www.ascens-ist.eu

ASCENS

Autonomic Service-Component Ensembles

D3.1: First Report on WP3

Software Requirements, Knowledge Modeling and Knowledge Representation for Self-Awareness - Report and Survey with Experimental Results for Intelligent Multi-agent Systems

Grant agreement number: **257414**
Funding Scheme: **FET Proactive**
Project Type: **Integrated Project**
Latest version of Annex I: **7.6.2010**

Lead contractor for deliverable: **Lero – University of Limerick**
Author(s): **Emil Vassev (UL), Mike Hinchey (UL), Benoit Gaudin (UL), Pady Nixon (UL), Nicola Bicocchi (UNIMORE), Franco Zambonelli (UNIMORE)**

Due date of deliverable: **September 30, 2011**
Actual submission date: **November 15, 2011**
Revision: **Final**
Classification: **PU**

Project coordinator: **Martin Wirsing (LMU)**
Tel: **+49 89 2180 9154**
Fax: **+49 89 2180 9175**
E-mail: **wirsing@lmu.de**

Partners: **LMU, UNIPI, UDF, Fraunhofer, UJF-Verimag, UNIMORE, ULB, EPFL, VW, Zimory, UL, IMT, Mobsya, CUNI**

SEVENTH FRAMEWORK PROGRAMME

Executive Summary

In the course of the first year of the ASCENS WP3, we studied modern approaches to knowledge representation and knowledge management in intelligent systems. Well employed knowledge helps such systems become aware of situations, recognize states and eventually respond to changes. The great challenge is to decide on the mechanism such systems should use to implement and maintain their knowledge. Our initial assumption is that knowledge representation can be regarded as a formal specification of knowledge data reflecting a system's understanding about itself and its surrounding world. The initial research helped us conclude major requirements needed to be met by KnowLang, a prospective specification language for knowledge representation in ASCENS systems. With KnowLang we provide a development environment that strives to answer fundamental questions related to knowledge representation and reasoning in ASCENS. Knowledge in such systems is structured into knowledge domains, each composed of domain ontology and a logical framework providing knowledge vocabulary and logical foundations used for reasoning. This document presents a "State of the Art" on knowledge representation and awareness along with a formal approach to KnowLang in terms of formal specification layers and parameterization. Moreover, important domain-specific issues are discussed as well.

Contents

Where is the wisdom we have lost in knowledge?
Where is the knowledge we have lost in information?
T.S. Eliot

1 Introduction

This famous quote of the American poet and Nobel laureate T.S. Eliot, cited in his poem "The Rock" in 1934, has become inspiration for many scientists studying knowledge at different levels of depth of meaning and concluding that the concept of intelligence is built upon four fundamental principles: *data*, *information*, *knowledge* and *wisdom*. Similar to human intelligence, knowledge is the source of intelligence in computer-based systems. In general, when we talk about knowledge, we mean facts, understanding, experience and associations. In computer science, all these notions at their very basic level are related to data, which takes the form of measures and notions. Here, the most intriguing question is how to represent this data and what mechanisms and algorithms are needed to derive knowledge from it. It should be noted that the knowledge of a system is represented in a way reflecting our understanding about the problem domain, and we implicitly choose a way to represent the phenomenon we are studying. Our initial assumption is that knowledge representation can be regarded as a formal specification of knowledge data reflecting the system's understanding about itself and its surrounding world. To specify a knowledge representation in ASCENS systems, we are currently developing a special formal language termed KnowLang.

In this document, we report research results obtained in the course of the first year of the ASCENS WP3. The document consists of two parts. In the first part, we outline a "State of the Art" on knowledge representation and awareness, along with a discussion on important issues related to this challenging topic. In the second part, we present KnowLang in terms of specification tiers and parameterization necessary to cover the specification of the ASCENS knowledge domains and reasoning primitives. A simple knowledge representation for the ASCENS robotics case study [BBR$^+$11] is presented as well.

Moreover, note that in the course of this initial year of research, we have published nine peer-reviewed scientific papers, including papers published in the proceedings of international conferences [VH11b, VHGN11, VH11c, VH11a, BCMZ11b, BCMZ11a], two journal papers [VH11d, Vas11a], and a book chapter on "Knowledge Representation" of the new Encyclopedia of Software Engineering, edited by P. A. Laplante [Vas11b].

2 Knowledge Representation and Awareness – State of the Art

Knowledge helps systems achieve awareness and autonomic behavior, where the more knowledgeable systems are, the closer we get to real intelligent systems. Here, the term "knowledge" is widely used in practice assuming rather vague distinctions among data, information, and intelligence. Along with such a context, any discussion on knowledge should refer to those categories. By its nature as a source of intelligence, knowledge allows system to recognize its states and helps to decide how to respond to situations. Pejman Makhfi [Mak11] concludes that the concept of intelligence is built upon four fundamental principles: *data*, *information*, *knowledge*, and *wisdom*. In this quartet, the basic compound for intelligence is data. In general, data takes the form of measures and representations of the internal and external worlds of a system, e.g., raw facts and numbers. Information is derived from data by assigning meaning to it relevant to the domain of interest, e.g., data in a specific context. Knowledge is a specific *interpretation of information* and wisdom is based on *awareness*, *judgment rules* and *principles of constructing new knowledge* from existing one.

2.1 Knowledge and Kinds of Knowledge

There are many kinds of knowledge that need to be considered for the development of intelligent systems. Conceptually, knowledge can be regarded as a large complex aggregation [DR99] composed of constituent parts representing knowledge of different kind. Each kind of knowledge may be used to derive knowledge models of specific domains of interest. For example, in [DR99] the following kinds of knowledge are considered:

- *domain knowledge* - refers to the application domain facts, theories, and heuristics;
- *control knowledge* - describes problem-solving strategies, functional models, etc.;
- *explanatory knowledge* - defines rules and explanations of the system's reasoning process, as well as the way they are generated;
- *system knowledge* - describes data contents and structure, pointers to the implementation of useful algorithms needed to process both data and knowledge, etc. System knowledge also may define user models and strategies for communication with users.

Moreover being considered as essential system and environment information, knowledge may be classified as 1) *internal knowledge* - knowledge about the system itself; and 2) *external knowledge* - knowledge about the system environment. Another knowledge classification could consider *a priori knowledge* (knowledge initially given to a system) and *experience knowledge* (knowledge gained from analysis of tasks performed during the lifetime of a system). Therefore, it depends on the problem domain what kinds of knowledge may be considered and what knowledge models may be derived from those kinds. For example, we may consider knowledge specific to: 1) the internal system component structure and behavior; 2) the system-level structure and behavior; 3) the environment structure and behavior; and 4) different situations where a system component or the system itself might end up in. In addition, knowledge of components' and system's capabilities of communication and integration with other systems might be considered as well.

2.2 Knowledge Representation Techniques

Different knowledge representation techniques might be used to represent different kinds of knowledge and it is our responsibility to pick up ones that suit our needs the most. In general, to build a knowledge model we need specific knowledge elements. The latter may be primitives such as frames, rules, logical expressions, etc. Knowledge primitives might be combined together to represent more complex knowledge elements. A knowledge model may classify knowledge elements by type and group those of the same type into collections. Different approaches to knowledge modeling and knowledge representation have been developed for intelligent systems. Note that it is important to structure the knowledge in such a way that it can be effectively processed by an intelligent system and perceived and updated by humans. The following subsections present some of the most popular approaches to knowledge representation [Gor00].

2.2.1 Rules

Rules can be easily understood by humans. By its nature, rules structure knowledge in the form of attribute-value pairs. In general, rules may take the following form [Gor00]:

```
if attribute A1 has value V1 and attribute A2 has value V2 then
  set attribute A3 to value V3
end if
```

The attribute part of a rule can consist of a series of clauses where the AND, and to a lesser degree OR and NOT, logical connectives are possible. Sometimes, rules might have the form of "IF premise THEN action", where "premise" is Boolean and "action" is a series of statements. Note that although similar, there are some important differences between the rules used for knowledge representation and the conditional statements found in conventional programming languages:

- Rules are relatively independent of one another.
- Rules can be based on heuristics or experiential reasoning.
- Rules can accept uncertainty in the reasoning process.
- Conditional statements combine the knowledge and reasoning in one structure.
- Rules simply represent the knowledge, i.e., the inference mechanism or reasoning is separate.

Shortliffe successfully applied the rule-based approach to the development of systems applying human knowledge and function at the level of a human expert [Sho76]. In this approach, attributes may represent internal data and both system input and output (e.g., a response from the user). In such a knowledge model, it is relatively easy to construct an engine that uses the set of rules in an automated reasoning system. Rules may be dynamic, i.e., they can be archived and updated as necessary.

2.2.2 Frames

Frame-based knowledge models represent simple concepts via a collection of information and associated actions. Often, the notion of frame is related to data structure containing typical knowledge about a concept or physical entity - an object, a person, etc. An example of a simple representation of a person with the frame-based approach is the following [Gor00]:

Frame: Ellery Stone
Specialization of: Frame Person
Date of Birth: 30:04:62
Sex: Male
Nationality: British
Home Town: St. Helens
Occupation: Tailor
Health: (Consult Medical system)

Frames represent knowledge about real-world entities by combining information, calls to information derivation functions and output assignments. A frame contains both descriptions of attributes and procedural details. As shown in the simple frame above, some of the slots have associated values and one slot refers to another system that must be used to find a value. A frame can encompass both semantic and procedural knowledge. In a frame we recognize two key elements: 1) *slots* - sets of attributes for specific entity that is described; and 2) *facets* - extended knowledge about an attribute in a frame. One of the main advantages of frames is the ability to include special *demons* to compute slot values. A demon can run a function that computes the value of a slot on demand.

2.2.3 Semantic Networks, Concept maps and Bayesian networks

The third of the basic approaches to knowledge representation is termed *semantic networks* [Sow92]. The idea behind semantic networks is that knowledge is often best understood as a set of concepts that are related to one another. Basically, a semantic network is a *directed graph* consisting of *nodes* (or vertices) connected with *edges* (or arcs). Nodes represent *concepts* and edges represent *semantic* *relations* between those concepts. There is no standard set of relations between concepts used in semantic networks, but the following relations are very common:

- instance: X is an instance of Y if X is a specific example of the general concept Y;
  Example: *Object A is an instance of Class B*

- isa: X isa Y if X is a subset of the more general concept Y;
  Example: *sparrow isa bird*

- haspart: X haspart Y if the concept Y is a part of the concept X;
  Example: *sparrow haspart tail*

Inheritance is a key notion in semantic networks and can be represented naturally by *isa* relations. Essentially, a computer-based semantic network uses metadata (data describing data) to represent the meaning of different information. Intelligent systems that recognize the meaning of information (for example, data stored in a warehouse) become immeasurably more intelligent. Extensible Markup Language (XML) [W3C11] and Resource Description Framework (RDF) [W3C10] are common content-management schemes that support semantic networks. Concept maps are similar to semantic networks, but they label the links between nodes in very different ways. They are considered more powerful than semantic networks because they can represent fairly complex concepts, such as a hierarchy of concepts with each node constituting a separate concept. Concept maps are useful when designers want to use an intelligent system to adopt a constructivist view of learning.

A special class of semantic networks is the so-called Bayesian networks [Nea03] where implications are used as the primary relation for connecting nodes. A Bayesian network is a directed acyclic graph where the nodes are variables labeled with local probability distributions on the possible node values and edges are used to assert statistical dependency relations between variables. Bayesian networks allow for probabilistic reasoning (see Section 2.2.6). Although useful, they are notoriously difficult to build accurately and efficiently, which has somewhat limited their application to real world problems.

2.2.4 Ontologies

Ontologies inherit the basic concepts provided by rules, frames, semantic networks, and concept maps. They explicitly represent domain concepts, objects, and the relationships among those concepts and objects to form the basic structure around which knowledge can be built [ST99]. The main idea is to establish standard models, taxonomies, vocabularies, and domain terminology and use them to develop appropriate knowledge and reasoning models. An ontology consists of hierarchies of concepts, for example, an "objects" concept tree or a "relations" concept tree. Each concept has properties, which can be regarded as a frame. The relationships among the concepts form semantic networks. Additional rules and constraints might impose restrictions on the relationships or define true statements in the ontology (facts). Figure 1 shows an ontology that represents the concept of a coffee machine. CM has properties such as height, weight, coffee bean hopper, touch screen, container, and so on. A semantic network defines the relationships between CM and the rest of the concepts in the ontology and includes the following properties: CM requires E (electricity), CM requires A (action), CM requires C (coffee), CM requires W (water), and CM makes CD (coffee drink). Some rules expressed with the ontology concepts add new knowledge about the coffee machine.

In general, to build ontology, a special ontology language is required. For example, the Web Ontology Language (OWL) [MH04] is such a formal language that evolved out of Description Logic [BN03] and is the result of research efforts aiming at providing a knowledge representation language for the semantic web. By using OWL, we build an ontology as an explicit and formal specification

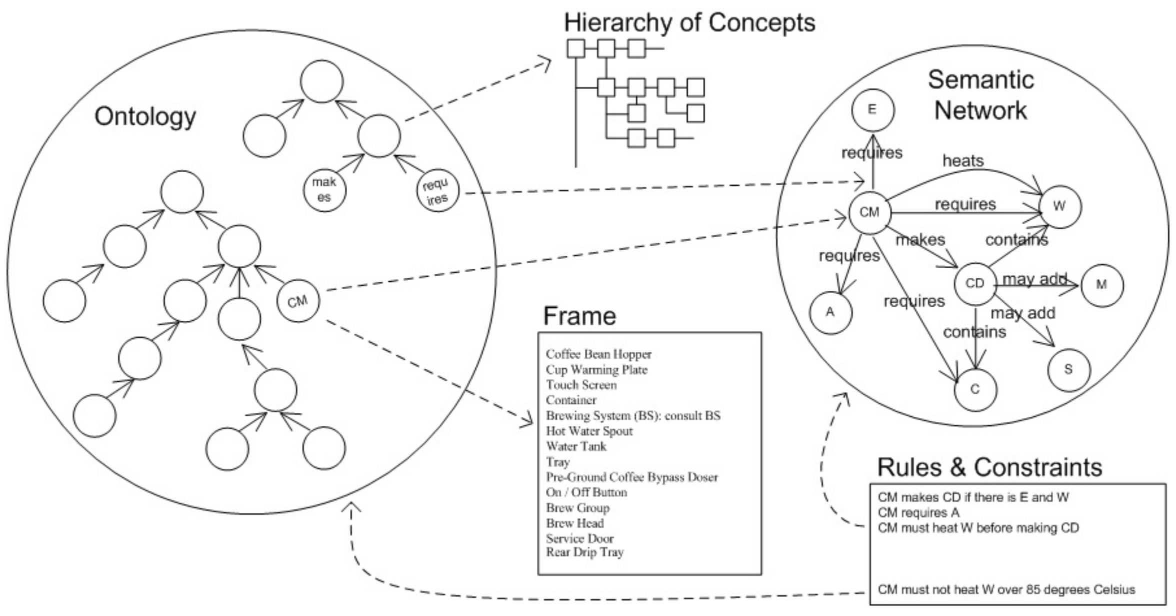

Figure 1: Knowledge Representation Techniques

of a conceptualization that eventually can be compared with a TBox ("T" states for "template" or "terminology") in Description Logic, i.e., it provides a vocabulary of concept definitions and defines relationships among these concepts. The concepts in turn represent knowledge about specific objects in the world.

2.2.5 Logic

To achieve the precise semantics necessary for computational purposes, intelligent system designers often use logic to formalize knowledge representation (KR). Moreover, logic is relevant to reasoning (inferring new knowledge from existing knowledge), which in turn is relevant to entailment and deduction [BL04]. The most prominent logical formalism used for KR is *first-order logic* [BL04]. FOL helps to 1) describe a knowledge domain as consisting of objects; and 2) construct logical formulas around those objects. Similar to semantic networks, statements in natural language can be expressed with logical formulas describing facts about objects using predicate and function symbols. Extensions of FOL such as *second-order logic* (SOL) and *temporal logics* strive to improve the logical formalism by increasing expressiveness. The problem with FOL is that it can quantify over individuals, but not over properties and time - we can thus specify a property's individual components, but not an individual's properties. With SOL, for example, we can axiomatize the sentence "component A and component B have at least one property in common, such as sharing at least one interface", which we can't do with FOL. Temporal logics makes it possible to model knowledge either as linear time or branching time temporal models, and can be used to describe and formalize complex reasoning patterns prescribing inference steps operating over temporal knowledge models. Another prominent formalism is *description logic* (DL) [BN03], which evolved from semantic networks. With DL, we represent an application domain's knowledge by first defining relevant concepts in TBox and then using ABox to specify properties of objects. While less expressive than FOL, DL has a more compact syntax and better computational characteristics.

2.2.6 Probability and Statistics

Decision-making is a complex process that is often based on more than logical conclusions. Probability and statistics may provide for the so-called *probabilistic* and *statistical reasoning* intended to capture uncertain knowledge in which additive probabilities are used to represent degrees of belief of rational agents in the truth of statements. For example, the purpose of a statistical inference might be to draw conclusions about a population based on data obtained from a sample of that population. Probability theory and Bayes' theorem [RB11] lay the basis for such reasoning where Bayesian networks [Nea03] are used to represent belief probability distributions, which actually summarize a potentially infinite set of possible circumstances. The key point is that nodes in a Bayesian network have direct influence on other nodes and given values for some nodes, it is possible to infer the probability distribution for values of other nodes. How a node influences another node is defined by the conditional probability for the nodes usually based on past experience.

For more information on the knowledge representation mechanisms and techniques, please, refer to our publications [Vas11b, VH11d, VH11b].

2.3 Awareness

Awareness is a concept playing a crucial role in intelligent systems. Conceptually, awareness is a product of knowledge representation, knowledge processing and monitoring. Autonomic computing [KC03], one of the modern approaches to knowledge systems, considers two kinds of awareness:

- self-awareness - a system has detailed knowledge about its own entities, current states, capacity and capabilities, physical connections and relations with other systems in its environment;
- context-awareness - a system knows how to negotiate, communicate and interact with environmental systems and how to anticipate environmental system states, situations and changes.

A key success factor for a knowledge system is to employ its knowledge to become an aware system. Such a system must be able to sense and analyze its components and the environment where it operates in. A primary task should be determining the state of each system component and the status of the service-level objectives. Thus, an aware system should be able to notice change and understand the implications of that change. Here, both self-monitoring and monitoring of the environment appear to be one of the key concepts in awareness. A possible approach will be to develop a monitoring system based on notification events. Moreover, an aware system should be able to apply both pattern analysis and pattern recognition to determine normal and abnormal states. Here, knowledge might be expressed in the form of operational patterns, e.g., performance, resource usage, etc. Figure 2 depicts

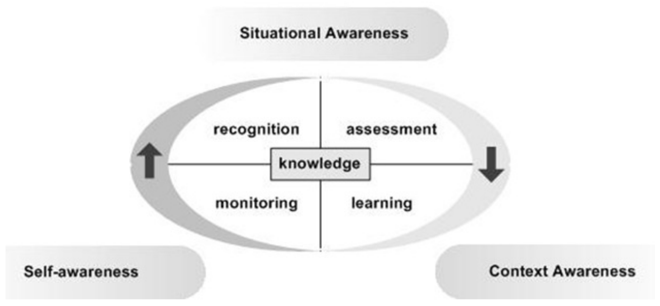

Figure 2: Generic Model for Awareness in Intelligent Systems

a generic model for system awareness based on structured knowledge. As shown, at the core of the awareness model is structured knowledge. This is the knowledge a system maintains in order to be aware of changes taking place in its internal world and in the environment (context knowledge model). Note that system components must have its internal knowledge and possibly have knowledge about the system as a whole and/or the environment. Moreover, a system must maintain special situational knowledge eventually expressed as situational knowledge patterns describing special situations that must be considered by that system. Such situations are relevant changes in the environment or in the system itself. A situation could be a joint situation where changes in different domains of interest are considered. For example, a change in the internal system's structure and a change in the environment might be considered as a situation. The model for system awareness comprises a special awareness control loop where we observe four distinct functions (see Figure 2):

- monitoring - collects, aggregates, filters, manages and reports internal and external details (e.g., metrics and topologies) gathered from the internal entities of a system and from its context;
- recognition - uses knowledge models to recognize changes in the system or in the context;
- assessment - determines entities (internal or external) of interest, generates hypotheses about situations involving these entities, and recognizes situational patterns;
- learning - generates new situational patterns and maintains history of property changes.

Therefore, in order to monitor and collect information consistently and to recognize changes, a system must have a well-structured uniform knowledge model of itself and eventually such of its context. Moreover, it must have situational knowledge patterns that help the system recognize situations. The four functions forming the awareness control loop help a system be aware of internal changes (self-awareness), of external changes (context awareness) and of situations (situational awareness). The awareness control loop might be developed as a Bayesian network [Nea03] component to provide the adaptive functionality and learning the effects of actions in the awareness mechanism.

2.4 Discussion

2.4.1 Encoded Knowledge versus Represented Knowledge

When working on knowledge representation, it is necessary to consider the fact that most of the time the system under development is not 100% knowledge-centered. This means that software engineers and developers may design and encode a big part of the "*a priory*" knowledge (knowledge given to the system before the latter actually runs) in the implemented program classes and routines. In such a case, the knowledge-represented pieces of knowledge (e.g., concepts, relations, rules, etc.) may complement the knowledge codified into those program classes and routines. For example, rules could be based on classes and methods and a substantial concern about the rules organization is how to relate the knowledge expressed with rules to implemented methods and functions. A possible solution is to map concepts and objects to program classes and objects respectively and design rules working on the input (parameters, pre-conditions) and output (results, post-conditions) of implemented methods.

2.4.2 Completeness and Consistency

It should be noted also that an essential assumption when building knowledge models is that such cannot provide a complete picture of the domain of interest. The fundamental reasons are that domain objects often present real things that cannot be described by a finite set of symbolic structures. Moreover, such objects do not exist in isolation, but are included in unlimited sets of encompassing contexts. Therefore, incompleteness shall be considered when developing knowledge models and also the fact that an intelligent system must rely on reasoning to infer missing knowledge.

Another aspect of the knowledge completeness is related to the way a system assumes its operational world. In this regard, we may have:

- Closed World Assumption (CWA) - assumes a complete and closed model of the world;
- Open World Assumption (OWA) - assumes an incomplete and open model of the world.

Whereas the CWA strategy assumes that unless an atomic sentence is known to be true, it can be assumed to be false, the OWA strategy assumes that any information not explicitly specified (or such that cannot be derived from the known data) is considered unknown. Note that FOL imposes CWA semantics and Description Logics impose OWA semantics. The following example shows the difference between these two assumptions:

Given: *Gloria drives Mazda.*
Question: *Does Gloria drive a red Mazda*?
Answer: (CWA) *No.*
(OWA) *Unknown.* (The Gloria's Mazda could be red.)

Although more restrictive, CWA provides for avoiding inconsistency in knowledge - if consistent, knowledge cannot become inconsistent, because CWA does not allow adding new facts, which may lead to inconsistency. Note that knowledge consistency is important for efficient reasoning but is not mandatory in knowledge representation. Other approaches helping a system preserve knowledge consistency are "no negation" and the use of consistency rules and constraints. For example, rules may imply true facts or special relations between the concepts and objects or impose a special semantics for such relations. As part of the knowledge representation, rules may provide special constraints ensuring that the knowledge will be correctly processed by the inferential engines. There could be constraints for knowledge acquisition, knowledge retrieval, knowledge update and knowledge inference.

2.4.3 Knowledge Models

Conceptually, intelligent system knowledge should be represented in a way suitable for machine learning and data mining. Therefore, data must be structured (or classified) to become a source of knowledge. In general, to represent knowledge, we may consider special knowledge models for the single system component (*component knowledge model*), the system as a whole (*system knowledge model*) and the operational environment (*context knowledge model*). In addition, we also may consider specific situations, which may be expressed as special *situational knowledge patterns*. The latter will help a system recognize situations it has been before and eventually detect unknown ones. The following is a brief rational giving some hints about possible knowledge models and techniques we should consider.

There may be different knowledge models, depending on the complexity of the knowledge to be represented. For instance, single components are much less complex entities than the system hosting those components. Thus, it is more reasonable to model knowledge about components (*component knowledge model*) in much more detail than for the system as a whole. Also, because of the architecture of the system in question, it is reasonable to assume that each system component has accurate knowledge of its own situation, while the situation in which a system is would be less certain. For these reasons, we may consider modeling local knowledge of each system component (behavior, properties and policies) with *deterministic models* such as Finite State Machines or Logic Formulae. These models offer methods to accurately determine the current situation of a system component. On the other hand, knowledge about the system or the environment may be encoded into *probabilistic models* such as Bayesian Networks, Neural Network or Support Vector Machines. Although less accurate, these models offer practical ways for a system component to determine what is the most likely current situation of the system it belongs to. Moreover, probabilistic models are suitable for evolution and adaptation in case unplanned situations arise. This is of great interest when the system is facing an unknown situation. The learning capabilities of probabilistic models make it then possible to update knowledge representation with new situations. The information provided by the system component knowledge models will help each system component identify the current situation (expressed in the form of situational knowledge patterns) in which a system is.

2.4.4 Reasoning

When a knowledge system needs to decide on a course of action and there is no explicit knowledge about this, the system must reason. Basically, reasoning is how a system uses its knowledge to figure out what it needs to know from what is already known. There are two main categories of reasoning: 1) *monotonic reasoning* - new facts can only produce additional beliefs; and 2) *non-monotonic* reasoning - new facts will sometimes invalidate previous beliefs. Computations over the represented knowledge are done by *inferential engines* that act to produce new knowledge. Usually, the inferential engines are FOL-based or DL-based. The FOL-based inferential engines (e.g., VAMPIRE, SPASS, E-Theorem Prover, etc.) use algorithms from *automated deduction* dedicated to FOL, such as *theorem proving* and *model building*. Theorem proving can help in finding contradictions or checking for new information. Finite model building can be seen as a complementary inference task to theorem proving, and it often makes sense to use both in parallel. The problem with the FOL-based inference is that the logical entailment for FOL is *semi-decidable*, which means that if the desired conclusion follows from the premises then eventually resolution refutation will find a contradiction. As a result, queries often unavoidably do not terminate. Inference engines based on DL (e.g., Racer, DLDB, etc.) are extremely powerful when reasoning about taxonomic knowledge, since they can discover hidden subsumption relationships amongst classes. However, their expressive power is restricted in order to reduce the computational complexity and to guarantee the *decidability* (on DL are decidable) of their deductive algorithms. Consequently, this restriction prevents *taxonomic reasoning* from being widely applicable to heterogeneous domains (e.g. integer and rational numbers, strings) in practice. It may be considered that ontologies, frames, rules and semantic networks are intended to present distinct pieces of knowledge that are worth being differently represented. Note that an important distinction is between ontological and factual knowledge. Whereas the first is related to the general categories (presented as concepts) and important objects in the domain of interest, the second makes assertions about some specific concepts and objects. This is essential for reasoning based on DL. In addition, the rules and constraints are part of the so-called explicit knowledge suitable for FOL-based reasoning.

2.5 Experimenting with Knowledge Representation and Awareness

To experience the power and expressiveness of the ontology approach to knowledge representation, we created simple robot ontology specified in the Web-Ontology Language (OWL) [MH04] by using the OWL Protege editor [Sta11]. Next, we wrote simple inference algorithms in Prolog where we used the so-called SWI Prolog [SWI11], a Prolog implementation licensed under LGPL, and two Prolog libraries for semantic web data, namely Semweb Library [Jan11] and Thea OWL library [VWM09], as basic mechanisms.

In this exercise, we used the inference engine of Prolog, especially its backtracking mechanism, to answer queries and search for assignments of variables that fulfill certain constraints. For example, if we search for all sensors of a robot we can formulate two constraints (being component of the robot and being a sensor) in Prolog and the backtracking mechanism delivers all individuals that fulfill these constraints. All queries were presented in natural language and as Prolog predicates, e.g., queries such as *"Can robot R execute action A?"*. As Prolog is a logical programming language it fits nicely with our OWL modeling which is based on DL. There are minor differences, most notably the open world assumption in DL vs. the closed world assumption in Prolog.

Situational awareness (see Section 2.3) is a key feature in pervasive systems, such as mobile applications and services, mobile robots, smart buildings, e-vehicles and adaptive traffic management, etc. Aside from the ASCENS project, due to its increasing importance, a large body of recent research in pervasive computing is focusing on situation recognition. The emphasis is on algorithms and tools to identify specific situational aspects such as location, current activity, health status and daily routines from sensor data. A principle goal is to analyze and classify the raw streams of data produced by available sensors, e.g., assign compact and expressive labels to represent a situation. Some experiments targeting awareness were done in the course of the first year of WP3:

- We experimented with the design and implementation of different sources of environmental awareness [BCMZ11b, BCMZ11a]. In particular, sensors enabling the recognition of user activities based of accelerometer values and locations based on inferences on GPS signal features.
- We experimented with the design and implementation of a vision-based situation sensor [BLZ12]. Our work is based on an innovative approach that relies on decomposing an image to its individual entities and then reconstructing a comprehensive classification out of them by using the so-called ConceptNet ontology [Com11].
- We experimented with a framework based on *commonsense knowledge* [1] to integrate and use sensors for multimodal situation identification, so as to improve overall classification accuracy and deal with missing classification label [BCMZ11b, BCMZ11a, BLZ12].

3 KnowLang – Knowledge Representation Language for ASCENS

Knowledge representation can be regarded as a specification of the system's understanding about itself and that of its surrounding world. Thus, a knowledge representation model has its syntax and semantics (facts presenting the knowledge meaning) where both are provided by a special language we use to write knowledge representation. To avoid ambiguity in knowledge facts, such a language should be a formal language that provides mechanisms for verifying the consistency and eventually the correctness of the knowledge representation. In this section, we present an overview of the requirements and an initial model for such a formal language termed *KnowLang*, a language we are currently developing in the course of the ASCENS project.

3.1 KnowLang Requirements

Our initial study of the problem domain, where we emphasized the ASCENS case studies, has concluded that awareness of a SC is about entailing the possession of a complete self-model that encompasses the SC's functional features, execution history, current situation, activities, abilities, services, goals, policies, intentions, etc. In addition, a SC may have partial knowledge of the ensemble it is a member of in terms of common goals, ensemble states, communication mechanisms and interfaces, other SCs, etc. Finally, a SC should have a rather general understanding of its operational environment in terms of concepts, objects, events or situations. All these awareness aspects should be empowered

[1]Commonsense knowledge is the routine knowledge we use in our daily life activities.

by structured knowledge and autonomic reasoning and goal-directed (or utility-directed) planning abilities. Knowledge should be structured into different knowledge domains of interest providing together a complete knowledge model for ASCENS systems.

3.1.1 Knowledge Specification and Storing

In our systematic approach to the problem of knowledge representation, we assume that knowledge representation can be regarded as a formal specification of knowledge data reflecting the system's understanding about itself and its surrounding world. The inherited complexity, stemming from the *autonomic* [VH10, KC03] nature of ASCENS (ASCENS is a class of autonomic systems), necessitates the use of a specification model where knowledge can be presented at different levels of depth of meaning. Thus, KnowLang shall impose an ASSL-like multi-tier specification model [2] [Vas08] where we specify a knowledge model at different knowledge tiers nesting other tiers and so on representing the different levels of depth of meaning. In general, KnowLang should provide constructs not only for the specification of the ASCENS knowledge models, but also constructs for special *Knowledge Base Operators* (KB Operators) and *Inference Primitives*. The KB Operators should provide for a means for *storing*, *updating* and *retrieval/querying* knowledge. To physically store the specified knowledge models, we envision a database-like structure termed *ASCENS Knowledge Base* (AKB). The storage mechanism is going to be provided by the tuple-space mechanism of the KLAIM language [RDNP98], the basis for the SCEL language. Thus, the KB Operators should cope with the KLAIM mechanisms for reading and writing data from and to a KLAIM(SCEL) tuple space. In addition, the Inference Primitives should provide mechanisms for reasoning and knowledge inference. Note that KnowLang should provide syntax and semantics rules for each specification tier.

3.1.2 KnowLang Formalism

It is difficult to determine the formalism of KnowLang at this stage of the work, but an initial requirement is that it should be a declarative language. Going further, we can conclude that the language should be expressive enough to help developers benefit from the knowledge-expressive power of the popular knowledge-representation techniques such as ontologies, concept maps, rules, etc. Considering the current logics used to describe knowledge models (see Section 2.2.5), the underlying formalism for the ontology part should be derived from DL and for the rules, from FOL or its derivations. Thus, the Inference Primitives should cope with the DL and FOL inferential algorithms. For example, formal semantics and reasoning involving ontologies expressed with KnowLang might be achieved by mapping an ontology to DL and using the established and well-studied inference procedures for DL to implement reasoning.

3.1.3 Support to Ontologies

KnowLang should support the specification of ontologies, i.e., it should provide efficient constructs for encoding the ASCENS ontologies. The semantics for this part of KnowLang should be DL-based and open-world. Note that DL imposes open-world semantics, which means that if a statement can neither be proven to be true or false it is not judged as false, but as unknown. In general, ontologies provide a form of explicit representation of domain concepts and the relationships between those concepts to form the basic structure around which knowledge can be built. The main idea is to establish standard models, taxonomies, vocabularies and domain terminology and use those to develop appropriate knowledge and reasoning models. Any ontology is a formal and declarative representation of a

[2]The ASSL formal notation is based on a specification model exposed over hierarchically organized formalization tiers.

knowledge model of some topic or subject area. It provides concepts to be used for expressing knowledge in that subject area. This knowledge encompasses: types of entities, attributes and properties, relations and functions, as well as various constraints.

3.1.4 Logical Framework

A domain-specific ontology gives a formal and declarative representation of a knowledge domain in terms of explicitly described domain concepts, individuals and the relationships between those concepts/individuals. To help developers realize the explicit representation of particular and general factual knowledge, in terms of predicates, names, connectives, quantifiers and identity, KnowLang must provide a sort of *logical framework* assisting in the specification of such knowledge primitives. Such a logical framework should provide computational structures, additional to the domain ontology that basically determine the logical foundations helping a SC reason and infer knowledge. A logical framework should support the specification of facts, rules and constraints - all logically founded and built with ontology terms

3.1.5 Distributed Reasoning

Ideally, an ASCENS system must support distributed reasoning, and thus, KnowLang should provide mechanisms for that. Although, the main reasoning mechanisms will be implied by the Logical Framework and the Inference Primitives, specification of global properties such as *actions*, *events* and *situations* are also intended to provide support for distributed reasoning at the ontology level.

3.1.6 Support to Events and Situations

An important requirement for KnowLang is the need of constructs that give some means for associating event/situation terms with event/situation-describing sentences, so that the described events and situations can be referred to, temporally modified, causally related, or otherwise qualified. The formal semantics for KnowLang situations might be borrowed from the so-called Situation Calculus [Sch90] or eventually a probabilistic extension of the same [MPP$^+$01]. The Situation Calculus is a logic formalism based on SOL and designed for representing dynamic domains. It also, is very appropriate for various sorts of reasoning, including planning. Basically, Situation Calculus represents changing scenarios as a set of SOL formulae where the basic elements are: 1) actions that can be performed in the world; 2) fluents that describe the state of the world; and 3) situations that represent a history of action occurrences. The Situation Calculus has shown to be more expressive than it had been initially assumed with respect to concurrent, extended, and temporally qualified actions as well as causation. Another formalism that is appropriate for KnowLang situations is Event Calculus [KS86].

3.1.7 Support to Policies

To specify the behavior of a SC in important situations, KnowLang shall provide constructs for specifying special Policies. The formal semantics for KnowLang Policies could be borrowed from the semantics of self-managing policies specified in ASSL (Autonomic System Specification Language) [Vas08]. The ASSL Policies specify special self-managing behavior driven by special fluents and actions. A fluent presents a state where the system gets into when special conditions are met. Such conditions are driven by events. If a SC gets into a fluent then the policy behind that fluent is considered active and specific actions are executed.

3.1.8 Experience

KnowLang shall provide appropriate mechanisms and constructs to store and query SC's *experience* of actions' executions. This will help a SC be aware of the execution history of the actions to be executed and eventually compute the success probability for those actions. In that way, a SC may learn (infer new knowledge) not to execute actions that traditionally have low success rate. Experience plays a central role in the learning process. For example, an experience can be observed while the robot is acting and then eventually abstracted and stored in the ASCENS Knowledge Base (AKB). Thus, KnowLang should provide additional constructs allowing for experience abstraction.

In [VHGN11], we discuss some additional requirements related to the swarm robotics case study. Those requirements are case-study specific and show a deductive reasoning that should be performed when we approach the problem of knowledge representation for a particular system.

3.2 ASCENS Knowledge Models

The initial research results based on our awareness study concluded that a SC should have structured knowledge addressing the SC's structure and behavior, the SC Ensemble (SCE) structure and behavior, the environment entities and behavior and situations where that SC or the entire SCE might end up in. Based on these knowledge requirements, we defined four knowledge domains:

- SC knowledge - knowledge about internal configuration, resource usage, content, behavior, services, goals, communication ports, actions, events, metrics, etc.;
- SCE knowledge - knowledge about the whole system, e.g., architecture topology, structure, system-level goals and services, behavior, communication links, public interfaces, system-level events, actions, etc.;
- environment knowledge - parameters and properties of the operational environment, e.g., external systems, concepts, objects, external communication interfaces, integration with other systems, etc.;
- situational knowledge - specific situations, involving one or more SCs and eventually the environment.

These knowledge domains are going to be represented by four distinct knowledge corpuses – SC Knowledge Corpus, SCE Knowledge Corpus, Environment Knowledge Corpus and Situational Knowledge Corpus. Each knowledge corpus is structured into a special domain-specific ontology and a logical framework. The domain-specific ontology gives a formal and declarative representation of the knowledge domain in terms of explicitly described domain concepts, individuals (or objects) and the relationships between those concepts/individuals. The logical framework helps to realize the explicit representation of particular and general factual knowledge, in terms of predicates, names, connectives, quantifiers, and identity. The logical framework provides additional computational structures to determine the logical foundations helping a SC reason and infer knowledge. All four ASCENS knowledge corpuses together form the ASCENS Knowledge Base (AKB). The AKB is a sort of knowledge database where knowledge is stored, retrieved and updated. In addition to the knowledge corpuses, the AKB implies a knowledge-operating mechanism providing for knowledge storing, updating and retrieval/querying. Ideally, we can think of an AKB as a black box whose interface consists of two methods called TELL and ASK. TELL is used to add new sentences to the knowledge base and ASK can be used to query information. Both methods may involve knowledge inference, which requires that the AKB is equipped with a special inference engine (or multiple, co-existing inference engines) that reasons about the information in the AKB for the ultimate purpose of formulating new conclusions, i.e., inferring new knowledge.

3.3 KnowLang Specification Model

We envision KnowLang as a formal language providing a comprehensive specification model addressing all the aspects of an ASCENS Knowledge Corpus and providing formalism sufficient to specify the AKB operators and theories helping the AKB inference mechanism. The complexity of the problem necessitates the use of a specification model where knowledge can be presented at different levels of depth of meaning. Thus, KnowLang imposes a multi-tier specification model (see Figure 3), where we specify the ASCENS knowledge corpuses, KB operators and inference primitives at different hierarchically organized knowledge tiers.

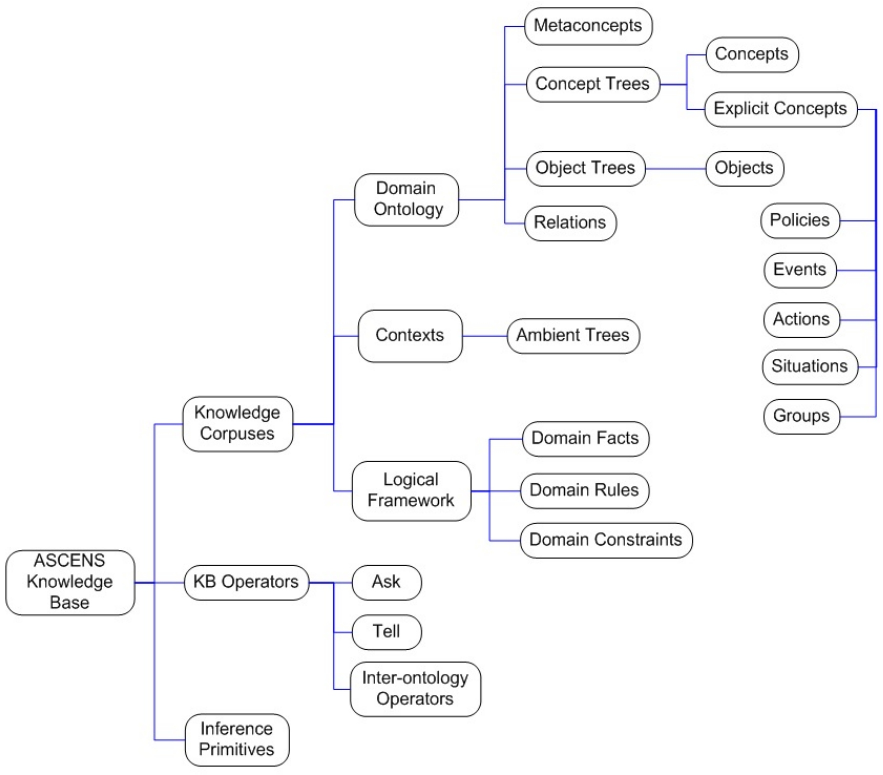

Figure 3: KnowLang Multi-Tier Specification Model

Definitions 1 through 49 (see the definitions following Figure 3) outline a formal representation of the KnowLang specification model. As shown in Definition 1, an ASCENS Knowledge Base is a tuple of three main knowledge components - knowledge corpus (Kc), KB operators (Op) and inference primitives (Ip). A Kc is a tuple of three knowledge components - ontologies (O), contexts (Cx) and logical framework (Lf) (see Definition 2).

FORMAL REPRESENTATION OF KNOWLANG

Def. 1 $Kb = \{Kc, Op, Ip\}$ *(ASCENS Knowledge Base)*

Def. 2 $Kc = \{O, Cx, Lf\}$ *(ASCENS Knowledge Corpus)*

ASCENS ONTOLOGIES

Def. 3 $O = \{o_{sc}, o_{sce}, o_{env}, o_{si}\}$ *(ASCENS Ontologies)*

Def. 4 $o = \{Cm, Ct, Ot, R\}, o \in O$ *(ASCENS Ontology)*

Def. 5 $Cm = \{cm_0, cm_1, ..., cm_n\}$ *(Meta-concepts)*

Def. 6 $cm = \{[cx], i\}, i \in Icx$ *(Meta-concept, cx - Context, i - Interpretation)*

Further, an ASCENS ontology is composed of hierarchically organized sets of meta-concepts (Cm), concept trees (Ct), object trees (Ot) and relations (R) (see Definition 4). Meta-concepts (Cm) provide a context-oriented interpretation (i) (see Definition 6) of concepts. Concept trees (Ct) consist of semantically related concepts (C) and/or explicit concepts (Ce). Every concept tree (ct) has a root concept (tr) because the architecture ultimately must reference a single concept that is the connection point to concepts that are outside the concept tree. A root concept may optionally inherit a meta-concept, which is denoted $[tr \succ cm]$ (see Definition 8). The square brackets [] state for "optional" and $\succ$ is "inherits" relation. Every concept has a set of properties (P) and optional sets of functionalities (F), parent concepts (Pr) and children concepts (Ch) (see Definition 10).

Def. 7 $Ct = \{ct_0, ct_1, ..., ct_n\}$ *(Concept Trees)*

Def. 8 $ct = \{tr, C, [Ce]\}$ *(Concept Tree)*

$tr \in (C \cup Ce), [tr \succ cm]$ (tr - Tree Root)

Def. 9 $C = c_0, c_1, ., c_n$ *(Concepts)*

Def. 10 $c = \{P, [F], [Pr], [Ch]\}$ *(Concept)*

$Pr \subset (C \cup Ce), c \succ Pr$ (Pr - Parents)
$Ch \subset (C \cup Ce), Ch \succ c$ (Ch - Children)

Def. 11 $P = \{p_0, p_1, ..., p_n\}$ *(Properties)*

Def. 12 $F = \{f_0, f_1, ..., f_n\}$ *(Functionalities)*

Explicit concepts are concepts that must be presented in the knowledge representation of an ASCENS system. Explicit concepts are mainly intended to support 1) the autonomic behavior of the SCs; and 2) distributed reasoning and knowledge sharing among the SCs. These concepts might be policies (Π), events (E), actions (A), situations (Si) and groups (Gr) (see Definition 13).

Def. 13 $Ce = \{\Pi, E, A, Si, Gr\}$ *(Explicit Concepts)*

Def. 14 $\Pi = \{\pi_0, \pi_1, ..., \pi_n\}$ *(Policies)*

Def. 15 $\pi = \{g, N_\pi, A_\pi, map(N_\pi, A_\pi)\}$ *(Policy)*

$A_\pi \subset A, N_\pi \rightarrow A_\pi$ (A_π - Policy Actions)
$E_\pi \subset E, E_\pi \subset N_\pi$ (E_π - Policy Events)

Def. 16 $N_\pi = \{n_0, n_1, ..., n_n\}$ *(Policy Conditions)*

Def. 17 $n := bf(O)$ *(Condition - Boolean Statement)*

A policy has a goal (g) and policy conditions (N_π) mapped to policy actions (A_π), where the evaluation of N_π may imply the evaluation of actions (denoted with ($N_\pi \rightarrow A_\pi$) (see Definition 15). A condition is a Boolean function over ontology (see Definition 17). Note that the policy conditions may be expressed with policy events. A goal is a desirable transition from a state to another state (denoted with $s \Rightarrow s'$) (see Definition 18). The system may occupy a state (s) when the properties of an object are updated (denoted with $Tell \rhd ob.P$), the properties of a set of objects get updated, or some events have occurred in the system or in the environment (denoted with $Tell \rhd E_s$) (see Definition 19). Note that $Tell$ is a KB Operator involving knowledge inference (see Definition 46).

Def. 18 $g = (s \Rightarrow s')$ *(Goal)*

Def. 19 $s = \langle Tell \rhd ob.P \rangle | \langle Tell \rhd \{ob_0.P, ob_1.P, ..., ob_n.P\} \rangle | \langle Tell \rhd E_s \rangle$ *(State)*

$E_s \subset E$ (E_s - State Events)

Def. 20 $E = \{e_0, e_1, ..., e_n\}$ *(Events)*

Def. 21 $A = \{a_0, a_1, ..., a_n\}$ *(Actions)*

A situation is expressed with a state (s), a history of actions ($A_{si}^{\leftarrow}$) (actions executed to get to state s), actions A_{si} that can be performed from state s and an optional history of events $E_{si}^{\leftarrow}$ that eventually occurred to get to state s (see Definition 23).

Def. 22 $Si = \{si_0, si_1, ..., si_n\}$ *(Situations)*

Def. 23 $si = \{s, A_{si}^{\leftarrow}, [E_{si}^{\leftarrow}], A_{si}\}$ *(Situation)*

$A_{si}^{\leftarrow} \subset A$ ($A_{si}^{\leftarrow}$ - Executed Actions)
$A_{si} \subset A$ (A_{si} - Possible Actions)
$E_{si}^{\leftarrow} \subset E$ (E_{si} - Situation Events)

Def. 24 $Gr = \{gr_0, gr_1, ..., gr_n\}$ *(Groups)*

Def. 25 $gr = \{Ob_{gr}, R_{gr}\}$ *(Group)*

$Ob_{gr} \subset Ob$ (Ob_{gr} - Group Objects, Ob - Objects)
$R_{gr} \subset R$ (R_{gr} - Group Relations)

Def. 26 $Ot = \{ot_0, ot_1, ..., ot_n\}$ *(Object Trees)*

Def. 27 $ot = \{ob, [Pb]\}$ *(Object Tree)*

Def. 28 $ob = \{instof(c), P\}, ob \subset Ob$ *(Object)*

Def. 29 $Pb := \{ob_0, ob_1, ..., ob_n\}, Pb \subset P$ *(Object Properties)*

Def. 30 $R = \{r_0, r_1, ..., r_n\}$ *(Relations)*

Def. 31 $r = \{c_k, rn, [Z], c_n\} | \{ob_k, rn, [Z], ob_n\}$ *(Relation, rn - name, Z - probability distribution)*

A group involves objects related to each other through a distinct set of relations (see Definition 25). Note that groups are explicit concept intended to (but not restricted) represent knowledge about the SCE structure topology. Object trees (Ot) are conceptualization of how objects existing in the world of interest are related to each other. The relationships are based on the principle that objects have properties, where sometimes the value of a property is another object, which in turn also has properties. Such properties are termed as object properties (Pb). An object tree consists of a root object (ob) and an optional set of object properties (Pb) (see Definition 27). An object (ob) has a set of properties (P) including object properties (Pb) and is an instance of a concept (denoted as $instof(c)$ - see Definition 28). Relations connect two concepts or two objects and may have probability distribution Z (e.g., over time). Probability distribution is provided to support *probabilistic reasoning* (see Section 3.5.4). Note that we consider *binary relations* only.

ASCENS CONTEXTS

Def. 32 $Cx = \{cx_0, cx_1, ..., cx_n\}$ *(Contexts)*

Def. 33 $cx = \{At, [Icx]\}$ *(Context)*

Def. 34 $At = \{at_0, at_1, ..., at_n\}$ *(Ambient Trees)*

Def. 35 $at = \{ct, Ca, [i]\}$ *(Ambient Tree)*

$ct \subset Ct$ (Concept Tree described by an ontology)
$Ca \subset C$ (Ca - Ambient Concepts)
$i \subset Icx$ (i-Ambient Tree Interpretation)

Def. 36 $Icx = \{i_0, i_1, ..., i_n\}$ *(Context Interpretations)*

Contexts are intended to extract the relevant knowledge from an ontology. Moreover, contexts carry interpretation for some of the meta-concepts (see Definition 6), which may lead to a new interpretation of the descendant concepts (derived from a meta-concept - see Definition 8). We consider a very broad notion of context, e.g., the environment in a fraction of time or a generic situation such as currently-ongoing important system function. Thus, a context must emphasize the key concepts in an ontology, which helps the inference mechanism narrow the domain knowledge (domain ontology) by exploring the concept trees down to the emphasized key concepts only. Thus, depending on the context, some low-level concepts might be subsumed by their upper-level parent concepts, just because the former are not relevant to that very context. For example, a robot wheel can be considered as a thing or as an important part of the robot's motion system. As a result, the context interpretation of knowledge will help the system deal with "clean" knowledge and the reasoning shall be more efficient. A context (cx) consists of ambient trees (At) and optional context interpretations (Icx) (see Definition 33). An ambient tree (at) consists of a real concept tree (ct) described by an ASCENS ontology, ambient concepts (Ca) part of the concept tree and optional context interpretation (i).

ASCENS LOGICAL FRAMEWORK

Def. 37 $Lf = \{Fa, Rl, Ct\}$ *(ASCENS Logical Framework)*

Def. 38 $Fa = \{fa_0, fa_1, ..., fa_n\}$ *(Facts)*

Def. 39 $fa = bf(O) \rightarrow \boldsymbol{T}$ *(Fact - True statement over ontology)*

Def. 40 $Rl = \{rl_0, rl_1, ..., rl_n\}$ *(Rules)*

Def. 41 $rl =$ ***if*** fa_1 ***then*** fa_2 ***else*** fa_3 *(Rule)*

Def. 42 $Ct = \{ct_0, ct_1, ..., ct_n\}$ *(Constraints)*

Def. 43 $ct = \langle$***if*** fa_1 ***then*** $MUST\ fa_2\rangle \mid \langle$***if*** fa_1 ***then*** $MUST\ \neg fa_2\rangle$ *(Constraint)*

$fa_1, fa_2 \in Fa$

An ASCENS Logical Framework (Lf) is composed of facts (Fa), rules (Rl) and constraints (Ct) (Definition 37). As shown in Definitions 38 through 43, the Lf's components are built with ontology terms:

- *facts* - define true statements in the ontologies (O);
- *rules* - express knowledge such as: 1) *if H than C*; or 2) *if H than C1 else C2*; where H is hypothesis of the rule and C is the conclusion;
- *constraints* - used to validate knowledge, i.e., to check its consistency. Can be positive or negative and express knowledge of the form:

  1) *if A holds, so must B*;

  2) *if A holds B must not*.

Constraints might be also used as consistency rules helping the knowledge-processing engines check the consistency of a KC (knowledge corpus).

ASCENS KNOWLEDGE BASE OPERATORS

Def. 44 $Op = \{Ask, Tell, Oop\}$ *(ASCENS Knowledge Base Operators)*

Def. 45 $Ask = retrieve(Kc) \rightarrow Ip \lhd Kc$ *(query knowledge base)*

Def. 46 $Tell = update(Kc) \rightarrow Ip \rhd Kc$ *(update knowledge base)*

Def. 47 $Oop = fo(Oi) \rightarrow Ip \rhd Kc,\ Oi \subset O$ *(Inter-ontology Operators)*

ASCENS INFERENCE PRIMITIVES

Def. 48 $Ip = \{ip_0, ip_1, ..., ip_n\}$ *(Inference Primitives)*

Def. 49 $ip = impl(FOL)|impl(FOPL)|impl(DL)$ *(Inference Primitive)*

The *ASCENS Knowledge Base Operators* (*Op*) can be grouped into three groups: Ask operators (retrieve knowledge from a knowledge corpus Kc), $Tell$ operators (update a Kc) and inter-ontology operators (*Oop*) intended to work on one or more ontologies (see Definitions 44 through 47). Such operators can be: *merging*, *mapping*, *alignment*, etc. Note that all the Knowledge Base Operators (*Op*) may imply the use of *inference primitives*, i.e., new knowledge might be produced (inferred) and stored in the KB (see Definitions 45 through 47).

The ASCENS Inference Primitives (Ip) are intended to specify algorithms for reasoning and knowledge inference. The inference algorithms will be based on reasoning algorithms relying on First Order Logic (FOL) [BL04] (and its extensions), First Order Probabilistic Logic (FOPL) [Hal90] and on Description Logics (DL) [BN03]. FOPL increases the power of FOL by allowing us to assert in a natural way "likely" features of objects and concepts via a probability distribution over the possibilities that we envision. Having logics with semantics gives us a notion of deductive entailment. It is our intention to address the following inference techniques inherent in FOL and DL:

- *induction* (FOL) - induct new general knowledge from specific examples;

  Example: *Every robot I know has grippers. $\rightarrow$ Robots have grippers.*

- *deduction* (FOL) - deduct new specific knowledge from more general one;

  Example: *Robots can move. MarXbot is a robot. $\rightarrow$ MarXbot can move.*

- *abduction* (FOPL) - conclude new knowledge based on shared attributes.

  Example: *The object was pulled by a robot.*

  MarXbot has a gripper. $\rightarrow$ MarXbot pulled the object.

- *subsumption* (DL) - the act of subsuming a concept by another concept;

  Example: *Exploit the taxonomy structure of concepts that are defined in the ontology and compute a new taxonomy for a set of concepts or derive matching statement from computed generalization/specialization relationships between task and query.*

- *classification* (DL) - assessing the category a given object belongs to;

- *recognition* (DL) - recognizing an object in the environment.

Note that *uncertainty* is an important issue in abductive reasoning (abduction), which cannot be handled by the traditional FOL, but by FOPL. Abduction is inherently uncertain and may lead to multiple plausible hypotheses, and to find the right one those hypotheses can be ranked by their plausibility (probability) if the latter can be determined. For example, given rules $A \rightarrow B$ and $C \rightarrow B$, and fact B, both A and C are plausible hypotheses and the inference mechanism shall select the one with higher probability.

3.4 Representing Knowledge in Robotic Systems with KnowLang

We have used an initial version of KnowLang to build a simple knowledge representation for the marXbot robotics platform [BBR+11]. Currently, the marXbot robots are able to work in teams where they coordinate based on simple interactions on group tasks. For example, a group of marXbot robots may collectively move a relatively heavy object from point A to point B by using their grippers.

3.4.1 The marXbot Robot Ontologies

To tackle the marXbot knowledge representation problem, an initial structure for the marXbot Robot Ontology (SC Ontology) has been developed. Figure 4 depicts a concept tree ct (see Definition 8) with a tree root tr "Thing". The concept "Thing" is determined by the metaconcept Cm (see Definition 6) "Robot Thing", which carries information about the interpretation of the root concept "Thing" such as "*Thing is anything that can be related to a marXbot robot*". According to this concept tree there are two categories of things in a marXbot robot: *entities* and *virtual entities*, where both are used to organize the vocabulary in the internal robot domain. Note that all the explicit concepts Ce (see Definition 13) are presented as concepts in this concept tree (qualified path: $Thing \rightarrow VirtualEntity \rightarrow Phenomenon$), i.e., in this SC Ontology, the explicit concepts inherit ("$\succ$") the concepts "Phenomenon", "Virtual Entity" and "Thing". In addition to the marXbot SC Ontology, to represent the knowledge in all necessary aspects, we have developed initial variants of the other three ASCENS ontologies - SCE Ontology, Environment Ontology, and Situational Ontology (see Section

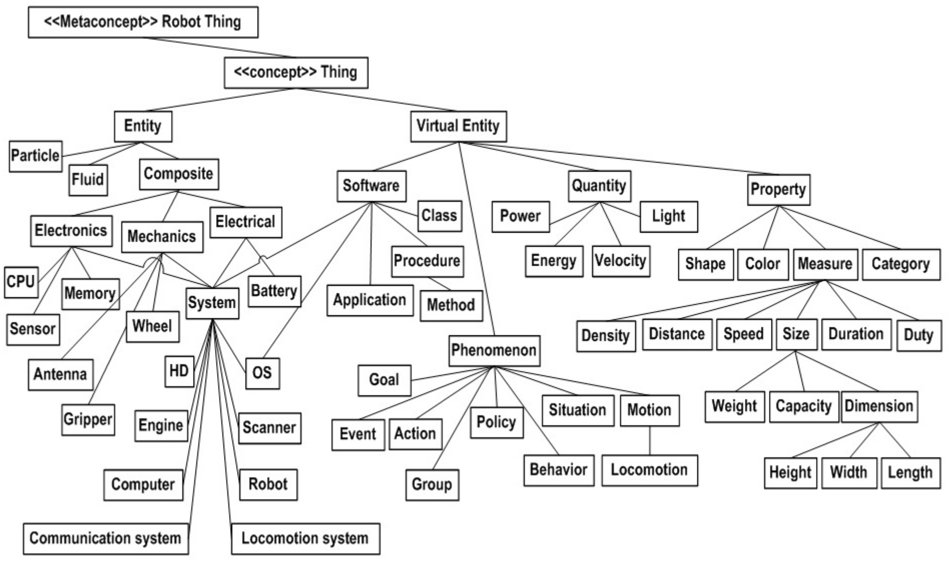

Figure 4: The marXbot Robot SC Ontology: Robot Concept Tree

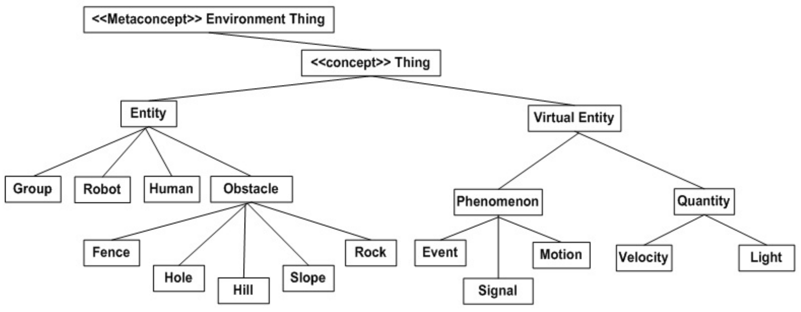

Figure 5: The marXbot Robot Environment Ontology: Environment Concept Tree

3.2). Figure 5 depicts a partial concept tree of the marXbot Robot Environment Ontology. This ontology presents parameters and properties of the robot's operational environment, e.g., external systems (humans, other robots, etc.), concepts (velocity, event, signal, etc.), obstacles, etc. Due to space limitations, the other ontologies with their concept and object trees are not presented here. Figure 6 depicts a partial object tree ot (see Definition 27) of the marXbot Robot SC Ontology. As shown, the Robot Object Tree shows the object properties of the marXbot Robot object. Note that the marXbot Robot SC Ontology contains a few more concept and object trees, such as the Relations Concept Tree not presented here, etc.

3.4.2 The marXbot Robot Contexts and Logical Framework

In specific situations, the robot's inferential engine narrows the scope of knowledge in order to reason more efficiently. This ability is supported by the KnowLang's Context construct cx (see Definition 33). For the purpose of this case study, we have specified a few Contexts, e.g., a context corresponding to the situation ”Robot cannot move”. This Context cx is specified with one single Ambient Tree as following (for more information on the initial KnowLang syntax, please see [VHGN11]):

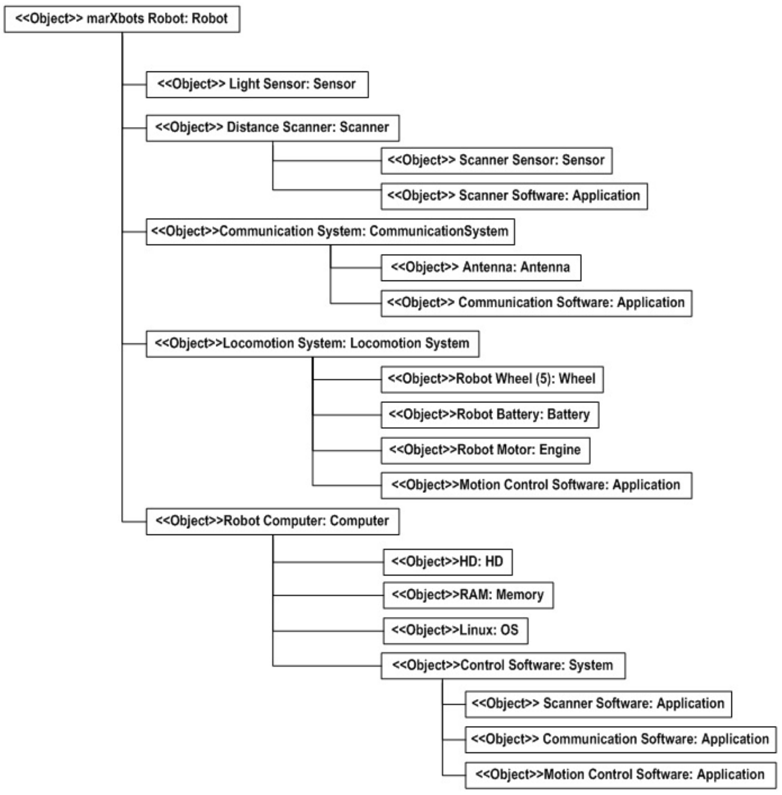
<<Object>> marXbots Robot: Robot
<<Object>> Light Sensor: Sensor
<<Object>> Distance Scanner: Scanner
<<Object>> Scanner Sensor: Sensor
<<Object>> Scanner Software: Application
<<Object>>Communication System: CommunicationSystem
<<Object>> Antenna: Antenna
<<Object>> Communication Software: Application
<<Object>>Locomotion System: Locomotion System
<<Object>>Robot Wheel (5): Wheel
<<Object>>Robot Battery: Battery
<<Object>>Robot Motor: Engine
<<Object>>Motion Control Software: Application
<<Object>>Robot Computer: Computer
<<Object>>HD: HD
<<Object>>RAM: Memory
<<Object>>Linux: OS
<<Object>>Control Software: System
<<Object>> Scanner Software: Application
<<Object>> Communication Software: Application
<<Object>>Motion Control Software: Application

Figure 6: The marXbot Robot SC Ontology: Robot Object Tree

```
CONTEXT name = "Robot cannot move" {
        AMBIENT_TREE {
                ONTOLOGY: Robot;
                CONCEPT_TREE: Thing;
                AMBIENT_CONCEPTS {Electronics, Software, Property}
        }
}
```

This Context is applied automatically at runtime to narrow the scope of knowledge as shown in Figure 7. Note that the ambient concepts define the ”depth” of the concept tree in that specific context, i.e., all the concepts descending from those ambient concepts are generalized (or abstracted) by the ambient concepts. For example, we do not deal with Shape, Color, or Measure anymore, but with Property, because the latter is the ambient concept for the former.

When a robot ends up in such a situation, it should check for possible action determined by policies (specified by one of the robot's ontologies) (see Definition 15) or by rules (specified by the robots' Logical Framework) (see Definition 41). For example, for the purpose of this case study, we have specified two rules as part of the robot's Logical Framework:

```
RULE {
        IF "robot cannot move" THEN {
                DO ACTION "check battery"
        }
}
RULE {
        IF "robot cannot move" AND "battery is charged" THEN {
                DO ACTION "run scanner for obstacles on road"
        }
}
```

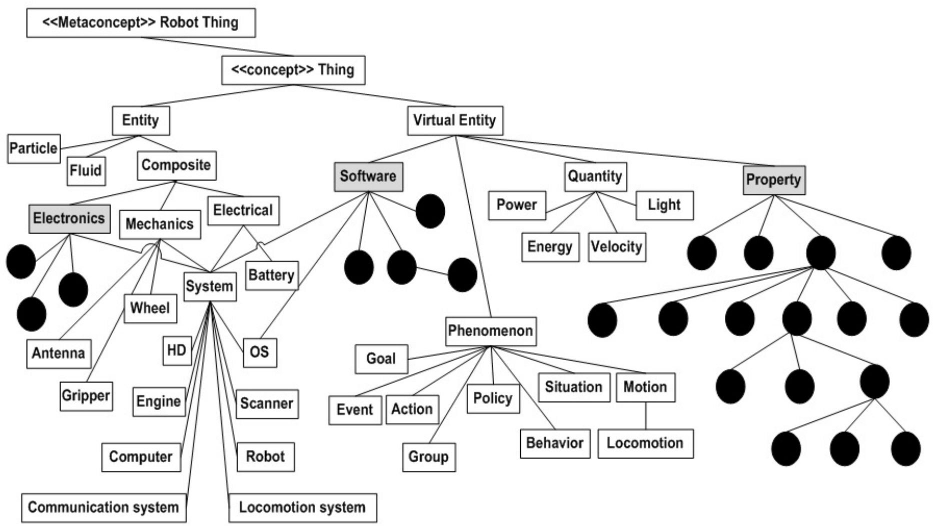

Figure 7: The marXbot Robot Environment Ontology: Narrowed Robot Concept Tree

3.5 Discussion

In comparison with other knowledge representation systems, KnowLang has increased expressive power derived from its layered specification structure and knowledge representation features like:

- *Domain ontologies* - intended to provide domain concepts and the relationships between those concepts to form the basic structure around which knowledge can be built. With constructs like concept trees (see Definitions 8 through 12) and object trees (see Definitions 27 through 29), the KnowLang ontologies establish taxonomies, vocabularies and domain terminology suitable for DL-based reasoning.
- *Meta-concepts* - help ontologies be regarded from different context perspectives by establishing different meanings to some of the key concepts. This is a very powerful construct, which allows defining a sort of converse of a concept and its derived concept tree depending on the current context.
- *Explicit concepts* - allow for distributed reasoning (see Section 3.5.4) and quantification over actions, events, policies, situations and groups.
- *Relations* - the distinct specification layer for relations allow for description of complex general and individual relations applicable to concepts and objects, rather than isolated assertions.
- *Autonomous behavior* - there are specification constructs dedicated to the autonomous behavior of the SCs composing an ASCENS system. Such constructs are policies, actions, events and situations.
- *Quantification* - it is possible to quantify over explicit concepts, concepts and objects.
- *Negation* - KnowLang allows negation, which is a major limitation in systems based on Horn clauses, e.g., Prolog.
- *Contexts and ambient trees* - help to specify what part of the entire knowledge is relevant to a particular situation, which helps to "clean" the knowledge for reasoning.

- *Logical framework* - provides a distinct set of facts, rules and constraints establishing knowledge that complements the ontologies. This knowledge is expressed in FOL and FOPL and is the basis for some of the reasoning algorithms specified at the level of inference primitives (see Definition 49). The presence of both ontologies and logical framework helps the system do hybrid reasoning (see Section 3.5.4), thus making the overall reasoning process more efficient.
- *Inference primitives and KB operators* - provide mechanisms for knowledge inference and knowledge retrieval and update.

The following subsections discuss important questions and challenges related to the KnowLang's features.

3.5.1 Explicit Knowledge

In our approach, different pieces of knowledge shall be presented by different formalism constructs. Ontology, facts, rules and constraints are intended to present distinct pieces of knowledge that are worth being differently represented. Note that an important distinction is between ontological, factual and rule-based knowledge. Whereas the first is related to the general categories (presented as concepts) and important objects in the domain of interest, the second makes assertions about some specific concepts and objects. This distinction is essential for reasoning based on DL. In addition, the rule-based knowledge is part of the so-called explicit knowledge suitable for FOL-based reasoning. The rules may imply special relations between the concepts and objects or impose a special semantics for such relations.

The ASCENS platform does not necessarily emphasize knowledge-centered systems. This means that software engineers using the ASCENS development platform may encode a big part of the "a priory" knowledge (knowledge given to the system before the latter actually runs) in the implemented classes and routines. In such a case, the knowledge-represented pieces of knowledge (e.g., concepts, relations, rules, etc.) may complement the knowledge codified into implemented program classes and routines. For example, rules could be based on SC's classes and methods and a substantial concern about the rules organization is how to relate the knowledge expressed with rules to implemented methods and functions. A possible solution is to map concepts and objects to program classes and objects respectively and design rules working on the input (parameters, pre-conditions) and output (results, post-conditions) of implemented methods. Additional explicit knowledge comes in the form of constraints (see Definition 43), which are intended to provide special rules. There could be constraints for knowledge acquisition, knowledge retrieval, knowledge update and knowledge inference.

3.5.2 Inferred Implicit Knowledge

Explicit knowledge can be used by the system to infer implicit knowledge. The process is based on extracting the knowledge explicitly represented and acquiring the implicit knowledge through augmentation and inference techniques. As mentioned in Section 3.3, the Ask and $Tell$ groups of operators (see Definitions 45 and 46) are intended to query and update the KCs, which may imply inference of new knowledge. One of the challenges we need to overcome is how the system shall differ between the knowledge that is inferred and such that is explicitly stored, e.g., inferred facts versus explicit facts. Here, one of the important questions is "Could knowledge that can be inferred also be available as explicit facts?". Due to its self-learning capabilities, the system shall be able to register new concepts, objects, relations, facts, etc. Therefore, the question is not whether the system needs to store inferred knowledge, but rather how to decide on thresholds determining what part of the inferred knowledge should be stored and when. Note that storing (and thus making explicit) all the inferred knowledge is not a solution, because this will introduce huge redundancy and eventually inconsistency, both having a great impact on reasoning. Our approach to discovering such inferred-explicit knowledge thresholds is to determine: 1) when an inferred piece of knowledge (a concept, an object, a relation, a fact, etc.) called knowledge k is used as a basis to infer a new piece of knowledge called knowledge k'; and 2) check whether knowledge k' is not further used to infer knowledge k (*cycling inference*). Thus, the inferred knowledge should be only one level deep, i.e., a piece of inferred knowledge should be only based on explicit concepts, objects, etc.

3.5.3 Knowledge Consistency

It is important to ensure knowledge consistency. Consistency shall be automatically ensured at runtime to prevent:

- conflicts between rules, between facts, between relations and between constraints, e.g., two rules have the same conditions but different results;
- redundancy in the ontologies and the logical framework, e.g., two rules are applicable to identical situations and conclude the same results;
- rule (constraint) subsumption when two rules conclude the same results, but the first is more restrictive than the second one and whenever the first one succeeds the second one succeeds too.

Constraints may provide for consistency enforcement. For example, a constraint could be one preventing the system from inferring knowledge from inferred knowledge. Other generic constraints can deal with cycling relations and cycling inference. A cycling inference exists when a piece of knowledge is used to infer another piece of knowledge that is consecutively used to infer the first one. Constraints can provide solution to the problem in the form of:

- stop after fixed number of iterations;
- stop when there are no changes in the knowledge.

3.5.4 Reasoning (Inference)

The ASCENS's inference mechanism shall comprise a few high-order inference engines based on FOL and DLs and driven by the inference primitives defined by KnowLang. KnowLang will provide a predefined set of such primitives implementing reasoning based on the inference techniques: induction, deduction, abduction, subsumption, classification and recognition (see Section 3.3). In addition, developers will define their own inference primitives. Supplying the knowledge and reasoning mechanisms to infer the correct decision creates several research challenges:

Hybrid reasoning. The ASCENS knowledge model is highly expressive. The multi-layer structure of knowledge representation together with the multiple inference primitives emphasize hybrid reasoning, i.e., different inferential engines should be used, and their results should be eventually "fused" together.

Distributed reasoning. The SCs (service components) forming an ASCENS system shall be able to reason on their own and share knowledge with other SCs. Common vocabulary formed by the explicit concepts (see Section 3.3) ensures the common interpretation of the shared knowledge.

Reasoning at the conceptual level. Different rules will be used to define links (e.g., hierarchies) within the different concept trees presenting different categories of concepts (e.g., entities, functions, etc.). Correspondingly, different deductions should hold for different categories.

Reasoning over "clean" knowledge. For efficient reasoning, it should be possible to reason by emphasizing on the relevant knowledge and ignore selected parts of the KB. In our approach, contexts (see Definition 33) help to retrieve the context-relevant knowledge and help do deductive reasoning, which would not be otherwise highlighted. Contexts via their ambient concept trees provide a sort of a condensed and explicit/symbolic representation of the world. This representation is cleaned from the overwhelming information that is non-relevant to the context and thus, it provides an efficient model of the world to reason about.

Heuristic reasoning about potential correlations. Groups of concepts or attributes whose names (structures) have terms (components) in common should be suggested as candidates for explicit relationships between them.

Probabilistic reasoning. To allow for probabilistic reasoning, we need to construct a sort of variety model per ontology consisting of a set of variables and a prior probability distribution. To support that, KnowLang allows for the specification of probability distribution for the relations between concepts and objects (see Definition 31). The probability distributions might be associated with time or property values, the properties of the related concepts (or objects) might take. Thus, by specifying relations with probability distributions we actually specify Bayesian networks connecting the concepts and objects of that ontology.

3.6 KnowLang-SCEL and KnowLang-SOTA Relationships

KnowLang is going to be integrated in SCEL, the Service Component Ensembles Language tackled by WP1. Ideally, KnowLang will enrich SCEL with a formal notation, mechanisms and tools dedicated to knowledge representation in ASCENS systems. For compatibility reasons, KnowLang must cope with the SCEL constructs at two levels:

1. The KB Operators, classified as ASK and TELL, should be compatible with the SCEL operators for reading from and writing to the SCEL tuple space. Note that AKB is going to be stored in a SCEL tuple space, where the physical location of the possibly distributed data will be transparent and access to this data will be abstracted via a set of predefined input/output SCEL operators.

2. The domain ontologies specified with KnowLang must incorporate some parts of the SCEL description of the ASCENS system under development. This will help the developers map concepts and objects specified with KnowLang to program structures specified with SCEL. Recall that parts of the so-called “a priory” knowledge could be encoded in the implementation (see Section 3.5.1).

Moreover, KnowLang will be used to model situations and self-adaptation policies determined with SOTA, the State Of The Affairs framework tackled by WP4. SOTA can be used to produce a set of requirements and guidelines for knowledge modeling and representation, which will be then specified with KnowLang.

4 Summary and Future Goals

Knowledge representation is a subject of interest for many research fields. Some of these fields are ontologies, intelligent agents, autonomic computing, pattern recognition, knowledge processing, knowledge discovery and data mining, self-awareness, etc. Conceptually, these research fields recognize that a key success factor for an intelligent system is to employ its knowledge to become an aware system. Such a system must be able to sense and analyze its system components and the environment where it operates.

This document has presented our state-of-the-art vision and initial research results on knowledge representation and awareness in ASCENS (Autonomic Service-Component ENSembles). Such systems need to be developed with initial knowledge and learning capabilities based on knowledge processing and awareness. It is very important how the system knowledge is both structured and modeled to provide essence of both self-awareness and context awareness. In our approach, we consider structured knowledge based on four knowledge models such as SC knowledge, SCE knowledge, SCE environment knowledge and situational knowledge.

In the course of the first year of WP3, we started developing the KnowLang formal language for knowledge representation in ASCENS systems. To provide comprehensive and powerful specification formalism, we propose a special multi-tier specification model, allowing for knowledge representation at different depths of knowledge. The KnowLang specification model defines an AKB (ASCENS Knowledge Base) as composed of special knowledge corpuses, knowledge-base operators and inference primitives. A knowledge corpus is built of a domain ontology, special knowledge-narrowing contexts and a special logical framework providing facts, rules and constraints. The knowledge base operators allow for knowledge retrieval, update and special inter-ontology operations. All these, may rely on the inference primitives, and therefore new knowledge might be inferred. KnowLang is an extension of SCEL (Service Component Ensemble Language) and as such it must maintain compatibility with the SCEL tuple space and its associated input/output operators. Moreover, to integrate encoded knowledge in the AKB, developers should be able to map program's constructs specified with SCEL to concepts and objects specified with KnowLang.

Our plans for the second year of WP3 are mainly concerned with further development of KnowLang. As part of Task T3.1, we shall complete the formal notation and implement appropriate tools for KnowLang, and develop high-level generic knowledge models for SCE systems. Task T3.2 will continue with further implementation of the Reference Model for Self-awareness and Task T3.3 will emphasize investigation of techniques for pattern recognition and data mining.

Published Papers

[BCMZ11a] N. Bicocchi, G. Castelli, M. Mamei, and F. Zambonelli. Augmenting Mobile Localization with Activities and Common Sense Knowledge. In *Proceedings of the International Joint Conference on Ambient Intelligence*. IEEE Computer Society, 2011.

[BCMZ11b] N. Bicocchi, G. Castelli, M. Mamei, and F. Zambonelli. Improving Situation Recognition via Commonsense Sensor Fusion. In *Proceedings of the 1st DEXA Workshop on Information Systems for Situation Awareness and Situation Management*. IEEE Computer Society, 2011.

[BLZ12] N. Bicocchi, M. Lasagni, and F. Zambonelli. Bridging Vision and Commonsense for Multimodal Situation Recognition in Pervasive Systems. In *Submitted to PERCOM 2012 (main track)*. pending acceptance, 2012.

[Vas11a] E. Vassev. Knowledge Representation for Autonomous Systems the ASCENS Case Study. *Dagstuhl Reports*, 1(5):16–17, 2011.

[Vas11b] E. Vassev. Knowledge Representation for Intelligent Systems. In *Encyclopedia of Software Engineering (ed. P. A. Laplante)*. Taylor & Francis, 2011.

[VH11a] E. Vassev and M. Hinchey. Representing Knowledge in Robotic Systems with KnowLang. In *Proceedings of the 1st International ISoLA Workshop on Software Aspects of*

Robotic Systems - Communications in Computer and Information Science. Springer Verlag, Heidelberg, 2011.

[VH11b] E. Vassev and M. Hinchey. Knowledge Representation and Awareness in Autonomic Service-Component Ensembles State of the Art. In *Proceedings of the 14th IEEE International Symposium on Object/Component/Service-oriented Real-time Distributed Computing Workshops (ISORCW2011)*, pages 110–119. IEEE Computer Society, 2011.

[VH11c] E. Vassev and M. Hinchey. Towards a Formal Language for Knowledge Representation in Autonomic Service-Component Ensembles. In *Proceedings of the 3rd International Conference on Data Mining and Intelligent Information Technology Applications (ICMIA2011)*. AICIT, IEEE Xplore, 2011.

[VH11d] E. Vassev and Mike Hinchey. Knowledge Representation and Reasoning for Intelligent Software Systems. *IEEE Computer*, 44(8):96–99, 2011.

[VHGN11] E. Vassev, M. Hinchey, B. Gaudin, and P. Nixon. Requirements and Initial Model for KnowLang - a Language for Knowledge Representation in Autonomic Service-Component Ensembles. In *Proceedings of the Fourth International C* Conference on Computer Science & Software Engineering (C3S2E2011)*, pages 35–42. ACM, 2011.

References

[BBR$^+$11] M. Bonani, T. Baaboura, P. Retornaz, F. Vaussard, S. Magnenat, D. Burnier, V. Longchamp, and F. Mondada. marXbot - Laborotoire de Systemes Robotiques (LSRO), Ecole Polytechnique Federale de Lausanne. http://www.makhfi.com/, 2011.

[BL04] R.J. Brachman and H.J. Levesque. *Knowledge Representation and Reasoning*. Elsevier, New Yorkl, 2004.

[BN03] F. Baader and W. Nutt. Basic Description Logics. In *The Description Logic Handbook (ed. F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, and P.F. Patel-Schneider*, pages 43–95. Cambridge University Press, Cambridge, UK, 2003.

[Com11] Common Sense Computing Initiative - at the MIT Media Lab. ConceptNet. http://csc.media.mit.edu/conceptnet, 2011.

[DR99] V. Devedzic and D. Radovic. A Framework for Building Intelligent Manufacturing Systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C - Applications and Reviews*, 29:422–439, 1999.

[Gor00] J. L. Gordon. Creating Knowledge Maps by Exploiting Dependent Relationships. *Journal of Knowledge-Based Systems, Elsevier Science*, 13:71–79, 2000.

[Hal90] J. Y. Halpern. An Analysis of First-Order Logics of Probability. *Artificial Intelligence*, 46:311–350, 1990.

[Jan11] Jan Wielemaker. SWI-Prolog Semantic Web Library. http://www.swi-prolog.org/pldoc/package/semweb.html, 2011.

[KC03] J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. *IEEE Computer*, 36(1):41–50, 2003.

[KS86] R. Kowalsky and M. Sergot. A Logic-Based Calculus of Events. *New Generation Computing*, 4(1):67–95, 1986.

[Mak11] P. Makhfi. MAKHFI - Methodic Applied Knowledge to Hyper Fictitious Intelligence. http://www.makhfi.com/, 2011.

[MH04] D.L. McGuinness and F. Van Harmelen. OWL Web Ontology Language Overview. *W3C Recommendation. World Wide Web Consortium*, 2004.

[MPP$^+$01] P. Mateus, A. Pacheco, J. Pinto, A. Sernadas, and C. Sernadas. Probabilistic Situation Calculus. *Annals of Mathematics and Artificial Intelligence*, 32(1–4):393–431, 2001.

[Nea03] R. Neapolitan. *Learning Bayesian Networks*. Prentice Hall, 2003.

[RB11] P.N. Robinson and S. Bauer. *Introduction to Bio-Ontologies*. CRC Press, 2011.

[RDNP98] G. L. Ferrari R. De Nicola and R. Pugliese. KLAIM: A Kernel Language for Agents Interaction and Mobility. *IEEE Computer*, 24(5):315–330, 1998.

[Sch90] L. K. Schubert. Monotonic Solution of the Frame Problem in the Situation Calculus: An Efficient Method for Worlds with Fully Specified Actions. In *Knowledge Representation and Defeasible Reasoning*, pages 23–67. Kluwer Academic Press, 1990.

[Sho76] E. H. Shortliffe. *Computer Based Medical Consultations: MYCIN*. Elsevier, New Yorkl, 1976.

[Sow92] J. F. Sowa. Semantic Networks. In *Encyclopedia of Artificial Intelligence (ed. S. C. Shapiro)*. Wiley, New York, 2 edition, 1992.

[ST99] W. Swartout and A. Tate. Ontologies. *IEEE Intelligent Systems*, 14:18–19, 1999.

[Sta11] Stanford Center for Biomedical Informatics Research. Protege. http://protege.stanford.edu/, 2011.

[SWI11] SWI Prolog. SWI-Prolog's Home. http://www.swi-prolog.org/, 2011.

[Vas08] E. Vassev. *Towards a Framework for Specification and Code Generation of Autonomic Systems*. PhD thesis, Computer Science and Software Engineering Department, Concordia University, Quebec, Canada, 2008.

[VH10] E. Vassev and M. Hinchey. The Challenge of Developing Autonomic Systems. *IEEE Computer*, 43(12):93–96, 2010.

[VWM09] V. Vassiliadis, J. Wielemaker, and C. Mungall. Processing OWL2 Ontologies Using Thea: An Application of Logic Programming. In *Proceedings of the the 6th International Workshop on OWL: Experiences and Directions (OWLED 2009)*, 2009.

[W3C10] W3C - Semantic Web. Resource Description Framework (RDF). http://www.w3.org/RDF/, 2010.

[W3C11] W3C - Ubiquitous Web Domain. Extensible Markup Language (XML). http://www.w3.org/XML/, 2011.