 September, Dresden, Germany.
- Dreyfus, H.L. (1993) 'What computers still can't do', *Deutsche Zeitschrift Fur Philosophie*, Vol. 41, No. 4, pp.653–680.
- Dreyfus, H.L. and Dreyfus, S.E. (2000) *Mind and Machine*, Free Press, New York.
- Du, W. and Wang, L. (2008) 'Context-aware application programming for mobile devices', *C3S2E*, Montreal, Quebec, Canada.
- Eberhart, A. (2002) 'Automatic generation of Java/SQL based inference engines from RDF Schema and RuleML', *First International Semantic Web Conference, Iswc 2002. Semantic Web*, 9–12 June, Sardinia, Italy, pp.102–116.

- Economides, A.A. (2009) 'Adaptive context-aware pervasive and ubiquitous learning', *International Journal of Technology Enhanced Learning*, Vol. 1, No. 3, pp.169–192.
- Eiter, T., Brewka, G., Dao-Tran, M., Fink, M., Ianni, G. and Krennwallner, T. (2009) 'Combining nonmonotonic knowledge bases with external sources', *7th International Symposium, FroCoS 2009. Frontiers of Combining Systems*, 16–18 September, Trento, Italy.
- Eiter, T., Ianni, G., Krennwallner, T. and Polleres, A. (2008a) 'Rules and ontologies for the semantic web', *4th International Summer School 2008. Reasoning Web*, 7–11 September, Venice, Italy.
- Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R. and Tompits, H. (2008b) 'Combining answer set programming with description logics for the semantic web', *Artificial Intelligence*, Vol. 172, Nos. 12–13, pp.1495–1539.
- Endsley, M.R. (1996) 'Automation and situation awareness', in Prasuraman, R. and Mouloua, M. (Eds.): *Automation and Human Performance – Theory and Application*, Lawrence Erlbaum Associates, New Jersey, pp.163–181.
- Erickson, T. (2002) 'Some problems with the notion of context-aware computing – ask not for whom the cell phone tolls', *Communications of the ACM*, Vol. 45, No. 2, pp.102–104.
- Esposito, M. (2008) 'An ontological and non-monotonic rule-based approach to label medical images', *3rd International IEEE Conference on Signal-Image Technologies and Internet-Based System*, 16–18 December, Shanghai, China, pp.603–611.
- Fonseca, F. (2007) 'The double role of ontologies in information science research', *Journal of the American Society for Information Science and Technology*, Vol. 58, No. 6, pp.786–793.
- Fonseca, F. and Martin, J. (2007) 'Learning the differences between ontologies and conceptual schemas through ontology-driven information systems', *Journal of the Association for Information Systems*, Vol. 8, No. 2, pp.129–142.
- Fritzsche, M., Bruneliere, H., Vanhooff, B., Jouault, F. and Gilani, W. (2009) 'Applying Megamodelling to Model Driven Performance Engineering', *16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, 14–16 April, San Francisco, California, USA.
- Gali, A., Chen, C.X., Claypool, K.T. and Uceda-Sosa, R. (2004) 'From ontology to relational databases', *ER 2004 Workshops CoMoGIS, CoMWIM, ECDM, CoMoA, DGOV, and eCOMO. Conceptual Modeling for Advanced Application Domains*, 8–12 November, Shanghai, China.
- Garlan, D., Siewiorek, D.P., Smailagic, A. and Steenkiste, P. (2002) 'Project Aura: toward distraction-free pervasive computing', *IEEE Pervasive Computing*, Vol. 1, No. 2, pp.22–31.
- Gasevic, D. and Devedzic, V. (2006) 'Petri net ontology', *Knowledge-Based Systems*, Vol. 19, No. 4, pp.220–234.
- Geihs, K., Barone, P., Eliassen, F., Floch, J., Fricke, R., Gjørven, E., Hallsteinsen, S., Horn, G., Khan, M.U., Mamelli, A., Papadopoulos, G.A., Paspallis, N., Reichle, R. and Stav, E. (2009) 'A comprehensive solution for application-level adaptation', *Software-Practice & Experience*, Vol. 39, No. 4, pp.385–422.
- Georges, T.M. (2004) *Digital Soul: Intelligent Machines and Human Values*, Westview Press, New York.
- Girardi, R. and Faria, C. (2003) 'A generic ontology for the specification of domain models', *1st International Workshop on Component Engineering Methodology (WCEM'03) at Second International Conference on Generative Programming and Component Engineering*, 24 September, Erfurt, Germany.
- Gitzel, R., Korthaus, A. and Schader, M. (2007) 'Using established web engineering knowledge in model-driven approaches', *Science of Computer Programming*, Vol. 66, No. 2, pp.105–124.
- Goldman, N.M. (2003) 'Ontology-oriented programming: static typing for the inconsistent programmer', *Second International Semantic Web Conference, Iswc 2003. Semantic Web*, 20–23 October, Sanibel Island, FL, USA.
- Gomez-Perez, A., Fernandez-Lopez, M. and Corcho, O. (2003) *Ontological Engineering*, Springer-Verlag, Berlin Heidelberg.
- Gonzalez-Perez, C. and Sellers, B.H. (2007) 'Modelling software development methodologies: a conceptual foundation', *Journal of Systems and Software*, Vol. 80, No. 11, pp.1778–1796.
- Grosz, B.N., Horrocks, I., Volz, R. and Decker, S. (2003) 'Description logic programs: combining logic programs with description logic', *12th International Conference on World Wide Web, WWW'03*, 20–24 May, Budapest, Hungary.
- Gruninger, M. and Lee, J. (2002) 'Ontology – applications and design', *Communications of the ACM*, Vol. 45, No. 2, pp.39–41.
- Gu, T., Pung, H.K. and Zhang, D.Q. (2005) 'A service-oriented middleware for building context-aware services', *Journal of Network and Computer Applications*, Vol. 28, No. 1, pp.1–18.
- Guarino, N. (1998) 'Formal ontology and information systems', *Formal Ontology in Information Systems, FOIS98*, 6–8 June, Trento, Italy.
- Haghighi, P.D., Krishnaswamy, S., Zaslavsky, A. and Gaber, M.M. (2008) 'Reasoning about context in uncertain pervasive computing environments', *Third European Conference, EuroSSC 2008. Smart Sensing and Context*, 29–31 October, Zurich, Switzerland.
- Hagras, H. (2007) 'Embedding computational intelligence in pervasive spaces', *IEEE Pervasive Computing*, Vol. 6, No. 3, pp.85–89.
- Hassenzahl, M. and Tractinsky, N. (2006) 'User experience – a research agenda', *Behaviour & Information Technology*, Vol. 25, No. 2, pp.91–97.
- Hatala, M., Wakkary, R. and Kalantari, L. (2005) 'Rules and ontologies in support of real-time ubiquitous application', *Journal of Web Semantics*, Vol. 3, No. 1, pp.5–22.
- Helal, S. (2010) 'Programming pervasive spaces', *7th International Conference, UIC 2010. Ubiquitous Intelligence and Computing*, 26–29 October, Xian, China.
- Helmreich, S. (2000) *Silicon Second Nature: Culturing Artificial Life in a Digital World*, University of California Press, Berkeley, California.
- Henderson-Sellers, B. (2011) 'Bridging metamodels and ontologies in software engineering', *Journal of Systems and Software*, Vol. 84, No. 2, pp.301–313.
- Henricksen, K. and Indulska, J. (2006) 'Developing context-aware pervasive computing applications: models and approach', *Pervasive and Mobile Computing*, Vol. 2, No. 1, pp.37–64.

- Henricksen, K., Indulska, J. and Rakotonirainy, A. (2002) 'Modelling context information in pervasive computing systems', *1st International Conference, Pervasive 2002. Pervasive Computing*, 26–28 August, Zurich, Switzerland.
- Henze, N., Dolog, P. and Nejd, W. (2004) 'Reasoning and ontologies for personalized e-learning in the semantic web', *Educational Technology & Society*, Vol. 7, No. 4, pp.82–97.
- Hewlett-Packard Development Company (2011) *Jena: A Semantic Web Framework for Java*, Obtained through the internet: <http://jena.sourceforge.net> [Accessed 2011].
- Hitzler, P., Angele, J., Motik, B. and Studer, R. (2005) 'Bridging the paradigm gap with rules for OWL', *W3C Workshop on Rule Languages for Interoperability*, Washington, USA.
- Hofweber, T. (2004) *Logic and Ontology*, Obtained through the internet: <http://plato.stanford.edu/entries/logic-ontology/> [Accessed 2011].
- Horrocks, I. (2002) 'Reasoning with expressive description logics: theory and practice', *18th International Conference on Automated Deduction, CADE-18. Automated Deduction*, 27–30 July, Copenhagen, Denmark.
- Horrocks, I. and Patel-Schneider, P. F. (2004) 'A proposal for an OWL rules language', *The 13th International Conference on World Wide Web, WWW'04*, 17–22 May, New York, USA.
- Horrocks, I., Parsia, B., Patel-Schneider, P. and Hendler, J. (2005) 'Semantic web architecture: Stack or two towers?', *Third International Workshop, PPSWR 2005. Principles and Practice of Semantic Web Reasoning*, 11–16 September, Berlin Dagstuhl Castle, Germany.
- Horrocks, I., Patel-Schneider, P.F. and Van Harmelen, F. (2003) 'From SHIQ and RDF to OWL: the making of a web ontology language', *Journal of Web Semantics*, Vol. 1, No. 1, pp.7–26.
- Indulska, J. and Sutton, P. (2003) 'Location management in pervasive systems', *Australasian information Security Workshop Conference on ACSW Frontiers*.
- Jameson, A. and Schwarzkopf, E. (2002) 'Pros and cons of controllability: an empirical study', *Second International Conference, AH 2002. Adaptive Hypermedia and Adaptive Web-Based Systems*, 29–31 May, Malaga, Spain.
- Jensen, K. and Kristensen, L.M. (2009) *Coloured Petri Nets Modelling and Validation of Concurrent Systems*, Springer, Berlin Heidelberg.
- Kadiyala, M. and Crynes, B.L. (1998) 'Where's the proof? A review of literature on effectiveness of information technology in education', *28th Annual Frontiers in Education Conference, FIE'98*, 4–7 November, Tempe, Arizona, USA.
- Kalyanpur, A., Pastor, D.J., Battle, S. and Padget, J. (2004) 'Automatic mapping of OWL ontologies into Java', *16th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, 20–24 June, Banff, Canada.
- Kasabov, N. (2008) 'Evolving intelligence in humans & machines: integrative evolving connectionist systems approach', *IEEE Computational Intelligence Magazine*, Vol. 3, No. 3, pp.23–37.
- Katasonov, A. and Palviainen, M. (2010) 'Towards ontology-driven development of applications for smart environments', *8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, 29 March–2 April, Mannheim, Germany.
- Kazakov, Y., Sattler, U. and Zolin, E. (2007) 'How many legs do I have? Non-simple roles in number restrictions revisited', *14th International Conference, LPAR 2007. Logic for Programming, Artificial Intelligence, and Reasoning*, 15–19 October, Yerevan, Armenia.
- Kifer, M. (2005) 'Rules and ontologies in F-logic', *1st International Summer School Reasoning Web*, 25–29 July, Msida, Malta.
- Kifer, M., Lausen, G. and Wu, J. (1995) 'Logical-foundations of object-oriented and frame-based Languages', *Journal of the Association for Computing Machinery*, Vol. 42, No. 4, pp.741–843.
- Knublauch, H. (2004) 'Ontology-driven software development in the context of the semantic web: an example scenario with portege/OWL', *International Workshop on the Model-Driven Semantic Web*, Monterey, Canada.
- Korpipää, P., Hakkila, J., Kela, J., Ronkainen, S. and Kansala, I. (2004) 'Utilising context ontology in mobile device application personalisation', *The 3rd international conference on Mobile and Ubiquitous Multimedia*. College Park, Maryland.
- Krotzsch, M., Hitzler, P., Vrandečić, D. and Sintek, M. (2006) 'How to reason with OWL in a logic programming system', *Second International Conference on Rules and Rule Markup Languages for the Semantic Web*, 10–11 November, Athens, Georgia, USA.
- Krumm, J. (Ed.) (2009) *Ubiquitous Computing Fundamentals*, CRC Press, Boca Raton, Florida.
- Lehmann, J. and Gangemi, A. (2007) 'An ontology of physical causation as a basis for assessing causation in fact and attributing legal responsibility', *Artificial Intelligence and Law*, Vol. 15, No. 3, pp.301–321.
- Levesque, H.J. and Brachman, R.J. (1985) A fundamental tradeoff in knowledge representation and reasoning', *Readings in Knowledge Representation*, Morgan Kaufmann, San Francisco, California, pp.41–70.
- Lieberman, H., Paterno, F. and Wulf, V. (Eds.) (2006) *End-User Development*, Springer, Berlin.
- Lin, X., Li, S.P., Yang, Z.H. and Shi, W. (2005) 'Application-oriented context modeling and reasoning in pervasive computing', *5th International Conference on Computer and Information Technology, CIT'05*, 21–23 September, Shanghai, China.
- Maher, M.L., Merrick, K. and Macindoe, O. (2006) 'Intrinsically motivated intelligent sensed environments', *13th EG-ICE Workshop 2006. Intelligent Computing in Engineering and Architecture*, 25–30 June, Ascona, Switzerland.
- Mankoff, J., Abowd, G.D. and Hudson, S.E. (2000) 'OOPS: a toolkit supporting mediation techniques for resolving ambiguity in recognition-based interfaces', *Computers & Graphics*, Vol. 24, No. 6, pp.819–834.
- Mccarthy, J. (2007) 'From here to human-level AI', *Artificial Intelligence*, Vol. 171, No. 18, pp.1174–1182.
- Mehroliassani, M. and Elci, A. (2009) 'Developing a UML to OWL conversion model for semantic web based application development', *International Conference on Enterprise Information Systems and Web Technologies, EISWT-09*, Orlando, FL, USA.
- Mellor, S.J. and Balcer, M. (2002) *Executable UML – A Foundation for Model-Driven Architecture*, Addison-Wesley, Boston, Massachusetts.

- Mellor, S.J., Clark, A.N. and Futagami, T. (2003) 'Model-driven development', *IEEE Software*, Vol. 20, No. 5, pp.14–18.
- Mellor, S.J., Scott, K., Uhl, A. and Weise, D. (2002) 'Model-driven architecture', *OOIS 2002 Workshops. Advances in Object-Oriented Information Systems*, 2 September, Montpellier, France.
- Meservy, T.O. and Fenstermacher, K.D. (2005) 'Transforming software development: an MDA road map', *Computer*, Vol. 38, No. 9, pp.52–58.
- Motik, B. and Rosati, R. (2010) 'Reconciling description logics and rules', *Journal of the ACM*, Vol. 57, No. 5, pp.1–62.
- Motik, B., Horrocks, I., Rosati, R. and Sattler, U. (2006) 'Can OWL and logic programming live together happily ever after?', *5th International Semantic Web Conference, ISWC 2006. The Semantic Web*, 5–9 November, Athens, GA, USA.
- Motik, B., Studer, R. and Sattler, U. (2005) 'Query answering for OWL-DL with rules', *Journal of Web Semantics*, Vol. 3, No. 1, pp.41–60.
- Murata, T. (1989) 'Petri nets: properties, analysis and applications', *Proceedings of the IEEE*, Vol. 77, No. 4, pp.541–580.
- Murch, R. (2004) *Autonomic Computing*, IBM Press and Prentice-Hall, Englewood Cliffs, New Jersey.
- Nicklas, D., Grossmann, M., Minguéz, J. and Wieland, M. (2008) 'Adding high-level reasoning to efficient low-level context management: a hybrid approach', *6th Annual IEEE International Conference on Pervasive Computing and Communications, PerCom 2008*, 17–21 March, Hong Kong.
- Noguera, M., Hurtado, M.V., Rodriguez, M.L., Chung, L. and Garrido, J.L. (2010) 'Ontology-driven analysis of UML-based collaborative processes using OWL-DL and CPN', *Science of Computer Programming*, Vol. 75, No. 8, pp.726–760.
- Noy, N.F. and McGuinness, D.L. (2001) *Ontology Development 101: A Guide to Creating Your First Ontology*, Stanford University, Stanford.
- O'Brien, H.L. and Toms, E.G. (2008) 'What is user engagement? A conceptual framework for defining user engagement with technology', *Journal of the American Society for Information Science and Technology*, Vol. 59, No. 6, pp.938–955.
- OMG (2006) *UML 2.0 OCL Specification*.
- OMG (2008) *Software Process Engineering Meta-model (SPEM) Specification*, Technical Report ptc/2008-04-01, Object Management Group.
- OMG (2009a) *Ontology Definition Metamodel (ODM)*, <http://www.omg.org/spec/ODM/1.0/>, release date 2009, last accessed May, 2011.
- OMG (2009b) *Unified Modeling Language: Superstructure*, <http://www.omg.org/spec/UML/2.2/>, release date 2009, last accessed May, 2011.
- Padovitz, A., Loke, S.W. and Zaslavsky, A. (2008) 'Multiple-agent perspectives in reasoning about situations for context-aware pervasive computing systems', *IEEE Transactions on Systems Man and Cybernetics Part a-Systems and Humans*, Vol. 38, No. 4, pp.729–742.
- Pahl, C. (2007) 'Semantic model-driven architecting of service-based software systems', *Information and Software Technology*, Vol. 49, No. 8, pp.838–850.
- Pan, Y., Xie, G.T., Ma, L., Yang, Y., Qiu, Z.M. and Lee, J. (2006) 'Model-driven ontology engineering', in Spaccapietra, S. (Ed.): *Journal on Data Semantics VII*, Springer-Verlag, Berlin, pp.57–78.
- Parreiras, F.S. and Staab, S. (2010) 'Using ontologies with UML class-based modeling: the two use approach', *Data & Knowledge Engineering*, Vol. 69, No. 11, pp.1194–1207.
- Peng, X. and Silver, D.L. (2005) 'User control over user adaptation: a case study', *10th International Conference, UM 2005. User Modeling 2005*, 24–29 July, Edinburgh, UK.
- Perttunen, M., Riekki, J. and Lassila, O. (2009) 'Context representation and reasoning in pervasive computing: a review', *International Journal of Multimedia and Ubiquitous Engineering*, Vol. 4, No. 9, pp.1–28.
- Pervez, A. and Ryu, J. (2008) 'Safe physical human robot interaction-past, present and future', *Journal of Mechanical Science and Technology*, Vol. 22, No. 3, pp.469–483.
- Pidd, M. (2000) *Tools for Thinking – Modelling in Management Science*, Wiley, New York.
- Pisanelli, D.M., Gangemi, A. and Steve, G. (2002) 'Ontologies and information systems: the marriage of the centry?', *LYEE Workshop*, Paris.
- Preuveneers, D. (2010) *Context-Aware Adaptation for Ambient Intelligence: Concepts, Methods and Applications*, LAP Lambert Academic Publishing, Germany.
- Preuveneers, D. and Berbers, Y. (2008a) 'Internet of things: a context-awareness perspective', in Yan, L., Zhang, Y., Yang, L.T. and Ning, H. (Eds.): *The Internet of Things. From RFID to the Next-generation Pervasive Network Systems*, Auerbach Publications, Taylor & Francis Group, pp.287–307.
- Preuveneers, D. and Berbers, Y. (2008b) 'Pervasive services on the move: smart service diffusion on the OSGi framework', *5th International Conference, UIC 2008. Ubiquitous Intelligence and Computing*, 23–25 June, Oslo, Norway.
- Ranganathan, A., Al-Muhtadi, J. and Campbell, R.H. (2004) 'Reasoning about uncertain contexts in pervasive computing environments', *IEEE Pervasive Computing*, Vol. 3, No. 2, pp.62–70.
- Ranganathan, A., Mcgrath, R.E., Campbell, R.H. and Mickunas, M.D. (2003) 'Use of ontologies in a pervasive computing environment', *Knowledge Engineering Review*, Vol. 18, No. 3, pp.209–220.
- Reiter, R. (1992) 'What should a database know?', *Journal of Logic Programming*, Vol. 14, Nos. 1–2, pp.127–153.
- Rodriguez, D., Garcia, E., Sanchez, S. and Nuzzi, C.R.S. (2010) 'Defining software process model constraints with rules using OWL and SWRL', *International Journal of Software Engineering and Knowledge Engineering*, Vol. 20, No. 4, pp.533–548.
- Roy, N., Julien, C. and Das, S.K. (2009) 'Resolving and mediating ambiguous contexts for pervasive care environments', *6th Annual International Mobile and Ubiquitous Systems: Networking & Services, MobiQuitous 2009*, Toronto, ON, Canada.
- Ruiz, F. and Hilara, J.R. (2006) 'Using ontologies in software engineering and technology', in Calero, C., Ruiz, F. and Piattini, M. (Eds.): *Ontologies in Software Engineering and Software Technology*, Springer-Verlag, pp.49–102.
- Ruiz, F., Vizcaino, A., Piattini, M. and Garcia, F. (2004) 'An ontology for the management of software maintenance projects', *International Journal of Software Engineering and Knowledge Engineering*, Vol. 14, No. 3, pp.323–349.
- Salehie, M. and Tahvildari, L. (2009) 'Self-adaptive software: landscape and research challenges', *ACM Transactions on Autonomous and Adaptive Systems*, Vol. 4, No. 2, pp.1–42.

- Satyanarayanan, M. (2001) 'Pervasive computing: vision and challenges', *IEEE Personal Communications*, Vol. 8, No. 4, pp.10–17.
- Schilit, B., Adams, N. and Want, R. (1994) 'Context-aware computing applications', *Workshop on Mobile Computing Systems and Applications*. Santa Cruz, CA, USA.
- Schmidt, D.C. (2006) 'Model-driven engineering', *Computer*, Vol. 39, No. 2, pp.25–31.
- Schneider-Hufschmidt, M., Kühme, T. and Malinowski, U. (Eds.) (1993) *Adaptive User Interfaces: Principles and Practice*, North-Holland, Amsterdam.
- Searle, J.R. (1980) 'Minds, brains, and programs', *Behavioral and Brain Sciences*, Vol. 3, No. 3, pp.417–425.
- Selic, B. (2003) 'The pragmatics of model-driven development', *IEEE Software*, Vol. 20, No. 5, pp.19–25.
- Serral, E., Valderas, P. and Pelechano, V. (2010) 'Towards the model driven development of context-aware pervasive systems', *Pervasive and Mobile Computing*, Vol. 6, No. 2, pp.254–280.
- Shadbolt, N., Hall, W. and Berners-Lee, T. (2006) 'The semantic web revisited', *IEEE Intelligent Systems*, Vol. 21, No. 3, pp.96–101.
- Sicilia, M.A., Garcia-Barriocanal, E., Sanchez-Alonso, S. and Rodriguez-Garcia, D. (2009) 'Ontologies of engineering knowledge: general structure and the case of software engineering', *Knowledge Engineering Review*, Vol. 24, No. 3, pp.309–326.
- Sicilia, M.A., Lytras, M., Rodriguez, E. and Garcia-Barriocanal, E. (2006) 'Integrating descriptions of knowledge management learning activities into large ontological structures: a case study', *Data & Knowledge Engineering*, Vol. 57, No. 2, pp.111–121.
- Silva Parreiras, F., Staab, S. and Winter, A. (2007) 'On marrying ontological and metamodeling technical spaces', *6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Dubrovnik, Croatia.
- Singh, Y. and Sood, M. (2009) 'Model driven architecture: a perspective', *IEEE International Advance Computing Conference, IACC 2009*, Patiala, India.
- Soyly, A. and De Causmaecker, P. (2009) 'Merging model driven and ontology driven system development approaches pervasive computing perspective', *24th International Symposium on Computer and Information Sciences (ISCIS)*, 14–16 September, Guzelyurt, Cyprus.
- Soyly, A., De Causmaecker, P. and Desmet, P. (2009) 'Context and adaptivity in pervasive computing environments: links with software engineering and ontological engineering', *Journal of Software*, Vol. 4, No. 9, pp.992–1013.
- Soyly, A., De Causmaecker, P. and Wild, F. (2010a) 'Ubiquitous web for ubiquitous environments: the role of embedded semantics', *Journal of Mobile Multimedia*, Vol. 6, No. 1, pp.26–48.
- Soyly, A., Modritscher, F. and De Causmaecker, P. (2010b) 'Utilizing embedded semantics for user-driven design of pervasive environments', *4th International Conference, MTSR 2009. Metadata and Semantic Research*, 20–22 October, Alcalá de Henares, Spain.
- Soyly, A., Vandewaetere, M., Wauters, K., Jacques, I., De Causmaecker, P., Desmet, P., Clarebout, G. and Van Den Noortgate, W. (2010c) 'Ontology-driven Adaptive and Pervasive Learning Environments – APLEs: an interdisciplinary approach', *First International Conference on Interdisciplinary Research on Technology, Education and Communication (ITEC 2010). Interdisciplinary Approaches to Adaptive Learning: A Look at the Neighbours*, 25–27 May, Kortrijk, Belgium.
- Spiekermann, S. (2008) *User Control in Ubiquitous Computing: Design Alternatives and User Acceptance*, Shaker Verlag, Aachen.
- Strang, T. and Linnhoff-Popien, C. (2004) 'A context modelling survey', *Workshop on Advanced Context Modelling*, Nottingham, UK.
- Studer, R., Benjamins, V.R. and Fensel, D. (1998) 'Knowledge engineering: principles and methods', *Data & Knowledge Engineering*, Vol. 25, Nos. 1–2, pp.161–197.
- Tetlow, P., Pan, J.Z., Oberle, D., Wallace, E., Uschold, M. and Kendall, E. (2006) *Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering*, W3C, Obtained through the internet: <http://www.w3.org/2001/sw/BestPractices/SE/ODA/> [Accessed 2011].
- Tribus, M. and Fitts, G. (1968) 'Widget problem revisited', *IEEE Transactions on Systems Science and Cybernetics*, Vol. SSC4, No. 3, pp.241–248.
- Uschold, M. and Gruninger, M. (1996) 'Ontologies: principles, methods and applications', *Knowledge Engineering Review*, Vol. 11, No. 2, pp.93–136.
- Uschold, M. and Jasper, R. (1999) 'A framework for understanding and classifying ontology applications', *Workshop on Ontologies and Problem-Solving Methods, IJCAI-99*, 2 August, Stockholm, Sweden.
- Valiente, M.C. (2010) 'A systematic review of research on integration of ontologies with the model-driven approach', *International Journal of Metadata, Semantics and Ontologies*, Vol. 5, No. 2, pp.134–150.
- Valla, M. (2010) *Final Research Results on Methods, Languages, Algorithms, and Tools to Modeling and Management of Context*, MUSIC Project, http://www.ist-music.eu/docs/MUSIC_D2.4.pdf
- Vasilecas, O. and Bugaite, D. (2007) 'An algorithm for the automatic transformation of ontology axioms into a rule model', *International Conference on Computer Systems and Technologies, CompSysTech 2007*, 14–15 June, Bulgaria.
- Vasilecas, O., Kalibatiene, D. and Guizzardi, G. (2009) 'Towards a formal method for the transformation of ontology axioms to application domain rules', *Information Technology and Control*, Vol. 38, No. 4, pp.271–282.
- Völkel, M. (2006) 'RDFReactor – from ontologies to programatic data access', *Jena User Conference*, HP Bristol.
- Vysniauskas, E. and Nemuraite, L. (2006) 'Transforming ontology representation from OWL to relational database', *Information Technology and Control*, Vol. 35, No. 3A, pp.335–345.
- Vysniauskas, E., Nemuraite, L. and Sukys, A. (2010) 'A hybrid approach for relating OWL 2 ontologies and relational databases', *9th International Conference, BIR 2010. Perspectives in Business Informatics Research*, 29 September–1 October, Rostock Germany.

- Wandke, H. (2005) 'Assistance in human-machine interaction: a conceptual framework and a proposal for a taxonomy', *Theoretical Issues in Ergonomics Science*, Vol. 6, No. 2, pp.129–155.
- Wang, X. and Chan, C.W. (2001) *Ontology Modeling using UML*, Springer-Verlag London Ltd., Godalming.
- Weiser, M. (1991) 'The computer for the 21st century', *Scientific American (International Edition)*, Vol. 265, No. 3, pp.66–75.
- Wild, F., Mödritscher, F. and Sigurdarson, S.E. (2008) 'Designing for change: mash-up personal learning environments', *eLearning Papers*, Vol. 9.
- Winograd, T. and Flores, F. (1987) 'On understanding computers and cognition: a new foundation for design', *Artificial Intelligence*, Vol. 31, No. 2, pp.250–261.
- Woods, D.D. (1996) 'Decomposing automation: apparent simplicity, real complexity', in Parasuraman, R. and Mouloua, M. (Eds.): *Automation and Human Performance – Theory and Application*, Lawrence Erlbaum Associates, New Jersey, pp.3–17.
- Zadeh, L.A. (2008) 'Toward human level machine intelligence – Is it achievable? The need for a paradigm shift', *IEEE Computational Intelligence Magazine*, Vol. 3, No. 3, pp.11–22.
- Zhongli, D. and Yun, P. (2004) 'A probabilistic extension to ontology language OWL', *The 37th Annual Hawaii International Conference on System Sciences, HICSS '04*, 5–8 January, Big Island, HI.
- Zimbardo, P.G. and Gerrig, R.J. (1996) *Psychologie*, Springer Verlag, Berlin, Heidelberg, New York.