Using Contexts to Prove and Share Situations

Patrick Barlatier and **Richard Dapoigny**
LISTIC/Polytech'Savoie, University of Savoie, France,
email:richard.dapoigny@univ-savoie.fr

Abstract

The context paradigm emerges from different areas of Artificial Intelligence. However, while significative formalizations have been proposed, contexts are either mapped on independent micro-theories or considered as different concurrent viewpoints with mappings between contexts to export/import knowledge. These logical formalisms focus on the semantic level and do not take into account dynamic low-level information such as those available from sensors. This information is a key element of contexts in pervasive computing environments. In this paper, we introduce a formal framework where the knowledge representation of context bridges the gap between semantic high-level and low-level knowledge. The logical reasoning based on intuitionistic type theory and the Curry-Howard isomorphism is able to incorporate expert knowledge as well as technical resources such as task properties. Based on our context model, we also present the foundations of a Context-Aware architecture (Softweaver) for building of context-aware services.

INTRODUCTION

In this paper we examine the notion of context within the area of Artificial Intelligence (AI). Context is an issue which arises in many different areas of AI such as in Natural Language Processing (Clark and Carlson 1981; Leech 1981), Categorization (Barwise and Seligman 1992), Knowledge Representation and Reasoning (McCarthy 1987; Shoham 1991), and more recently in Ubiquitous Computing and Context-aware Web Engineering (Kaltz04). While there has been an increasing amount of research focusing on the problems related to the modelling, representation and use of context (Brezillon 1999), few conceptual models have given meaningful solutions and implementations. The basic goal of context-aware systems is to make software aware of the environment and to adapt to their changing context. However, there is a lack of appropriate formal models for dynamic environments. Context models are often expressed either through logical formalisms or with ontology-based approaches, but they show some difficulty to take in account additional information, to cope with spatio-temporal change, and to cover non-monotonic features.

Copyright © 2007, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

Context is ontologically speaking a moment universal: it has no meaning by itself, but must be related to some concept, but which one? How to describe contextual facts and interrelationships in a precise and traceable manner? Context sharing capabilities are required but how to provide a shared understanding? How the representation of context should cover different abstraction levels (encapsulation). The logic must take in account partial knowledge, it must be dynamic to cope with change and it must be decidable for concrete applications.

This enumeration of problems argues for an unification of the two approaches, i.e., a logical framework and an ontological support. We have suggested a logical framework based on intuitionism and on type theory to represent contexts from an AI perspective (Dapoigny & Barlatier 2007). It combines the strengths of both approaches while trying not to carry their specific weaknesses into the resulting formal framework. For this purpose, it relies both on a knowledge representation with ontologies and on a logical reasoning with Dependent Record Types (DRT) based on intuitionistic type theory and the Curry-Howard isomorphism. This logic model can be applied to any kind of process-based applications. In this paper, we recall the basic principles of the logical framework and describe the context-sensitive reasoner. It makes use of DRT to represent typical problem-solving situations. The first section summarizes relevant works on context formalization in AI. In section 2, we present the Dependent Record Types based on intuitionistic logic including context types and context tokens. The third section gives some account of the implementation including a Graphical User Interface (GUI), a reasoning engine[1] and a local ontology. The last section concludes and presents some future ways of investigation.

Context in AI

The need for a Context Model

The issue of context has been emerged in various areas of AI, including knowledge representation, natural language processing, and intelligent information retrieval. In context-aware systems, a well-known definition is expressed as follows "a system is context-aware if it uses context to provide

[1]The information the GUI obtains from its users is expected to increase the accuracy of the reasoning engine.

relevant information and/or services to the user, where relevancy depends on the user's task" (Anind K. 2000) But such a definition requires first to categorize explicitly the concept of context. The strong link between contexts and situations is highlighted in (Giunchiglia 1992) where three different possibilities are detailed - a fourth one is the combination of the three basic ones -. In the first one, a given situation is related to multiple contexts, each being a different approximate of the situation. These contexts are different because they correspond to different goals that we have in mind, arguing for a *context-of* definition. In his one-to-one correspondence between situation and context, the author gives an example involving a time evolution of a physical process. Considering first that a physical process has a global goal and second, that this goal can be divided into sub-goals, each of them being achieved successively, it results that, at a given time a single context is available - i.e., the *context-of* the related sub-goal -. The third possibility, in which a single context corresponds to many situations, argues for a hierarchical property of contexts. If we can merge all contexts corresponding to a situation, provided that the situation can be itself divided into sub-parts each of them related to a context, then we can conclude that context must offer hierarchical properties. As a consequence, these assumptions give some evidence for an ontological definition.

Related Works

One of the most difficult problems facing context modelling is defining a theory incorporating linguistic knowledge from ontologies and logic. Most works on conceptual modelling of contexts stem from two basic approaches, that are either ontology-based or logic-based.

Ontology-based approaches are a promising framework to specify concepts and their interrelations (Mike Uschold 1996). They are particularly suitable to project parts of the information describing and being used in our daily life onto a data structure utilizable by computers. The development of context models based on ontologies take advantage of their knowledge sharing, logic inferencing and knowledge reuse capabilities. In (Wang *et al.* 2004), the authors have created an upper ontology which captures general features of basic contextual entities and a collection of domain specific ontologies with their features in each subdomain. The CoBrA system (Chen, Finin, & Joshi 2003) provides a set of ontological concepts to characterize entities such as persons, places or several other kinds of objects within their contexts. It implements a broker-centric agent architecture to provide runtime support for context-aware systems. A recent survey of the most significant context modelling approaches (Strang & Linnhoff-Popien 2004) argued that ontology-based approaches seem one of the most promising assets for context modelling.

In logic-based models, contextual information is introduced, updated and deleted in terms of facts or inferred from a set of rules. In the early nineties's, the most significant general foundations for reasoning about contexts have explored logics rooted in the Situation Calculus. In (McCarthy 1993), contexts are first class objects and the formula $ist(c, p)$ asserts that the formula p is true in the context c. Lifting rules are able to relate the truth in one context to the truth in another one. In addition, operations such as entering and leaving contexts were introduced. This work was the starting point for a formalization in propositional logic (Buvac, Buvac, & Mason 1995) extended to first-order languages (Buvac 1996). In (Giunchiglia 1992), the formalization relies on two assumptions, namely the locality principle where reasoning happens inside a context and the principle of compatibility where relationships can occur between reasoning processes in different contexts. The approach known as Multi-context Systems considers a context as a subset of the complete state of an individual. Contexts are seen as partial theories which can interact with each other through bridge rules allowing to infer a conclusion in one context from a premise in another one. The core problem with these approaches is not the lack of powerful rules engine, but rather the lack of ways to easily represent real-world scenarios in terms of rules. More recently, an approach of contexts exploring type-theoretic ideas has been proposed in (Thomason 1998)(Thomason 1999). Three primitive syntactic constructions are derived from type theory, namely identity, functional application and lambda abstraction. Built on a previous work on intensional logic (Montague 1970), propositions are defined as sets of possible worlds and a context is identified with a set of propositions. In Human Computer Interactions, context is seen as a feature of interaction - a property of information, or of objects - and the proposed a model includes context and activity which are mutually constituent (Dourish 2004).

The Context Theory

The Context Logic

While the approaches described in section cover a wide spectrum of AI modelling, they show some difficulty to manage the intensional aspect. Most of them present modality as a solution, but modal logics rely on the "essentialism" concept highlighting arbitrary axioms (Girard. 1989). In addition they suffer from the limitations of reasoning capabilities of first-order logic, the lack of dynamic capabilities and the need of partial validation. One important aspect already underlined in (Giunchiglia 1992), is the locality principle. This principle which states that different portions of knowledge are defined, is taken as a basis in the present work.

Then, observing that physical objects are a central element of process activity and building on recent works (Boldini 2000; Ranta 2004; Villadsen 2004), we have proposed a context model based on Intuitionistic Type Theory (ITT) for engineering applications. Intuitionism says that only those mathematical concepts that can be demonstrated, or constructed are legitimate. We consider the *context-of* the application, that is, the *context-of* the global goal since the application is itself characterized by an objective. The analysis of contextual reasoning requires a sound and expressive formalism. Widely used in Natural Language processing (Boldini 2000; Ranta 2004) and in Programming Languages (Bove & Capretta 2001; Paulson 1989; Coquand & Coquand 1999), ITT (Martin-Löf 1982) has been proven to be appropriate to support the linguistic nature of physical situations. We extend these works with the description of contexts. However, contexts are not situations but they are related to them and since contexts describe knowledge extracted from situations, they have a natural expressivity with types. Moreover, ITT provides an intensional type theory for representing most of the features characterizing contexts in an AI perspective. In the framework of physical processes, we introduce the Context Record Structure (CRS) described by a record in which the fields detail the physical items (i.e., objects and their properties) of that context. Record types and record tokens (i.e., instances of types) are also introduced to separate the specification of potential contexts through types with their implementation through tokens. A further attraction is that this subdivision also supports ontological engineering for the specification of the applications during the design step.

The theory of types has been extended with Dependent Record Types (DRT) (Betarte 2000; Kopylov 2003) formalizing a proof of a basic property of groups. Their ability to provide a simple structure that can be reused to specify different kinds of structured semantic objects is very attractive.

Definition 1 *A dependent record type is a sequence of fields in which labels l_i correspond to certain types T_i, that is, each successive field can depend on the values of the preceding fields:*

$$< l_1 : T_1, l_2 : T_2(l_1) \dots, l_n : T_n(l_1 \dots l_{n-1}) > \quad (1)$$

where the type T_i may depend on the preceding labels $l_1, ..., l_{i-1}$.

A similar definition holds for record tokens where a sequence of values is such that a value v_i can depend on the values of the preceding fields $l_1, ..., l_{i-1}$:

$$< l_1 = v_1, ..., l_n = v_n > \quad (2)$$

The empty sequence $<>$ is a record type and the type T_i is a family of types over the record type $l_1 : T_1, ..., l_{i-1} : T_{i-1}$. Assuming that Γ is a valid context[2], the record type formation rules are, provided that l is not already declared in R :

$$\frac{}{\Gamma \vdash <>: record-type} \qquad \frac{R : record-type}{\Gamma \vdash R \sqsubseteq <>}$$

$$\frac{\Gamma \vdash R : record-type \;\; \Gamma \vdash T : record-type \rightarrow type}{\Gamma \vdash < R, l : T >: record-type} \quad (3)$$

We also assume that the general rules of construction for Set and $Prop$ types are valid and that primitive syntactic constructions - i.e., equality, functional application and lambda abstraction - hold (for more details see (Martin-Löf 1982)). An important aspect of DRT is that sub-typing is allowed, for example a DRT with additional fields not mentioned in the type is still of that type. We can use the notion of record-type to offer the most rudimentary notion of physical context through the CRS, namely that it is a record which carries information about the semantic aspect of any physical variable. In such a way, CRS can range from the context of a single variable involved in a physical equation to the context of a real life situation. While types describe potential properties of the real world, tokens are dedicated to the run-time process. From basic ground types, e.g., $Location$, $Phys\_quantity$ - physical quantity -, ..., complex context types can be composed with more than one entity. Propositions are treated as individuals that can be arguments of predicates. Since the Curry-Howard isomorphism identifies proofs with programs, it can be used to prove a specification, that is to say, to select which definitions are needed for the specification to work properly.

$$\begin{array}{ll}
x : Vehicle & x = truck \\
y : RegistrationNumber & y = 2678KX69 \\
l_1 : GPSLongitude & l_1 = 12.0987 \\
l_2 : GPSLatitude & l_2 = 67.2365 \\
t_e : evTime & t_e = 2/11/06.11 : 33 \\
t_m : maxTime & t_m = 2/11/06.12 : 00 \\
q_1 : has\_identification(x, y) & q_1 = p_1 \\
q_2 : has\_Location(x, l_1, l_2) & q_2 = p_2 \\
t : Lt(evTime, maxTime) & t = p_3
\end{array}$$

In this example, p_1 is a proof of $has\_identification(truck, 2678KX69)$, p_2 is a proof of $has\_Longitude(truck, 12.0987)$ p_3 is a proof of $has\_Latitude(truck, 67.2365)$ and p_2 is a proof that the evaluation time $2/11/06.11 : 33$ is left than the maximum allowed time $2/11/06.12 : 00$. It states finally that the registration number of all vehicles present at this place before $2/11/06.12 : 00$ will be registered.

To cope with partial knowledge, a subtyping mechanism in dependent type theories is required as a crucial step toward large-scale applications. Major works either introduce coercive definition rules (Luo 1999) or record-based subtyping (Betarte 2000) Subtyping offers a great deal of flexibility for use in type theories but also raises some questions, such as coercions with non-intuitive actions. This question requires the knowledge of all possible coercions used for a given term and their precise effect, which is untractable in practice. This problem can be avoided by imposing semantic constraints on coercions: this is the case in record-based subtyping that we shall adopt here. The extension of a physical context type needs to define some basic rules of construction.

$$\frac{\Gamma \vdash c_1 : PC-type \;\; \Gamma \vdash c_2 : PC-type}{\Gamma \vdash c_1 \oplus c_2 : PC-type}$$

$$\frac{\Gamma \vdash c : PC-type}{\Gamma \vdash c \sqsubseteq <>}$$

$$\frac{\Gamma \vdash c_1 \oplus c_2 : PC-type}{\Gamma \vdash c_1 \oplus c_2 \sqsubseteq c_1} \qquad \frac{\Gamma \vdash c_1 \oplus c_2 : PC-type}{\Gamma \vdash c_1 \oplus c_2 \sqsubseteq c_2}$$

The first rule asserts that a concatenation of context types is again a context type. The second rule states that any context type is a sub-type of the empty sequence - since a context type with no labels doesn't impose any constraints on its objects -. The two remaining rules specify that any con-

[2]A valid context in type theory is a sequence $x_1 : T_1, \dots x_n : T_n$ such that there is a judgment having it as left side of a sequent.

catenation of context types is a subtype of each one of its component since the former contains more specific information. The extension of a physical context type c to a context type c' corresponds to the process of getting more information. These rules are extensible to any number of context types. Since in type theory, the analogue of a proposition is the judgement we can conclude that the judgement in c is lifted to the judgement in context c'.

Discussion

The logical framework must provide an adequate coverage through the following dimensions:

- partial logic. The logic must be partial to account for expressions which simply lack a value in some contexts. The partiality is obviously an inherent property of DRT since truth values are attached to the presence - or absence - of an element of a set.
- dynamic aspect. The dynamic aspect is revealed at runtime through the context tokens stating what is true within a context at this time.
- non-monotonicity. Roughly speaking, monotonicity indicates that learning a new information cannot reduce the set of known information. During context extension, if the assumptions that hold in the added PC contradict judgments of the initial PC, non-monotonicity occurs. As a consequence, a lifted judgment formally derives the absurd type - $\perp$ -. We face a belief revision process since beliefs have to be changed to accommodate the new belief otherwise inconsistency can occur. An efficient way to cope with non-monotonic situations as suggested in (Boldini 2000), consists either in declaring the extended PC as impossible, in other words to discard the extension, or alternatively, to revise the belief leading to a contradiction in the initial context. Anyway, one PC has to be declared as impossible context.

The Context Ontology

Context modelling abstractions are required to support common understanding and to facilitate communications between programm in distributed environments. A recent approach (Costa *et al.* 2006) has proposed to characterize the concept of context within an ontological framework. The authors introduce some concepts such as the "containment context" which associates multiple entities within a context, the quality requirement in which entities are bearers of qualities - intrinsic moment -, the formal relations between individuals and material relations in which entities share a common property. Despite its interesting definitions of entities and contexts, this conceptual model of context is not sufficient to cover all aspects of context reasoning since the underlying logic is not explicit. Moreover, the *context* concept described as a *moment universal* doesn't formalize the notion of *concept-of.*

The main idea is to substitute the *context* concept with the more atomic concept of *property*. As a result we obtain the Core Ontology described in figure. The kind of possessive, allows to semantically relate the entity noun to a property noun in a specific "*has_<property>*" relation. Through the fundamental categories of *Entity* and *Property*, all the basic components of the physical world are defined. Above these assumptions, the containment context is nothing else than a context extension - sub-context-of - shifted in the logic, the intrinsic properties and the material relations by "*has_<property>*" while formal relations are directly expressed by propositions in the logic. Classical subsumption relations and material relations between entities and their properties explicitly conceptualize in a simple way a given application. The resulting upper level ontology does'nt formalize the context, but rather its elementary components, i.e., the properties. However, the ontological part is not sufficient by itself and must be extended with an appropriate logic. In such a way, a Context Theory can be formulated with a dual model. The first component - Conceptual part - built in the design step defines the domain ontology of the application and follows the construction rules of the upper level ontology. The second component - logical part - addresses the logical aspect relating the concepts defined in the previous layer through the generation of context types including entities and propositions. It is only in this part - also built during the design step - that the context appears in a dynamic way.

From an extended perspective, the context model presents capabilities either for more applied processes or for a more conceptual approach. A practical extension of the model appears through to the Information Flow (IF) paradigm of (John Barwise 1997). The ontological part together with an executive module correspond to the classification of the IF model while the logical part is nothing else than the Theory within the IF model. The IF model serves as a basis for distributed reasoning and has already been applied to ontology alignment. As a consequence, it can extend easily the implementation of the present model in distributed environment under a categorical framework. Furthermore, a more higher level relates to the Standard Upper Ontology - SUO - with the lattice of Theories - LOT - (Kent 2004). It advocates that the present work also offers some facilities for an integration within this highly conceptual model.

Specification

Context types to share knowledge

Usually, ontologies are seen as content theories on object sorts, object properties and their relations. Under this assumption, they provide a sound model for the description of context types. In addition, they give programs the ability to access a shared knowledge across a network. As a consequence, programs can modify their knowledge at runtime. The application ontologies whose restricted validity is limited to the task execution are a suitable tool for the specifications of context types with their relations. A dynamic ontology has been designed including an intentional definition of concepts and their relations with operators allowing for the composition of new concepts and their relations. A main benefit of this approach is to avoid the combinatory explosion inherent to static approaches. The ontological theory will try to exploit the power of the Cyc project (Lenat & Guha 1990). The Knowledge Base consists of terms - the vocabulary of Cyc - and assertions which relate these terms. These assertions include both simple ground assertions and rules. A bundle of assertions sharing a common set of assumptions are known as micro-theories. Micro-theories are arranged into hierarchies in which any micro-theory has free access to its parent assertions. All Cyc concepts are either individuals or collections, each of them having the ability to be instances of a collection -*is_a* -. Cyc doesn't use a predefined inference scheme (e.g., $tell(KB, \sigma)$, in which σ is a given axiom, will return the new KB including σ). The representation of context types in Cyc permits the programs to share a concept base and allows inference and reasoning mechanisms requiring this base. The resulting software architecture is reported on fig. 1.

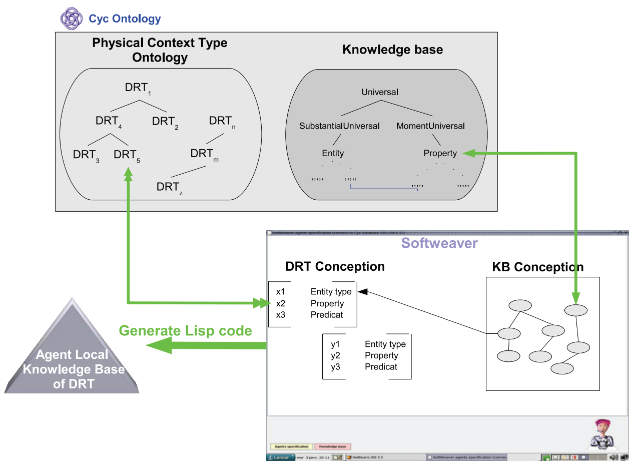

Figure 1: Overview of the Software Architecture

Context types to prove situation

The main idea consists in producing a valid specification through the Cyc tool, and then converting it in a programming language module. For that purpose, a programming language able to implement in an efficient way the logical specifications from the context types is required. The dynamic knowledge composition or change achieved by program is grounded in the context type description incoming from the ontology. The context types defined in a Cyc micro-theory must be understood as the reflexive representation of a program. Reflection is the ability of a program to self-analyze and eventually to modify its internal structure at run-time. The *structural* reflection allows to reify the program code and all abstract data types reachable through this program whereas the *behavioral* reflection allows its self-organization. The *proof-as-program* paradigm has been investigated by the Software Engineering community in order to offer programs in which a sound implementation of specifications is assumed. These programs are extracted from proofs with the Curry-Howard isomorphism as underlying mechanics.

With an ontological model of context types expressed as modules, the reification is possible both during the design

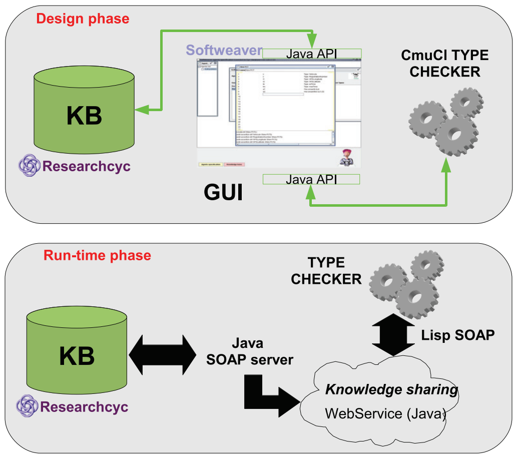

Figure 2: The design and run-time steps

and at run-time (see fig. 2). The selected formalism to represent contexts is used at runtime to check the type of a given situation. This situation describes a set of objects, more precisely entities (fluents) whose type, properties, as well as dependencies are the proof of one or more CRSs (see the code excerpt below).

```
... ... ... ..........................................
(defun get-type-token (label)
   (get-type-set label))
(defun get-type-set-type (label)
   (get-set-type (label)))
(defun get-validity-proposition (proposition-list)
   (eval proposition-list))
(defun equal-type (label1 label2)
   (equal (get-type label1) (get-type label2)))
(defun get-assertion (indice drt-name)
   (gethash indice (get drt-name 'record)))

(defun -> (record-token drt)
   (let ((i(get-indice-record record-token))
      (ii 1)
      (bValue T))
   (while (< ii i)
      (and bValue (if (get-set (get-element-token-at
                                 ii record-token))
         (equal-type (get-element-token-at ii record-token)
                                 (get-element-drt-at ii drt))
         (derivation-proposition (get (get-element-drt-at ii drt)
                                 'proposition) record-token)))
      (+ ii 1 ))
   (bvalue)
   ))
```

Conclusion

Dependent types offer a strong support for the large-scale engineering of systems. CRS are able to pack up data structures, operations over them, and also proofs of the properties of those operations. The increasing power of systems within small-sized components makes this approach attractive for future applications (for instance, it has been shown that current mobile phones have enough power to host a semantic web engine). Softweaver also allows automatic generation of the context model at run-time and incorporation of partial validation in the logic. Moreover, the type-based approach has an immediate computational impact.

Some limitations occur with the specification of types which are static. However, in distributed environments, new types can be acquired from other components. The KB (*ResearchCyc*) is impracticable in the case of hard-constrained applications such as hard real-time systems. There is (actually) no distributed version of the system.

Future works will extend the *Softweaver* tool for distributed environment with a multi-agent architecture, and improve the whole model including goals and actions to cope with web services design. Finally, we plan to provide a NLP-based interface with the user.

References

Anind K., G. D. A. 2000. Towards a better understanding of context and context-awareness. In *Procs. of the CHI 2000 Workshop on The What, Who, Where, When, and How of Context-Awareness*.

Betarte, G. 2000. Type checking dependent (record) types and subtyping. *Journal of Functional and Logic Programming* 10(2):137–166.

Boldini, P. 2000. Formalizing context in intuitionistic type theory. *Fundamenta Informaticae* 42(2):1–23.

Bove, A., and Capretta, V. 2001. Nested general recursion and partiality in type theory. In Boulton, R., and Jackson, P., eds., *Procs. of the 14th Int. Conf. on Theorem Proving in Higher Order Logics*, number 2152 in LNCS, 121–135. Springer.

Buvac, S.; Buvac, V.; and Mason, I. A. 1995. Metamathematics of contexts. *Fundamentae Informaticae* 23(3):412–419.

Buvac, S. 1996. Quantificational logic of context. In *Procs. of the 13th National Conference on Artificial Intelligence*, volume 1, 600–606.

Chen, H.; Finin, T.; and Joshi, A. 2003. Using owl in a pervasive computing broker. In *Procs. of Workshop on Ontologies in Open Agent Systems (AAMAS'03)*.

Coquand, C., and Coquand, T. 1999. Structured type theory. In *Workshop on Logical Frameworks and Metalanguages*.

Costa, P. D.; Almeida, J. P. A.; Pires, L. F.; Guizzardi, G.; and van Sinderen, M. 2006. Towards conceptual foundations for context-aware applications. In *Procs. of the AAAI'06 Workshop on Modeling and Retrieval of Context*, 54–58. AAAI Press.

Dapoigny, R., and Barlatier, P. 2007. Towards a context theory for context-aware systems. In *Procs. of the 2nd IJCAI Workshop on Artificial Intelligence Techniques for Ambient Intelligence*.

Dourish, P. 2004. What we talk about when we talk about context. *Personal and Ubiquitous Computing* 8(1):19–30.

Girard., J.-Y. 1989. *Proofs and Types.* Cambridge University Press.

Giunchiglia, F. 1992. Contextual reasoning. Technical Report 9211-20, Istituto per la Ricerca Scientifica e Technologica.

John Barwise, J. S. 1997. *Information Flow. The logic of Distributed Systems.* Distributed Systems. Cambridge Tracts in Theoretical Computer Science 44.

Kent, R. E. 2004. The information flow framework: A descriptive category metatheory. In *Procs. of the International Category Theory Conference (CT04).*

Kopylov, A. 2003. Dependent intersection: A new way of defining records in type theory. In *Procs. of the 18th An. IEEE Symposium on Logic in Computer Science*, 86–95.

Lenat, D., and Guha, R. V. 1990. *Building Large Knowledge-Based Systems: Represention and Inference in the Cyc Project*. Addison-Wesley, Readinf, MA.

Luo, Z. 1999. Coercive subtyping. *Journal of Logic and Computation* 9(1):105–130.

Martin-Löf, P. 1982. Constructive mathematics and computer programming. *Logic, Methodology and Philosophy of Sciences* 6:153–175.

McCarthy, J. 1993. Notes on formalizing context. In *Procs. of the 13th Int. Joint Conf. on Art. Intelligence*, 555–560.

Mike Uschold, M. G. 1996. Ontologies: Principles, methods and applications. *The Knowledge Engineering Review* 11(2):93–155.

Montague, R. 1970. Pragmatics and intensional logic. *Synthèse* 22:68–94.

Paulson, L. C. 1989. The foundation of a generic theorem prover. *Journal of Automated Reasoning* 5(3):363–397.

Ranta, A. 2004. Grammatical framework: A type-theoretical grammar formalism. *Journal of Functional Programming* 14(2):145–189.

Strang, T., and Linnhoff-Popien, C. 2004. A context modeling survey. In *Sixth International Conference on Ubiquitous Computing (UbiComp2004)*, 34–41.

Thomason, R. H. 1998. Representing and reasoning with context. In *Procs. of the International Conference on Artificial Intelligence and Symbolic Computation*, number 1476 in LNCS, 29–41.

Thomason, R. H. 1999. Type theoretic foundations for context, part 1: Contexts as complex type-theoretic objects. In *CONTEXT*, number 1688 in LNCS, 351–360. Springer.

Villadsen, J. 2004. Multi-dimensional type theory: Rules, categories and combinators for syntax and semantics. In *Int. Workshop on Constraint Solving and Language Processing*, 160–165.

Wang, X. H.; Gu, T.; Zhang, D. Q.; and Pung, H. K. 2004. Ontology based context modeling and reasoning using owl. In *Procs. of the 2nd IEEE Conference on Pervasive Computing and Communications (PerCom2004)*, 18–22.