Software Technologies

Awareness in Software-intensive Systems

Emil Vassev and Mike Hinchey

Lero—the Irish Software Engineering Research Centre

Closely related to artificial Intelligence, awareness depends on the knowledge transferred to software-intensive systems so they can use it to exhibit intelligence.

Conceptually, awareness is a product of knowledge and monitoring.

A large class of software-intensive systems—including those for industrial automation, consumer electronics, airplanes, automobiles, medical devices, and civic infrastructure—must interact with the physical world. More advanced systems, such as unmanned autonomous systems, don't just interact but also perceive important structural and dynamic aspects of their operational environment. To become interactive, an autonomous software- system must be aware of its physical environment and whereabouts, as well as its current internal status. This ability helps intelligent software systems sense, draw inferences, and react.

Closely related to artificial intelligence, awareness depends on the knowledge we transfer to software systems so they can use it to exhibit intelligence. In addition to knowledge, *artificial awareness* also requires a means of sensing changes so that the system can perceive both external and internal worlds through raw events and data.

Thus, self- and environment monitoring are crucial to awareness: to exhibit awareness, software-intensive systems must sense and analyze their internal components and the environment in which they operate. Such systems should be able to notice a change and understand its implications. Moreover, an aware system should apply both pattern analysis and pattern recognition to determine normal and abnormal states.

Ideally, awareness should be part of the cognitive process that underlies learning. An efficient awareness mechanism should also rely on both past experience and new knowledge. Awareness via learning is the basic mechanism for introducing new facts into the cognitive system—other possible ways are related to interaction with a human operator who manually introduces new facts into the knowledge base.

The aim of this article is to clarify and capture the nature of artificial awareness and its impact on contemporary software-intensive systems. A successful artificial awareness mechanism is critical in the construction of autonomous robotic systems able to perceive their environment and react to changes autonomously. This article clarifies implementation issues, and hence improves the understanding of artificial awareness.

CLASSES OF AWARENESS

Awareness generally is classified into two major areas: self-awareness, pertaining to the internal world, and context-awareness, pertaining to the external world. Autonomic computing research defines these two classes *("Autonomic Computing: IBM's Perspective on the State of Information Technology," IBM Autonomic Computing Manifesto 2001; http://www.research.ibm.com/autonomic/manifesto /autonomic_computing.pdf*):

- A self-aware system has detailed knowledge about its own entities, current states, capacity and

capabilities, physical connections, and ownership relations with other systems in its environment.

- A context-aware system knows how to sense, negotiate, communicate, and interact with environmental systems and how to anticipate environmental system states, situations, and changes.

Perhaps a third class could be situational awareness, which is self-explanatory; other classes could draw attention to specific problems, such as operational conditions and performance (operational awareness), control processes (control awareness), interaction processes (interaction awareness), and navigation processes (navigation awareness). Although classes of awareness can differ by subject, they all require a subjective perception of events and data "within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future" (M.R. Endsley, "Toward a Theory of Situation Awareness in Dynamic Systems," *Human Factors*, vol. 37, no. 1, 1995, pp. 32-64).

To better understand the idea of awareness in software-intensive systems, consider an exploration robot. Its navigation awareness mechanism could build a map on the fly, with landmarks represented as part of the environment knowledge, so that navigation becomes simply a matter of reading sensor data from cameras and plotting the robot's position at the time of observation. Via repeated position plots, the robot's course and land-reference speed can be established.

STRUCTURING AWARENESS

Recent research efforts have focused on awareness implementations in software-intensive systems. For example, commercially available server-monitoring platforms, such as Nimbus (www.nimbusproject.org) and Cittio's Watch Tower (*www.networkcomputing.com/data-protection/cittios-watchtower-30/229611534*), offer robust, lightweight sensing and reporting capabilities across large server farms. Such solutions are oriented toward massive data collection and performance reporting, so they leave much of the final analysis and decision making to a human administrator. Other approaches achieve awareness through model-based detection and response based on offline training and models constructed to represent different scenarios that the system can recognize at runtime.

To function, the mechanism implementing the awareness must be structured to take into consideration different stages—for example, it might be built over a complex chain of functions such as *raw data gathering*, *data passing*, *filtering*, *conversion*, *assessment*, *projection*, and *learning*. As Figure 1 shows, ideally, all the awareness functions could be structured as an awareness pyramid, forming the mechanism that converts raw data into conclusions, problem prediction, and eventually learning.

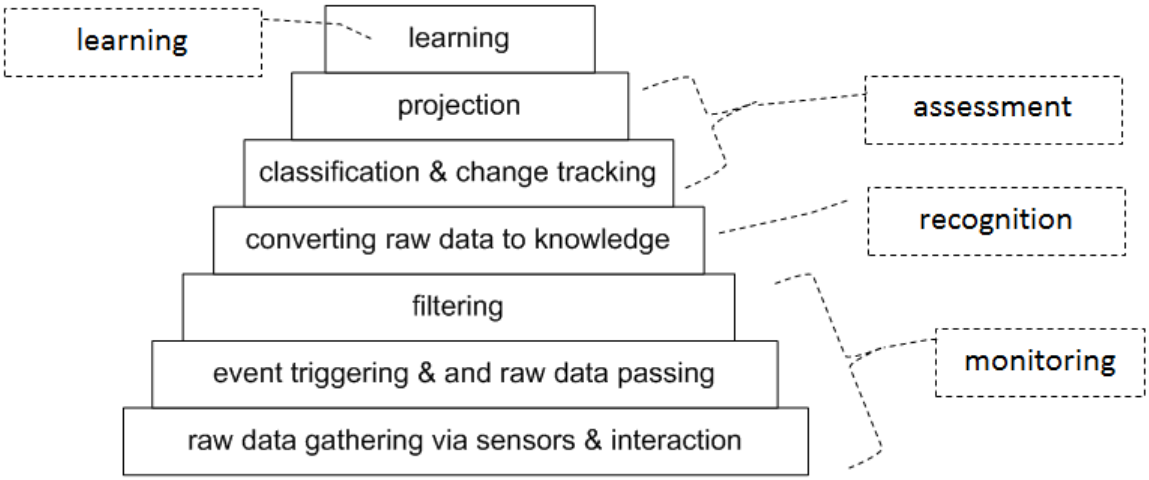

Figure 1. The awareness pyramid. The first three levels include monitoring tasks; the fourth, recognition tasks; the fifth and sixth, assessment tasks; and the last, learning tasks.

The pyramid levels in Figure 1 represent awareness functions that can be grouped into four specific tasks:

- *monitoring* collects, aggregates, filters, manages, and reports internal and external details such as metrics and topologies gathered from the system's internal entities and its context;
- *recognition* uses knowledge structures and data patterns to aggregate and convert raw data into knowledge symbols;
- *assessment* tracks changes and determines points of interest, generates hypotheses about situations involving these points, and recognizes situational patterns; and
- *learning* generates new situational patterns and maintains a history of property changes.

Aggregation can be included as a subtask at any function level; it's intended to improve overall awareness performance. For example, it can pull together large amounts of sensory data during the filtering stage or recognition tasks can apply it to improve classification.

The awareness process isn't as straightforward as it might seem—rather, it's cyclic, with several iterations over the various awareness functions. Closing the chain of awareness functions can form an *awareness control loop* in which different awareness classes can emerge (E. Vassev and M. Hinchey, "The Challenge of Developing Autonomic Systems," *Computer*, Dec. 2010, pp. 93-96).

The process's cyclic nature is why awareness itself is so complex, with several levels of exhibition and degrees of perception. The levels can be related to data readability and reliability—that is, they might include noisy data that must be cleaned up and eventually interpreted with some degree of probability. Other levels might include early awareness, which is 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 react to their first impression but then find that a later and better realization of the situation shifts their reaction, an aware software 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.

IMPLEMENTING AWARENESS

The four awareness functions require a comprehensive and well-structured *knowledge base* (KB) to hold *knowledge representation* (KR) symbols that can express the system itself with its proper internal structures and functionality as well as the environment (*E. Vassev and M. Hinchey, "Knowledge Representation and Reasoning for Intelligent Software Systems," Computer, Apr. 2011, pp. 96-99*).

Building an efficient awareness mechanism requires properly integrating the awareness pyramid within the implemented software system. The goal is to provide a means of monitoring and KR with a reasoner supporting awareness reasoning. KR adds a new open-world KR context to the program, and the reasoner operates in this context, taking into account the monitoring activities that drive the awareness control loop and deliver awareness results to the system itself.

Figure 2 depicts an approach the Autonomic Service-Component Ensembles (ASCENS) FP7 European Project is using to tackle the problem (*www.ascens-ist.eu*). In this approach, KnowLang, a special KR language, provides the constructs and mechanisms for specifying knowledge models at the ontology and logic foundation levels (*Emil Vassev and Michael 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). IEEE Computer Society, 2012, pp. 156-163.*). To specify knowledge with KnowLang, we 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; and 7) other important specifics that can be classified as concepts. As shown in Figure 2, the KB comprises KR structures such as concept trees, object trees, concept maps, etc. The system talks to the KnowLang reasoner via a predefined set of TELL and ASK operators forming a sort of communication interface connecting both the system and KB. TELL operators feed the KB with important information driven by errors, executed actions, new sensory data, and so forth,

thus helping the KnowLang reasoner update the KR Context with recent changes in both the system and execution environment. The system uses ASK operators to receive recommended behavior where knowledge is used against perception to generate appropriate actions in compliance with specific goals and beliefs. In addition, ASK operators can provide awareness-based conclusions about the system or environment's current state and ideally with behavior models for reaction.

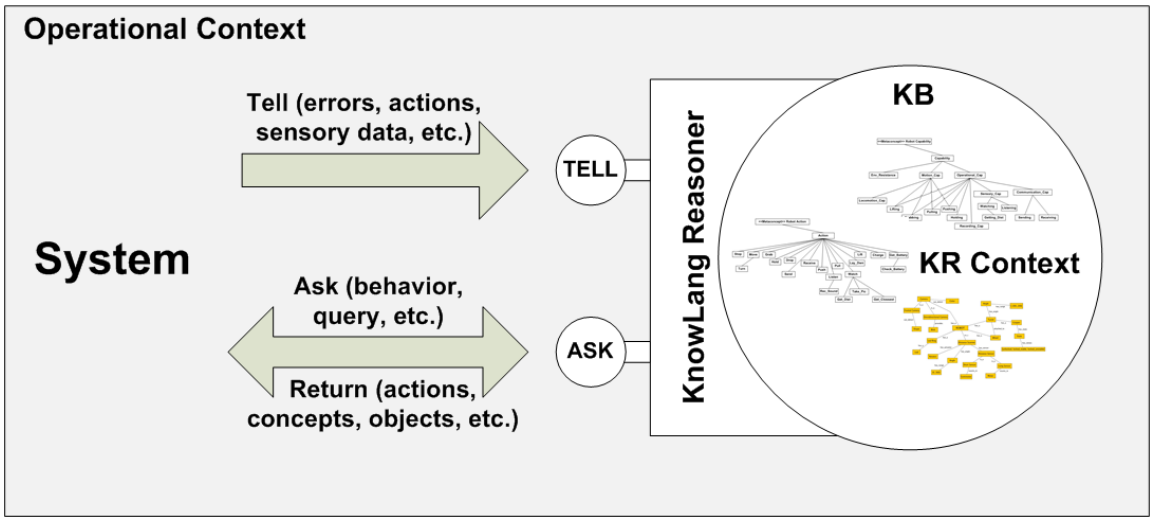

Figure 2. Implementing awareness

The following presents generic algorithms of how the KnowLang reasoner processes ASK and TELL operators:

TELL Algorithm

1. The system tells the KnowLang Reasoner about errors, sensory data, and execution of actions or actual updates.
2. The reasoner switches to the KR context and maps the input to KR symbols (concepts, objects, relations, etc.).
3. The reasoner updates the KB, e.g., it updates concepts/objects or adds new concepts/objects and changes states.
4. The reasoner draws awareness conclusions based on the new state changes and if necessary self-initiate for action.

ASK Algorithm

1. The system asks for behavior, current states or situations.
2. The reasoner switches to the KR context and maps the input to KR symbols.
3. The reasoner processes the query to get behavior actions or retrieve information (e.g., awareness conclusions) and eventually updates the KB.
4. The reasoner builds the output and returns the result to the system.

Note that in addition to the awareness abilities initiated via ASK and TELL operators, we can envision additional awareness capability based on self-initiation where the reasoner may initiate actions of its own based on state changes if those violate system's goals. For example, the system may decide to switch to an energy-saving mode if the current state is related to insufficient energy supply.

One of the biggest challenges in this approach is how to map sensory raw data to KR symbols. An aware software-intensive 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; and 2) those programing data structures must be labeled with KR symbols. Hence, it is required to relate encoded data structures with KR

symbols used for reasoning purposes.

Another considerable challenge is how to express states and reason about the same. KnowLang introduces an explicit STATES attribute that helps us to specify concepts with a set of important states the concepts 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. Further, a state in KnowLang is specified as a Boolean expression over ontology where we can use activation of events, execution of actions or changes in properties to build a state’s Boolean expression. To facilitate the evaluation of complex states, the reasoner can use special predicates where complex states (e.g., system states) are evaluated as the product of other states (e.g., the states of the system’s components).

The long-term impact of awareness-related research and development is a roadmap leading towards artificial intelligence. Machine intelligence depends on the ability to perceive the environment and react to changes in it. The awareness mechanism uses raw data gathered via system’s sensors to recognize objects, project situations, track changes, and learn new facts. A successful awareness mechanism can exhibit awareness at different levels of maturity and relevance. Noisy data can affect awareness relevance, which can lead to awareness results gradually changing over time and data input.

Ideally, the awareness **mechanism should** help intelligent systems to behave like humans, realizing situations and reacting progressively where often, the first impression triggers a reaction, which can change based on progressive realization of the current situation.

Emil Vassev *is a research fellow at Lero—the Irish Software Engineering Research Centre at the University of Limerick, Ireland. Contact him at emil.vassev@lero.ie.*

Mike Hinchey *is the director of Lero—the Irish Software Engineering Research Centre and a professor of software engineering at the University of Limerick, Ireland. Contact him at mike.hinchey@lero.ie.*

Editor: Mike Hinchey, Lero—the Irish Software Engineering Research Centre; mike.hinchey@lero.ie

software technologies, software-intensive systems, awareness, artificial intelligence