Implementing Artificial Awareness with KnowLang

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*—To become interaction-aware, an autonomic cyber-physical system needs to be aware of its physical environment and whereabouts and its current internal status. This ability is defined as artificial awareness and it helps intelligent software-intensive systems perceive changes, draw inferences for their own behavior and react. Originally, artificial awareness depends on the knowledge we transfer to a system and how we make the system use that knowledge, so it can exhibit intelligence. Artificial awareness requires a means of sensing changes, so the external and internal worlds can be perceived through their raw events and data. To build an efficient awareness mechanism, we need to provide a means of monitoring and knowledge representation along with a proper reasoner deriving awareness conclusions. In this paper, we present an approach to implementing artificial awareness with KnowLang, a special framework for knowledge representation and reasoning. KnowLang provides for a special knowledge context and a special reasoner operating in that context. The reasoner communicates with the host system via special ASK and TELL operators allowing for awareness conclusions and updates.**

Keywords—awareness; knowledge representation; reasoning; KnowLang

I. INTRODUCTION

In general, an autonomic cyber-physical system engages in various interactions where it is not just able to interact with its operational environment, but also to perceive important structural and dynamic aspects of the same. To become interaction-aware, such a system needs to be aware of its physical environment and whereabouts and its current internal status. This ability is defined as *artificial awareness* and it helps intelligent software-intensive systems perceive changes, draw inferences for their own behavior and react. Originally, artificial awareness depends on the knowledge we transfer to a system and make it use that knowledge, so it can exhibit intelligence. Artificial awareness also requires a means of sensing changes, so the external and internal worlds can be perceived through their raw events and data. To build an efficient awareness mechanism, we need to provide a means of monitoring and knowledge representation along with a proper reasoner deriving awareness conclusions.

In this paper, we present an approach to implementing artificial awareness with KnowLang, a special framework for knowledge representation and reasoning. KnowLang provides for a special knowledge context and a special reasoner operating in that context. The reasoner communicates with the host system via special ASK and TELL operators allowing for knowledge queries and updates. Whereas TELL Operators feed the knowledge context with important information driven by errors, executed actions, new sensory data, etc., ASK Operators provide the system with awareness-based conclusions about the current state of the system and environment and with behavior models for self-adaptation.

The rest of this paper is organized as follows. Section II briefly presents related work and Section III introduces the KnowLang framework. Section IV presents our approach to modeling artificial awareness. In Section V, we present how we implement artificial awareness based on knowledge structuring and reasoning. Section V also presents a brief, proof-of-concept case study. Section VI discuses some of the remarkable challenges KnowLang must overcome. Finally, Section VII concludes this paper with brief concluding remarks and future work.

II. RELATED WORK

The work that is most similar in spirit to our own is that on *context-aware* and *adaptive software systems*. Context-aware and adaptive systems are "systems that are able to adapt their operations to context changes without explicit user intervention" [1]. Their aim is to increase the usability and the effectiveness of the software systems by taking into account the environmental context. To be successful, the development of context-aware and adaptive systems needs to consider both the context awareness and self-adaptation perspectives in a holistic manner. However, current research pays attention to either the self-adaptation or the context-awareness. Research on context-aware systems [2, 3] concentrates on modeling the context information by processing the low level retrieved data to infer high level context information used for sensing and processing operations. However, the context-awareness is controlled by operations hardcoded into the functional system. Consequently, the system ability is limited to cope with a fixed set of context changes that has been built in during the system design time.

A number of frameworks have proposed to facilitate the development of self-adaptive software systems [4]. These frameworks consider the four operations of the control loop (monitoring, analyzing, deciding, and acting) to make the system able to change its structure and/or behavior in response to context/requirements changes. However, they provide a rather static context processing where the system environment context is considered implicitly. This makes not efficient the reasoning on the context data to infer context information.

KnowLang is extremely expressive and provides for explicit context representation via ontology's structures and explicit situations. In addition, it provides an approach to modeling self-adaptive behavior where context-awareness is integrated along with Bayesian network probability distributions, thus allowing for self-adaptation based on both logical and statistical reasoning. Finally, the approach allows for gradually-accurate artificial awareness and self-learning.

III. KNOWLANG

Developing intelligent systems with *knowledge representation and reasoning* (KR&R) has been an increasingly interesting topic for years. Examples are found in semantic mapping [5], improving planning and control aspects [6], and most notably HRI systems [7, 8]. Overall, KR&R strives to solve complex problems where the operational environment is non-deterministic and a system needs to reason at runtime to find missing answers.

KnowLang [9, 10] is a framework for KR&R that aims at efficient and comprehensive knowledge structuring and awareness based on logical and statistical reasoning. It helps us tackle: 1) explicit representation of domain concepts and relationships; 2) explicit representation of particular and general factual knowledge, in terms of predicates, names, connectives, quantifiers and identity; and 3) uncertain knowledge in which additive probabilities are used to represent degrees of belief. Other remarkable features of the framework are related to knowledge cleaning (allowing for efficient reasoning) and knowledge representation for autonomic self-adaptive behavior. Knowledge specified with KnowLang takes the form of a Knowledge Base that outlines a KR context. A special KnowLang Reasoner operates in this context to allow for knowledge querying and update. In addition, the reasoner can infer special self-adaptive behavior based on awareness. To specify knowledge with KnowLang you need to think about 1) domain concepts and their properties and functionalities (e.g., actions that can be realized in the environment); 2) important states of major concepts; 3) objects as realization of concepts; 4) relations to show how concepts and objects are related to each other; 5) self-adapting scenarios for the system in question, e.g., eventual problematic situations with desired outcome; 6) remarkable behavior in terms of policies driving the system out of specific situations; 7) other important specifics that can be classified as concepts (could be explicit) and objects, e.g., SLO (service-level objectives), QoS properties, system sensors, group formations, etc. With KnowLang we specify *knowledge models* of the internal and external "worlds" of a cyber-physical system at special ontology and logic foundation levels. Those models are used by the KnowLang Reasoner to realize artificial awareness.

When we specify knowledge with KnowLang, we build a KB with a variety of knowledge structures such as *ontologies*, *facts*, *rules* and *constraints* where we need to specify the ontologies first in order to provide the "vocabulary" for the other knowledge structures. A KnowLang ontology is specified over *concept trees* (data classes, similar to classes in a *domain model*), *object trees*, *relations* and *predicates* (see Figure 1). Each concept is specified with special properties and functionalities and is hierarchically linked to other concepts through PARENTS and CHILDREN relationships. In addition, for reasoning purposes every concept specified with KnowLang has an intrinsic STATES attribute that may be associated with a set of possible state values the concept instances might be in [9, 10]. The concept instances are considered as objects and are structured in *object trees*. The latter are a conceptualization of how objects existing in the world of interest are related to each other. The relationships in an object tree are based on the principle that objects have properties, where the value of a property is another object, which in turn also has properties. Figure 1 depicts the graphical representation of a concept tree specified with KnowLang.

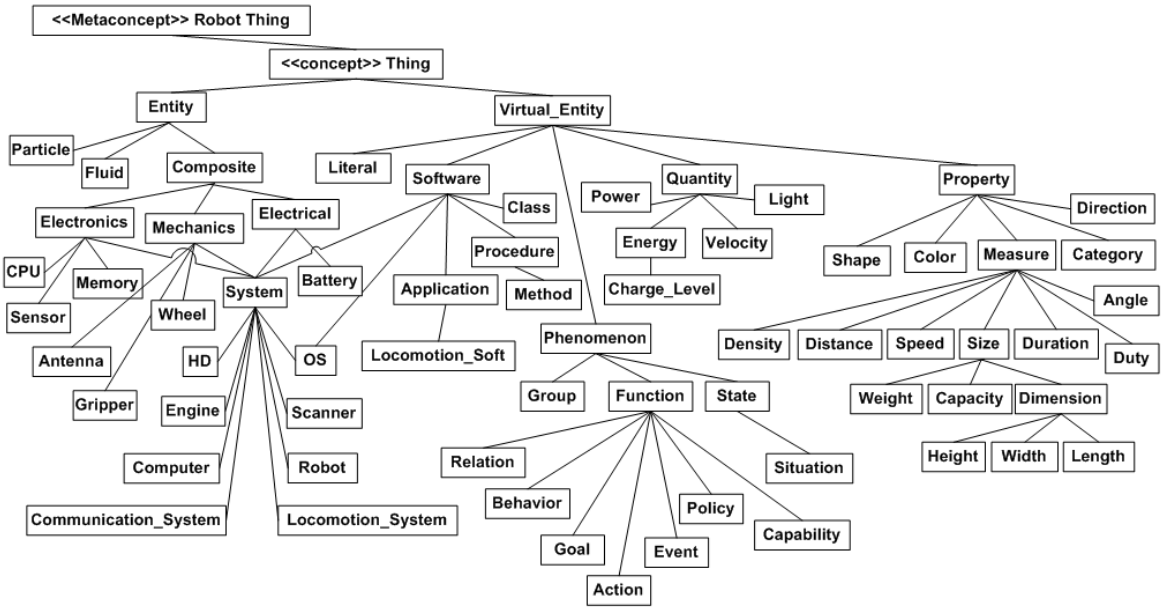

Fig. 1. KnowLang Ontology Specification Sample

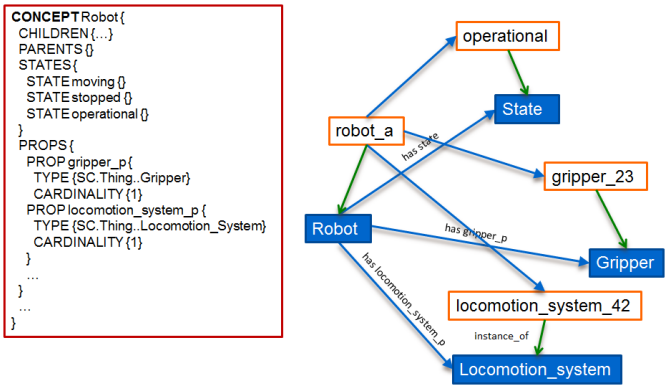

Fig. 2. KnowLang Specification Sample – Concept & Concept Map

In KnowLang, concepts and objects might be connected via *relations* expressing *relation requirements*. Relations connect two concepts, two objects, or an object with a concept and may have *probability-distribution attribute* (e.g., over time, over situations, over concepts' properties, etc.). Probability distribution is provided to support probabilistic reasoning and by specifying relations with probability distributions we actually specify Bayesian networks connecting the concepts and objects of an ontology. Figure 2 shows a KnowLang specification sample demonstrating both the language syntax [11] and its visual counterpart - a concept map based on inter-relations with no probability distributions.

IV. Modeling Awaraness

To function, the mechanism implementing awareness must be structured taking into consideration possible different stages of the *awareness process*. The mechanism of awareness can be built over a complex *chain of functions* pipelining the stages of the awareness process such as [12]: 1) raw data gathering; 2) data passing; 3) filtering; 3) conversion; 4) assessment; 5) projection; and 6) learning. As shown in Figure 3, ideally all the awareness functions can be structured as a *Pyramid of Awareness* forming the mechanism that converts raw data (facts, measures, raw events, etc.) into conclusions, problem prediction and eventually may trigger learning. The different pyramid levels represent awareness functions that can be grouped into four function groups determining specific awareness tasks [12]. The first three pyramid levels compose the group of *monitoring tasks*.

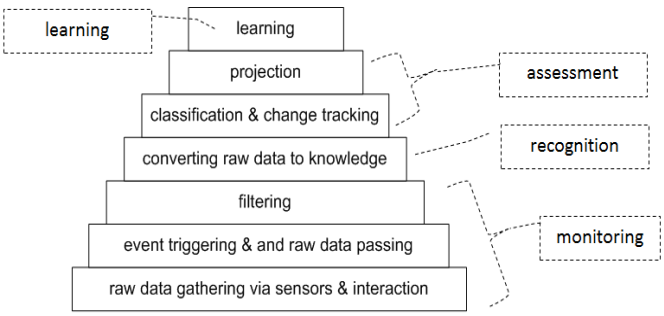

Fig. 3. The Pyramid of Awareness

Further, the fourth level forms the group of *recognition tasks*. The fifth and the sixth levels compose the group of *assessment tasks*, and finally, the last seventh level form the group of *learning tasks*. In addition, aggregation can be included as a subtask at any function level. Note that aggregation is intended to improve the overall awareness performance, e.g., aggregation techniques can be applied to aggregate large amounts of sensory data during the filtering stage, or can be applied by the recognition tasks to improve classification.

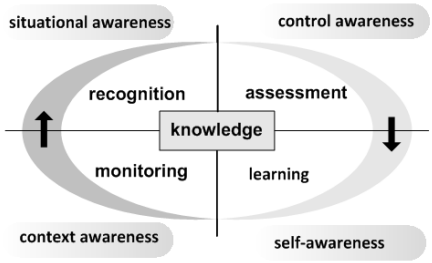

Fig. 4. Awareness Control Loop

Ideally, the four awareness function groups require a comprehensive and well-structured KB representing knowledge in KR Symbols expressing the system itself with its proper internal structures and functionality and the environment. Moreover, the awareness process is not as straightforward as one might think. Instead, it is a *cyclic* with many iterations over the awareness functions. Thus, by closing the chain of awareness functions we form a special *awareness control loop* [13] where different classes of awareness may emerge (see Figure 4).

Its cycling nature is the main reason to regard awareness as complex product with several levels of exhibition and eventually degree of awareness. The levels of awareness might be related to data readability and reliability, i.e., it could happen to have noisy data that must be cleaned up and eventually interpreted with some degree of probability. Other levels of awareness exhibition might be early awareness, which is supposed to be a product of one or two passes of the Awareness Control Loop and late awareness, which should be more mature in terms of conclusions and projections. Similar to humans who may react to their first impression and then the reaction might shift together with a late but better realization of the current situation, an aware computerized system should rely on early awareness to react quickly to situations when fast reaction is needed and on late awareness when more precise thinking is required. Ideally, awareness should be a part of the cognitive process where it might support learning. An efficient awareness mechanism should rely on both past experience and new knowledge introduced to the system.

V. Implementing Awaraness

To build an efficient awareness mechanism, we need to think how to properly integrate the Pyramid of Awareness within the KnowLang Framework. The baseline is to provide a means of monitoring and KR with proper reasoner supporting the Pyramid of Awareness. In this approach, the KnowLang Reasoner is supplied as a component hosted by a cyber-physical system and thus, it runs in the system's operational

context as any other system's component. However, it operates in a KR Context (formed by the knowledge models) and on the KR symbols (represented knowledge). The system talks to the reasoner via special ASK and TELL Operators allowing for knowledge querying and update, respectively (see Figure 3). Upon demand, the KnowLang Reasoner can also build up and return a self-adaptive behavior model (e.g., to react to a particular situation) consisting of a chain of actions to be realized in the environment or within the system itself.

TELL Operators feed the KR Context with important information driven by *errors*, *executed actions*, *new sensory data*, etc. (see Figure 3), thus helping the KnowLang Reasoner update the KR with recent changes in both the system and execution environment. The system uses ASK Operators to receive *recommended behavior* where knowledge is used against the perception of the world to generate appropriate actions in compliance to some goals and beliefs. In addition, ASK Operators may provide the system with awareness-based conclusions about the current state of the system or the environment and ideally with behavior models for self-adaptation. The following presents generic algorithms of how both classes of KB Operators operate with knowledge.

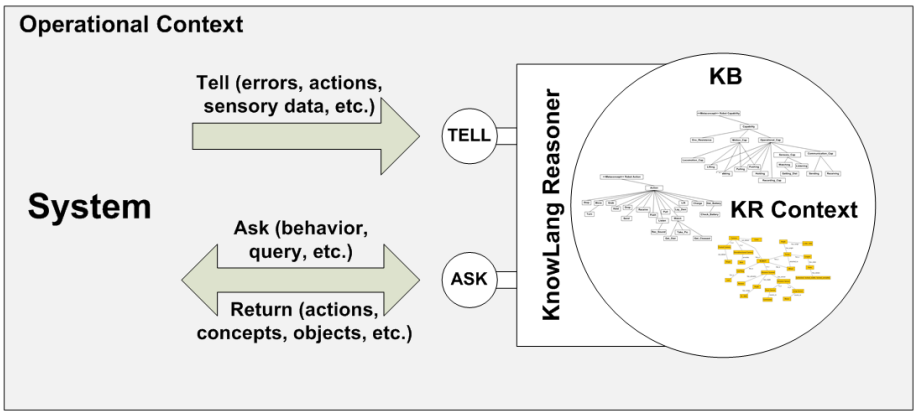

Fig. 5. KnowLang Reasoner

TELL Algorithm:

1. The system tells the KnowLang Reasoner about errors, sensory data, execution of actions or actual updates.
2. The KnowLang Reasoner switches to the KR Context and maps the input to KR symbols.
3. The KnowLang Reasoner updates the KB, e.g., updates concepts / objects or adds new concepts/objects and changes states.

ASK Algorithm:

1. The system asks for a self-adaptive behavior, rule-based behavior, current state or current situation.
2. The KnowLang Reasoner switches to the KR Context and maps the input to KR symbols.
3. The KnowLang Reasoner processes the query to get behavior actions or retrieve information and eventually updates the KB.
4. The KnowLang Reasoner builds the output and returns the result to the system.

Currently, KnowLang is equipped with the following TELL and ASK Operators [14]:

- TELL_ERR - tells about a raised error;
- TELL_SENSOR - tells about new data collected by a sensor;
- TELL_ACTION - tells about action execution;
- TELL_ACTION(behavior) - tells about action execution as part of behavior performance;
- TELL_OBJ_UPDATE - tells about a possible object update;
- TELL_CNCPT_UPDATE - tells about a possible concept update;
- ASK_BEHAVIOR - asks for self-adaptive behavior considering the current situation;
- ASK_BEHAVIOR(goal) - asks for self-adaptive behavior to achieve certain goal;
- ASK_BEHAVIOR(situation; goal) - asks for self-adaptive behavior to achieve certain goal when departing from a specific situation;
- ASK_BEHAVIOR(state) - asks for self-adaptive behavior to go to a certain state;
- ASK_RULE_BEHAVIOR(conditions) - asks for rule-based behavior;
- ASK_CURR_STATE(object) - asks for the current state of an object;
- ASK_CURR_STATE - asks for the current system state;
- ASK_CURR_SITUATION - asks for the current situation.

For example, when called the ASK_BEHAVIOR Operator will ask the Reasoner to generate a self-adaptive behavior by considering the actual situation the system is currently in. Thus, it looks up for the *current situation* by estimating the *current system state* and then evaluates the relations of that situation with special policies to determine which policy to apply. There are also other variants of the ASK_BEHAVIOR Operator, e.g., the system may ask for a self-adaptive behavior to achieve a *particular goal* or a behavior that will lead the system out of a particular situation. Note that the ASK_BEHAVIOR operates exclusively in the KR Context (see Figure 3) and thus, the relevance of its output highly depends on the relevance of the knowledge stored in the KB. Thus, it is very important that the system feeds the KB with any important relevant information about the system itself and the execution environment, e.g., errors, sensory data, raised events, executed actions, etc.

A. Navigation Awaraness

To demonstrate the approach described above, let us think about an autonomous mobile robot (it can move and act without human interference) using a special *navigation awareness* to move through natural, unstructured terrain. In our approach, *navigation awareness* requires context-relative plots of position that permit inference of robot speed and direction. Thus, landmarks should be represented as part of the KB or eventually scanned by the robot's sensors and reported to the KnowLang Reasoner via TELL_SENSOR Operator calls. Moreover, at the beginning of the navigation process a special "*navigation map*" should be built on the fly by the KnowLang Reasoner by using KB symbols. The ASK_BEHAVIOR(goal) Operator shall trigger the process of building the navigation map where the goal could be "*moving from point to point*" or simply "*moving south*". Then, navigation awareness is reading the sensor data from cameras and plotting the position of the robot at the time of observation achieved via TELL_SENSOR Operator calls. Via repeated position plots, the course and land-reference speed of the robot is established. For example, speed can be reported to the system via the ASK_CURR_STATE(speed) Operator.

B. Awaraness Based on Self-initiation

In addition to the awareness abilities initiated via ASK and TELL operators, we envision an additional awareness capability based on *self-initiation* where the KnowLang Reasoner may initiate actions without being asked for it. In this approach, we consider a behavior model based on the so-called Partially Observable Markov Decision Processes (POMDP) [15]. Note that this model is appropriate when there is uncertainty and lack of information needed to determine the state of the entire system. For example, individuals in complex systems like swarms of robots often might be idle, i.e., not actively participating in the swarm's activities, because they are not certain about the current swarm state. Thus, the POMDP model helps a robot reason on the current swarm state (or that of the environment) and eventually self-initiate when an action is needed to be performed. According to our POMDP-based model, a swarm robot takes as input observable situations, involving other swarm robots and the environment, and generates as output actions initiating robot activity. Note that the generated actions affect the global swarm state.

Formally, this model is a tuple

$$\mathbf{M :=< S; A; T; R; X; O >}$$

where:

- **S** is a finite set of states of the system that are not observable.
- An initial belief state $\mathbf{s_0 \in S}$ is based on $\mathbf{z_0\ (s_0;\ s_0 \in S)}$, which is a discrete probability distribution over the set of system states **S**, representing for each state the robot's belief that is currently occupying that state.
- **A** is a finite set of actions that may be undertaken by the robot. Note that the system state determines the current situation and thus, the possible set of actions is reduced to coop with that situation (or respectively state).
- $\mathbf{T : S \times A \rightarrow Z(S)}$ is the state transition function, giving for each system state s and robot action a, a probability distribution over states. Here, **T(s; a; s')** computes the probability of ending in state **s'**, given that the start state is s and the robot takes action a; **z(s'|s; a)**.
- $\mathbf{O : A \times S \rightarrow Z(X)}$ is the observation function giving for each system state **s** and robot action **a**, a probability distribution over observations **X**. For example, **O(s'; a; x)** is the probability of observing **x**, in state **s'** after taking action **a, z(x|s'; a)**.
- $\mathbf{R : S \times A \rightarrow}$ **R** is a reward function, giving the expected immediate reward gained by the robot for taking an action in a state s, e.g., **R(s; a)**. The reward is a scalar value in the range [0..1] determining, which action (among many possible) should be undertaken by the robot in compliance with the swarm goals.

To illustrate this model, let's assume that a *swarm of robots*, moving together heavy objects around, is currently occupying the state **s** = "*new object to be moved is discovered, but no moving team has been formed yet and still no other robot has self-initiated for team formation*". Let's assume there is at least one idle robot in the swarm ready to undertake a few actions **A**, including the action **a** = "*self-initiation for team formation*". The robot performs the following reasoning steps in order to self-initiate for team formation:

1. The robot computes its current belief state $\mathbf{s_0}$ - the robot picks up the state with the highest probability $\mathbf{z_0}$ and eventually $\mathbf{s_0 = s}$.
2. The robot computes the probability $\mathbf{z_1}$ of the swarm occupying the state **s'** = "*new object is discovered and a robot has self-initiated for team formation*" if the action **a** is undertaken from state $\mathbf{s_0}$.
3. The robot computes the probability $\mathbf{z_2\ (x|s';\ a)}$ of observation **x** = "*there are sufficient numbers of idle robots to form a new exploration team*".
4. The robot computes the reward $\mathbf{r\ (s_0;\ a)}$ for taking the action **a** (self-initiation for team formation) in state $\mathbf{s_0}$. If no other immediate actions should be undertaken (forced by other swarm goals), the reward

r should be the highest possible, which will determine the execution of **a**.

VI. MEETING THE CHALLENGES

The KnowLang framework considers artificial awareness as an introduction of machine intelligence (or AI – Artificial Intelligence) into the systems' command and control operations. The approach might be regarded as a four-part model (see Section IV) where monitoring, recognition, assessment and learning operations form a control loop delivering awareness about situations and needed actions to be undertaken to avoid damages and operational malfunction. The nature of the problems the framework needs to deal with introduces many challenges, some of which are discussed in this section.

A. Encoded versus Represented Knowledge

Recall that the KnowLang Reasoner operates as a component hosted by a functional system. Thus, developers may encode a large part of the "a priori" 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's classes and routines. For example, KnowLang actions could be based on classes and methods and a substantial concern about the KR of such actions is how to relate the knowledge expressed with actions to implemented methods and functions. A possible solution is to map KR concepts and KR objects to program classes and program objects respectively.

To properly represent the program implementation (classes, methods, etc.) in the KB, KnowLang supplies all the specified concepts and objects with an IMPL Property that relates a KnowLang structure to its program counterpart, if any. For example, a KnowLang concept might be specified with an IMPL property to link the concept to a program class or method. The following is the KnowLang Grammar's definition supporting that [14].

Concept-Impl → IMPL { Impl-Reference }

B. States, Situations, Goals and Policies

A big challenge is how to express situations and reason about the same. Situation-sensitivity has a direct impact on the artificial awareness. Ideally, situations can trigger self-adaptive behavior acquired by KnowLang POLICIES.

Definitions 1 through 9 express the relationships between policies and situations. As shown, the KnowLang policies Π drive the behavior of the system. A policy π has a goal g, policy situations Si_π, policy-situation relations R_π, and policy conditions N_π mapped to policy actions A_π where the evaluation of N_π may eventually (with some degree of probability) imply the evaluation of actions (denoted with $N_\pi \xrightarrow{[Z]} A_\pi$). To support this approach along with situation recognition, KnowLang has introduced a STATE explicit concept. This helps us specify every KnowLang concept with a set of important states the concept instances can be in. Thus, we explicitly specify a variety of states for important concepts (e.g., states "operational" and "non-operational" for a robot's Motion System). A KnowLang state is specified as a *Boolean expression over ontology* (see Definition 7) where we can use activation of events, execution of actions or changes in properties to build a state's Boolean expression [14]. Further, to facilitate the evaluation of complex states, we specify PREDICATE concepts. Complex states (e.g., system states) are the product of other states (e.g., the states of the system's components). States (usually system states) are also used to specify SITATUONS (see Definition 9) and GOALS (see Definition 5), another class of KnowLang explicit concepts. Recall that KnowLang goals participate in the specification of KnowLang POLICIES. A goal can be specified as a transition from a state to another. Moreover, policies and situations participate in special policy RELATIONS that derive self-adaptive behavior [9, 10]. Therefore, because every situation is explicitly related to a state (a situation is determined by a state), it is relatively easy to check for the feasibility of a policy triggered by a specific situation, i.e., the policy's goal must have the same departing state as the situation's state.

$$\Pi := \{\pi_1, \pi_2, \dots, \pi_n\}, n \geq 0 \quad (Policies) \quad (1)$$

$$\pi := \{g,\ Si_\pi, [R_\pi], N_\pi,\ A_\pi,\ map(N_\pi,\ A_\pi, [Z])\} \quad (Policy) \quad (2)$$

$$A_\pi \subset A_{si},\ N_\pi \xrightarrow{[Z]} A_\pi \quad (A_\pi - Policy\ Actions)$$

$$Si_\pi \subset Si, Si_\pi \xrightarrow{[R_\pi]} \pi \longrightarrow N_\pi \quad (Si_\pi - Policy\ Situations)$$

$$R_\pi \subset R, r_\pi := \{\pi, [rn], [Z], si_\pi\} \quad (R_\pi - Policy - Situation\ Realtion)$$

$$N_\pi := \{n_0, n_1, \dots, n_n\} \quad (Policy\ Conditions) \quad (3)$$

$$n := be(O) \quad (Condition - Boolean\ Expression\ over\ Ontology) \quad (4)$$

$$g := (\Longrightarrow s' \mid s \Longrightarrow s') \quad (Goal) \quad (5)$$

$$\Longrightarrow s := \langle TELL \rhd KB\rangle \mid [\langle ASK \rhd KB\rangle] \quad (State\ Transition - execution\ of\ TELL/ASK\ operator\ on\ KB) \quad (6)$$

$$s := be(O) \quad (State - Boolean\ Expression\ over\ Ontology) \quad (7)$$

$$Si := \{si_0, si_1, \dots, si_n\} \quad (Situations) \quad (8)$$

$$si := \{s, A_{si}^{\leftarrow}, [E_{si}^{\leftarrow}],\ A_{si}\}, n \geq 0 \quad (Situation) \quad (9)$$

$$A_{si}^{\leftarrow} \subset A \quad (A_{si}^{\leftarrow} - History\ of\ Executed\ Actions)$$

$$A_{si} \subset A \quad (A_{si} - Possible\ Actions)$$

$$E_{si}^{\leftarrow} \subset E \quad (E_{si} - History\ of\ Situation\ Events)$$

C. Converting Sensory Data to KR Symbols

Another considerable challenge is how to map raw sensory data to KR symbols. Our approach to this problem is to specify special explicit concepts called METRICS. In general, a SCE system has sensors that connect it to the world and eventually help it to listen to its internal components. These sensors generate raw data that represent the physical characteristics of the world. The problem is that these low-level data streams must be: 1) converted to programming variables or more complex data structures that represent collections of sensory data; 2) those programing data structures must be labeled with KR symbols. Hence, it is required to relate encoded data structures with KR concepts and objects used for reasoning purposes. In our approach, we assume that each sensor is controlled by a software driver (e.g., implemented in Java) where appropriate methods are used to control the sensor and read data from it. Both the sensory data and sensors should be represented in the KB by using METRIC concepts and instantiate objects of these concepts. By specifying a METRIC concept we introduce a class of sensors to the KB and by specifying objects, instances of that class, we give the actual KR of a real sensor. KnowLang allows the specification of four different types of metrics [14]:

- RESOURCE - measure system resources;
- QUALITY - measure system's qualities like performance, response time, etc.;
- ENVIRONMENT - measure environment's qualities and resources;
- ENSEMBLE - measure qualities addressed by multi-agent systems; might be a function of multiple metrics of RESOURCE and QUALITY type.

VII. Conclusion and Future Work

Machine intelligence, incorporated in a cyber-physical system, depends on the ability to perceive the environment and react to changes. The Pyramid of Awareness and the associated Awareness Control Loop provide for a mechanism were this ability exhibits as a result of special enchained functions working on raw data (gathered via system's sensors) to recognize objects, project situations, track changes and learn new facts. The Awareness Control Loop can exhibit awareness at different levels of maturity and relevance. The KnowLang Reasoner provides both the knowledge models and computational structures implementing the awareness mechanism. The reasoner operates in a special KR Context and the system talks to the reasoner via special ASK and TELL operators forming a communication interface. In this approach knowledge is used against the perception of the world to generate appropriate artificial awareness capabilities. Noisy data may introduce probability into the awareness relevance and the number of iterations in the control loop may result into early and late awareness. Ideally, the approach helps an intelligent system behave like humans, who realize situations and changes in a progressive manner and react progressively. Often the first impression (early awareness) triggers a reaction, whose course can be changed if the progressive realization of the current situation decides so.

Note that KnowLang is still under development as part of the ASCENS FP7 project (http://www.ascens-ist.eu/). Our plans are to completely develop KnowLang including a toolset for formal validation. Once fully implemented, KnowLang will be used to specify knowledge representation and develop artificial awareness in different case studies.

Acknowledgment

This work was supported by the European Union FP7 Integrated Project Autonomic Service-Component Ensembles (ASCENS) and by Science Foundation Ireland grant 03/CE2/I303_1 to Lero - the Irish Software Engineering Research Centre at University of Limerick, Ireland.

References

[1] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems", in International Journal of Ad Hoc Ubiquitous Computing, vol. 2(4), pp. 263-277, 2007.

[2] Q. Z. Sheng et al., "ContextServ: A platform for rapid and flexible development of context-aware web services", in Proceedings of the 31st International Conference on Software Engineering, IEEE Computer Society, pp. 619-622, 2009.

[3] E. Serral, P. Valderas, and V. Pelechano, "Towards the Model Driven Development of context-aware pervasive systems", in Pervasive and Mobile Computing, vol. 6(2), pp. 254-280, 2010.

[4] C. Bettini et al., "A survey of context modelling and reasoning techniques", in Pervasive and Mobile Computing, vol. 6(2), pp. 161-180, 2010.

[5] C. Galindo, J. Fernandez-Madrigal, J. Gonzalez, and A. Saffiotti, "Robot task planning using semantic maps", Robotics and Autonomous Systems, 56(11), pp. 955–966, 2008.

[6] O. Mozos, P. Jensfelt, H. Zender, G.-J. M. Kruijff, and W. Burgard, "An integrated system for conceptual spatial representations of indoor environments for mobile robots", in Proceedings of the IROS 2007 Workshop: From Sensors to Human Spatial Concepts (FS2HSC), pp. 25–32, 2007.

[7] G.-J. M. Kruijff, P. Lison, T. Benjamin, H. Jacobsson, and N. Hawes, "Incremental, multi-level processing for comprehending situated dialogue in human-robot interaction", in Proceedings of the Symposium on Language and Robots, 2007.

[8] H. Holzapfel, D. Neubig, and A. Waibel, "A dialogue approach to learning object descriptions and semantic categories", Robotics and Autonomous Systems, vol. 56(11), pp. 1004–1013, 2008.

[9] E. Vassev and M. Hinchey, "Knowledge representation and reasoning for self-adaptive behavior and awareness", in LNCS Transactions on Computational Collective Intelligence- Special Issue on ICECCS 2012, 2013.

[10] E. Vassev and M. Hinchey, "Knowledge representation for cognitive robotic systems", in Proceedings of the 15th IEEE International Symposium on Object/Component/Service-oriented Real-time Distributed Computing Workshops (ISCORCW 2012), pp. 156–163, 2012.

[11] E. Vassev, KnowLang Grammar in BNF, Technical Report Lero-TR-2012-04, Lero, University of Limerick, Ireland, 2012.

[12] E. Vassev and M. Hinchey, "Awareness in software-intesive systems", IEEE Computer, vol. 45(12), pp. 84-87, 2012.

[13] E. Vassev and M. Hinchey, "The challenge of eveloping autonomic systems", IEEE Computer, vol. 43(12), pp.93–96, 2010.

[14] E. Vassev, "Operational semantics for KnowLang ASK and TELL operators", Technical Report Lero-TR-2012-05, Lero, University of Limerick, Ireland, 2012.

[15] M. L. Littman, Algorithms for sequential decision making, PhD Thesis, Department of Computer Science, Brown University, 1996.