Towards a Formal Language for Knowledge Representation in Autonomic Service-Component Ensembles

Emil Vassev
Lero–the Irish Software Engineering Research Centre
University of Limerick
Limerick, Ireland
emil.vassev@lero.ie

Mike Hinchey
Lero–the Irish Software Engineering Research Centre
University of Limerick
Limerick, Ireland
mike.hinchey@lero.ie

***Abstract* — We aim at identifying the content and design principles of KnowLang, a new formal language for knowledge representation in a particular class of autonomic systems termed ASCENS. Autonomic Service-Component Ensembles (ASCENS) are multi-agent systems formed as mobile, intelligent and open-ended swarms of special autonomic service components capable of local and distributed reasoning. Such components encapsulate rules, constraints and mechanisms for self-adaptation and acquire and process knowledge about themselves, other service components, and their environment. 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. We formalize our approach to KnowLang in terms of formal specification layers, syntax, and parameterization required to cover the specification of the ASCENS knowledge domains and reasoning primitives.**

Keywords - knowledge representation; reasoning; ASCENS.

I. INTRODUCTION

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 levels are related to data. Hence, the basic building block of intelligence in computer-based systems is *data* [1], which takes the form of measures and representations of the internal and external worlds of a system, e.g., raw facts and numbers. When regarded in a specific context (domain of interest), data can be assigned relevant meaning to become *information*. Consecutively, knowledge is a specific interpretation of information, i.e., knowledge is created and organized by flows of information interpreted and shaped by the intelligent system. Here the most intriguing question is how to represent the 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. In this paper, we present our understanding about how we shall represent the knowledge in a particular class of intelligent systems termed Autonomic Service-Component Ensembles (or ASCENS) [2]. 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 paper, we present the language in terms of specification tiers and parameterization necessary to cover the specification of the ASCENS knowledge domains and reasoning primitives.

The rest of this paper is organized as follows: Section II introduces the notion of knowledge together with common knowledge representation techniques and inference engines. In Section III, we briefly present the ASCENS concept and our approach to knowledge representation for ASCENS systems. In Section IV, we present in a formal way the KnowLang language - our target language for specifying knowledge in ASCENS systems. Section V presents a discussion on important questions and challenges related to the KnowLang's features presented in Section IV. Finally in Section VI, we provide brief concluding remarks and a summary of our future research goals.

II. BACKGROUND

Conceptually, knowledge can be regarded as a large complex aggregation [3] 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 [3] 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).

There are different knowledge representation techniques that might be used to represent different kinds of knowledge and it is our responsibility to choose or create a technique that suits 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. Typical knowledge representation techniques are *rules*, *frames*, *semantic networks*, *concept diagrams*, *ontologies* and *logics* [4, 5]. Actually logics are used to formalise the knowledge representation techniques, which gives them a precise semantics. *Knowledge-based systems* integrate knowledge via knowledge representation techniques to build a computational model of some domain of interest in which symbols serve as *knowledge surrogates* for real world domain artefacts, such as physical objects, events, relationships, etc. The *domain of interest* can cover any part of the real world or any hypothetical system about which one desires to represent knowledge for computational purposes. Computations over represented knowledge are done by the so-called *inference engine* (or *inferential engine*) that acts on the knowledge facts to produce other facts that may need to be added to the knowledge base (KB). For example, if the KB contains *rules*, the inference engine may chain them either forward (e.g., for forecast) or backward (e.g., for diagnosis). The inference engines are logic-based, e.g., First Order Logic (FOL) and Description Logics (DL) [5, 6].

One way to implement inference is using 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. Some common FOL-based inference engines are VAMPIRE [7], SPASS [8], and the E theorem prover [9]. 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 Description Logics (DLs) (e.g., Racer [10], DLDB [11], 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* (DLs 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.

As we have seen, the existing inference mechanisms are far from being efficient, which is partially due to the very challenging task of knowledge representation.

III. ASCENS AND ASCENS KNOWLEDGE BASE

ASCENS is an FP7 (Seventh Framework Program) [12] project targeting the development of a coherent and integrated set of methods and tools providing a comprehensive development approach to developing *ensembles* (or swarms) of intelligent, autonomous, self-aware and adaptive *service components* (SC). Note that it is of major importance for an ASCENS system to acquire and structure comprehensive knowledge in such a way that it can be effectively and efficiently processed, so such a system becomes aware of itself and its environment. Our initial research on knowledge representation for ASCENS systems [4, 13] concluded that a SC should have structured knowledge addressing the SC's structure and behaviour, the SC Ensemble's (SCE) structure and behaviour, the environment entities and behaviour and situations where that SC or the entire SCE might end up in. Based on these considerations, we defined four *knowledge domains* in ASCENS [4]:

- *SC knowledge* – knowledge about internal configuration, resource usage, content, behaviour, 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, behaviour, 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* [14] 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.

IV. KNOWLANG

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 1), where we specify the ASCENS knowledge corpuses, KB operators and inference primitives at different hieratically organized *knowledge tiers*.

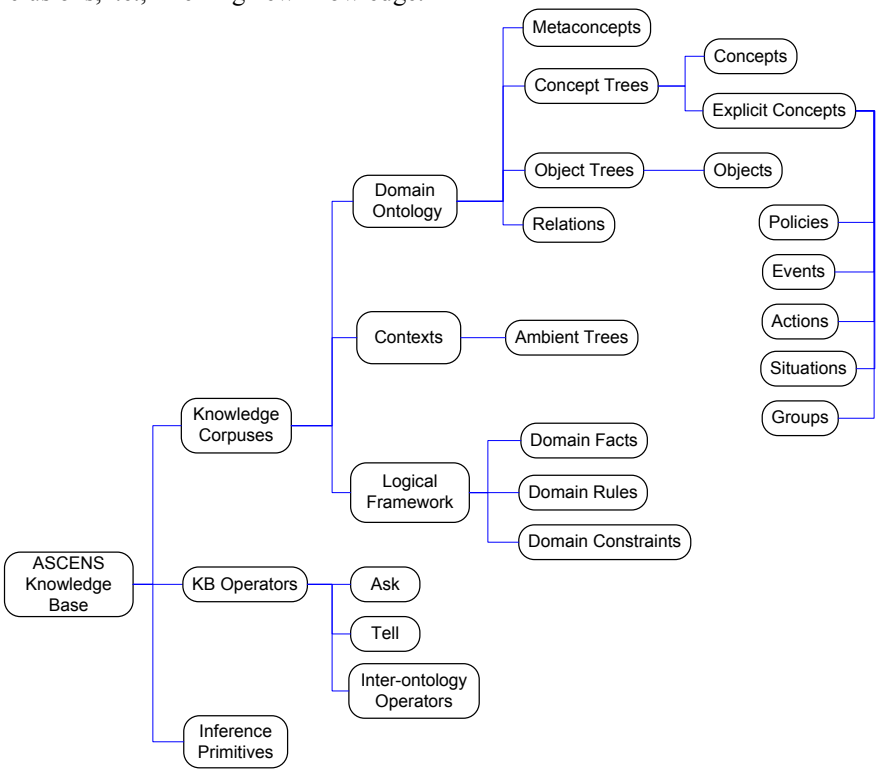

Fig.1 KnowLang multi-tier specification model

Definitions 1 through 49 (see the definitions following Figure 1) 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).

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 $[tr \twoheadrightarrow cm]$ (see Definition 8). The square brackets "[]" state for "optional" and "↠" 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).

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 behaviour 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).

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 ($N_\pi \longrightarrow A_\pi$) (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*.

FORMAL REPRESENTATION OF KNOWLANG

$Kb := \{Kc,\ Op,\ Ip\}$	(*ASCENS Knowledge Base*)	(1)
$Kc := \{O,\ Cx,\ Lf\}$	(*ASCENS Knowledge Corpus*)	(2)

ASCENS ONTOLOGIES

$O := \{o_{sc}, o_{sce}, o_{env}, o_{si}\}$	(*ASCENS Ontologies*)	(3)
$o := \{Cm,\ Ct, Ot, R\}$	(*ASCENS Ontology*)	(4)
$o \in O$		
$Cm := \{cm_0, cm_1, \dots, cm_n\}$	(*Meta-concepts*)	(5)
$cm := \{[cx],\ i\}$	(*Meta-concept, cx – Context*)	(6)
$i \in Icx$	(i *– Interpretation*)	
$Ct := \{ct_0,\ ct_1, \dots, ct_n\}$	(*Concept Trees*)	(7)
$ct := \{tr,\ C,\ [Ce]\}$	(*Concept Tree*)	(8)
$tr \in (C \cup Ce), [tr \twoheadrightarrow cm]$	(tr *– Tree Root*)	
$C := \{c_0,\ c_1, \dots, c_n\}$	(*Concepts*)	(9)
$c := \{P, [F], [Pr], [Ch]\}$	(*Concept*)	(10)
$Pr \subset (C \cup Ce),\ c \twoheadrightarrow Pr$	(Pr *– Parents*)	
$Ch \subset (C \cup Ce),\ c \twoheadleftarrow Ch$	(Ch *– Children*)	
$P := \{p_0, p_1, \dots, p_n\}$	(*Properties*)	(11)
$F := \{f_0, f_1, \dots, f_n\}$	(*Functionalities*)	(12)
$Ce := \{\Pi, E, A, Si, Gr\}$	(*Explicit Concepts*)	(13)
$\Pi := \{\pi_0, \pi_1, \dots, \pi_n\}$	(*Policies*)	(14)
$\pi := \{g,\ N_\pi,\ A_\pi,\ map(N_\pi,\ A_\pi)\}$	(*Policy*)	(15)
$A_\pi \subset A,\ N_\pi \longrightarrow A_\pi$	(A_π *– Policy Actions*)	
$E_\pi \subset E, E_\pi \subset N_\pi$	(E_π *– Policy Events*)	
$N_\pi := \{n_0, n_1, \dots, n_n\}$	(*Policy Conditions*)	(16)
$n := bf(O)$	(*Condition – Boolean Statement*)	(17)
$g := (s \Longrightarrow s')$	(*Goal*)	(18)
$s := \langle Tell \rhd ob.P\rangle \mid \langle Tell \rhd \{ob_0.P, ob_1.P, \dots, ob_n.P\}\rangle \mid \langle Tell \rhd E_s\rangle$ (*State*)		(19)
$E_s \subset E$	(E_s *– State Events*)	
$E := \{e_0, e_1, \dots, e_n\}$	(*Events*)	(20)
$A := \{a_0, a_1, \dots, a_n\}$	(*Actions*)	(21)
$Si := \{si_0, si_1, \dots, si_n\}$	(*Situations*)	(22)
$si := \{s, A_{si}^{\leftarrow}, [E_{si}^{\leftarrow}],\ A_{si}\}$	(*Situation*)	(23)
$A_{si}^{\leftarrow} \subset A$	($A_{si}^{\leftarrow}$ *– Executed Actions*)	
$A_{si} \subset A$	(A_{si} *– Possible Actions*)	
$E_{si}^{\leftarrow} \subset E$	(E_{si} *– Situation Events*)	
$Gr := \{gr_0, gr_1, \dots, gr_n\}$	(*Groups*)	(24)
$gr := \{Ob_{gr}, R_{gr}\}$	(*Group*)	(25)
$Ob_{gr} \subset Ob$	(Ob_{gr} *– Group Objects*, Ob *– Objects*)	
$R_{gr} \subset R$	(R_{gr} *– Group Relations*)	
$Ot := \{ot_0, ot_1, \dots, ot_n\}$	(*Object Trees*)	(26)
$ot := \{ob, [Pb]\}$	(*Object Tree*)	(27)
$ob := \{instof(c), P\}$	(*Object*)	(28)
$ob \in Ob$		
$Pb := \{ob_0, ob_1, \dots, ob_n\}$	(*Object Properties*)	(29)
$Pb \subset P$		
$R := \{r_0, r_1, \dots, r_n\}$	(*Relations*)	(30)
$r := \{c_k, rn, c_n\} \mid \{ob_k, rn, ob_n\}$	(*Relation,* rn *– relation name*)	(31)

A *goal* is a desirable transition from a *state* to another *state* (denoted with $s \Longrightarrow s'$) (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).

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).

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 (Pb) 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. Note that we consider *binary relations* only.

ASCENS CONTEXTS

$Cx := \{cx_0, cx_1, \ldots, cx_n\}$ (*Contexts*) (32)

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

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

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

$ct \in Ct$ ($Concept\ Tree\ described\ by\ an\ ontology$)

$Ca \subset C$ ($Ca - Ambient\ Concepts$)

$i \in Icx$ ($i - Ambient\ Tree\ Interpretation$)

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

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 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 for 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

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

$Fa := \{fa_0, fa_1, \ldots, fa_n\}$ (*Facts*) (38)

$fa := bf(O) \longrightarrow \mathbf{T}$ (*Fact – True statement over ontology*) (39)

$Rl := \{rl_0, rl_1, \ldots, rl_n\}$ (*Rules*) (40)

$rl := if\ \ fa_1\ then\ fa_2\ [else\ fa_3]$ (*Rule*) (41)

$Ct := \{ct_0, ct_1, \ldots, ct_n\}$ (*Constraints*) (42)

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

$fa_1, fa_2 \in Fa$

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 are rather consistency rules helping the knowledge-processing engines check the consistency of a KC (knowledge corpus).

ASCENS KNOWLEDGE BASE OPERATORS

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

$Ask := retrieve\ (Kc) \longrightarrow Ip\ \lhd\ Kc$ (*query knowledge base*) (45)

$Tell := update\ (Kc) \longrightarrow Ip\ \rhd Kc$ (*update knowledge base*) (46)

$Oop := fo\ (Oi\ ) \longrightarrow Ip\ \rhd Kc$ (*Inter-ontology Operators*) (47)

$Oi\ \subset O$

ASCENS INFERENCE PRIMITIVES

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

$ip := impl\ (FOL)\ |\ impl\ (FOPL)\ |\ impl\ (DL)$ (*Inference Primitive*) (49)

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) [5] (and its extensions), First Order Probabilistic Logic (FOPL) [15] and on Description Logics (DL) [6]. 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.* → *Robots have grippers.*
- *deduction* (FOL) – deduct new specific knowledge from more general one;
  Example: *Robots can move. MarXbot is a robot.* → *MarXbot can move.*
- *abduction* (FOPL) – conclude new knowledge based on shared attributes.
  Example: *The object was pulled by a robot.*
  MarXbot has a gripper. → *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 to which 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 $A \longrightarrow B$ and $C \longrightarrow B$, and fact B, both A and C are plausible hypotheses and the inference mechanism shall pick up the one with higher probability.

V. 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 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 instances (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 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 sections discuss important questions and challenges related to the KnowLang's features.

A. 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*.

B. 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 4, 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.

C. Knowledge Consistency

It is important to ensure knowledge. 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 as well.

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.

D. 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*, *subsupmtion*, *classification* and *recognition* (see Section IV). 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 IV) 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.

VI. Conclusion and Future Work

As part of a major international European project, we are currently developing the KnowLang formal language for knowledge representation in a special class of autonomous systems termed as ASCENS. 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 as composed of a special *knowledge corpus*, *knowledge base operators* and *inference primitives*. The 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 relay on the inference primitives, and therefore new knowledge might be inferred.

Our plans for future work are mainly concerned with further and complete development of KnowLang including a toolset for formal validation. Once implemented, KnowLang will be used to specify the knowledge representation for the three ASCENS case studies.

Acknowledgement

This work was supported in part by Science Foundation Ireland grant 03/CE2/I303_1 to Lero—the Irish Software Engineering Research Centre and ASCENS.

References

[1] P. Makhfi, *MAKHFI - Methodic Applied Knowledge to Hyper Fictitious Intelligence* , 2008, http://www.makhfi. com/.

[2] *ASCENS – Autonomic Service-Component Ensembles*. 2010. http://www.ascens-ist.eu/.

[3] 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, 1999, pp. 422–439.

[4] E. Vassev and M. Hinchey, "Knowledge Represen-tation 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*, IEEE Computer Society, 2011, pp. 110–119.

[5] R. J. Brachman and H. J. Levesque, *Knowledge representation and reasoning*, Elsevier, San Francisco, 2004.

[6] F. Baader and W. Nutt, "Basic Description Logics", in *The Description Logic Handbook*, F. Baader, D. Calvanese D. McGuinness, D. Nardi and P. Patel-Schneider (ed.), 2002.

[7] A. Riazanov, *Implementing an Efficient Theorem Prover*, Ph.D. Dissertation, University of Manchester, 2003.

[8] C. Weidenbach, *Handbook of Automated Reasoning,* Elsevier, Chapter SPASS: Combining superposition, Sorts and Splitting, 1999.

[9] S. Schulz, "E - a brainiac theorem prover", *Journal of AI Communications* 15(2), 2002, pp. 111–126.

[10] RacerPro 2.0, http://www.racer-systems.com.

[11] Y. Guo, J. Heflin, and Z. Pan, "Benchmarking DAML+OIL repositories", in *Proceedings of the Second Int. Semantic Web Conf. (ISWC 2003)*, 2870 in LNCS, Springer Verlag, 2003, pp. 613–627.

[12] European Commission – CORDIS, *Seventh Framework Program (FP7)*, http://cordis.europa.eu/fp7/home_en.html.

[13] 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 (C3S2E 2011)*, ACM, 2011, pp. 35–42.

[14] W. Swartout and A. Tate, "Ontologiess", *IEEE Intelligent Systems*, 14 (1999), 1999, pp. 18–19.

[15] J. Y. Halpern, "An analysis of first-order logics of probability", *Artificial Intelligence*, 46, 1990, pp. 311–350.